В.С. Бурцев

Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ

МВК "Эльбрус"

Москва 1998

Всеволод Сергеевич Бурцев

Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ. МВК "Эльбрус"

М.,1998

В книге представлены основные статьи академика В.С. Бурцева, в которых прослеживается путь создания МВК "Эльбрус-1" и "Эльбрус-2" с момента формирования проекта до его завершения -освоения серийного производства и окончания Государственных испытаний.

В книге представлены основные статьи автора, в которых прослеживается путь создания МВК "Эльбрус-1" и "Эльбрус-2" с момента формирования проекта до его завершения - освоения серийного производства и окончания Государственных испытаний.

Даты публикаций статей не всегда совпадают с периодом возникновения тех или иных идей решений и их теоретических обоснований. Так, статья по надежности многопроцессорных комплексов написана после завершения Государственных испытаний МВК «Эльбрус-2», в то время как идеи, изложенные в ней, и модульный принцип построения комплекса были с успехом реализованы в системе С-300 [9] (1967-1972 г.г.) и нашли дальнейшее развитие в проекте МВК «Эльбрус-1» и «Эльбрус-2». Поэтому последовательность представления статей в книге преследует цель лучшей методики изложения материала для его восприятия читателем и не всегда соответствует временной последовательности их публикаций.

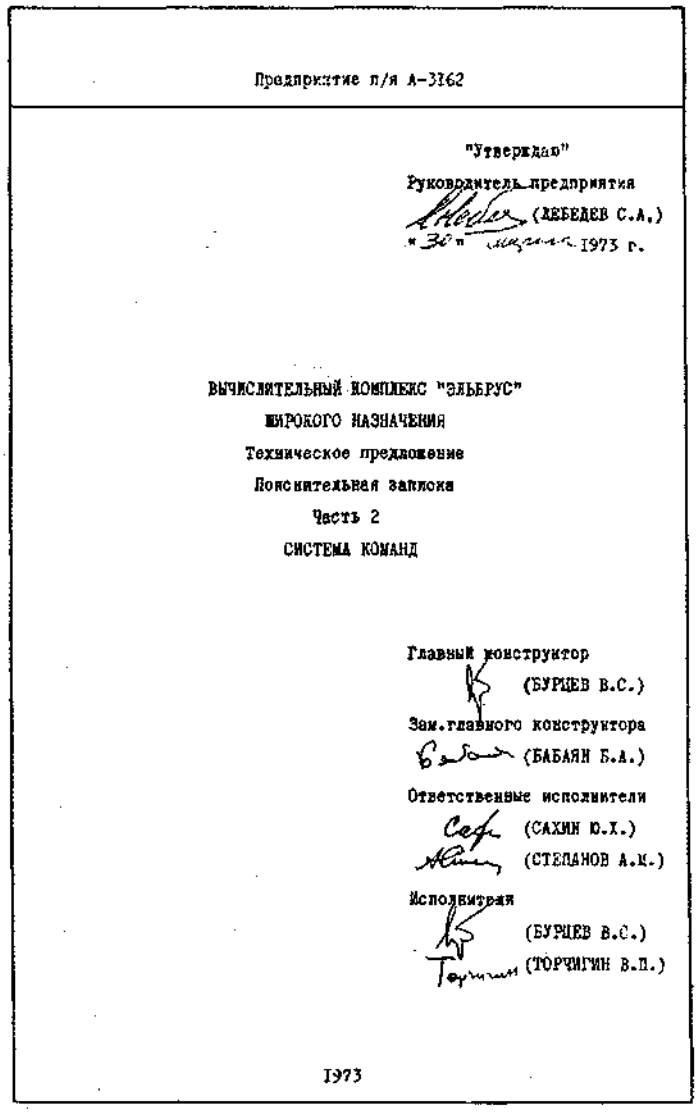
Разработка МВК "Эльбрус" началась фактически с 1968 года. В это время нами были закончены работы по вычислительным комплексам подмосковного ПРО и Генеральный конструктор ПРО Г.В.Кисунько заявил, что для разработки следующего поколения ПРО ему необходима производительность вычислительных средств не ниже 100 млн. операций в секунду. Производительность самых лучших суперЭВМ того времени не превосходила 5 млн. операций в секунду для специального применения. С этого времени мы стали продумывать возможность построения требуемого комплекса. Соображения и обоснования решений того времени (1969-1970 годы) изложены в статьях о перспективах развития высокопроизводительных систем [1, 2]. Первый проект (технические предложения) был выпущен в 1972 году. Раздел системы команд (титульный лист документа показан на странице 4) был написан мной совместно с профессором В.П.Торчигиным и представлен в настоящей книге [4]. В процессе создания МВК "Эльбрус-1" и "Эльбрус-2" эта система команд была уточнена в части оптимизации ее реализаций. Эти изменения никак не сказались на основных принципах построения этих комплексов, поэтому мы не корректировали, например, распределение разрядов по различным полям в словах и ряд других моментов. В некоторых местах внесены комментарии к произошедшим изменениям.

Статья «Принципы построения многопроцессорных вычислительных комплексов "Эльбрус"» [3] написана в период разработки технического проекта по МВК "Эльбрус-2". В книге эта статья для лучшего понимания материала помещена перед статьей «Неформальное описание системы команд» [4].

Принципам работы операционной системы посвящена статья, написанная совместно с В.П.Торчигиным. Она написана значительно позднее и ранее не печаталась [5].

Пять первых статей книги по мнению автора позволяют изучить основные принципы, заложенные в проектах МВК "Эльбрус-1" и "Эльбрус-2", основной целью которых было достижение:

- заданной предельной производительности на имеющейся элементной базе (т = 10 нс для "Эльбрус-1" и т = 2 нс для "Эльбрус-2");
- высокой надежности комплекса и достоверности выдаваемой информации (вероятность выполнения боевого цикла P = 0,999) при относительно малой надежности элементной базы ($\lambda = 10^{-6}$);
- удобства программирования и отладки как системных программ, так и программ пользователя.



Предприятие п/я А-3162

"Утверждаю"
Руководитель предприятия
(ЛЕБЕДЕВ С.А.)
"30" 1973 г.

ВЫЧИСЛИТЕЛЬНЫЙ КОМПЛЕКС "ЭЛЬБРУС"
ШИРОКОГО НАЗНАЧЕНИЯ
Техническое предложение
Пояснительная записка
Часть 2
СИСТЕМА КОМАНД

Главный конструктор
(БУРЦЕВ В.С.)
Зам.главного конструктора
(БАБАЯН Б.А.)
Ответственные исполнители
(САХИН Ю.Х.)
(СТЕПАНОВ А.К.)
Исполнители
(БУРЦЕВ В.С.)
(ТОРЧИГИН В.П.)

1973

Дополнительные требования к проекту состояли в том, чтобы комплекс воспринимал все задачи, написанные для БЭСМ-6, на задачах быстрого преобразования Фурье имел производительность порядка 600 млн. операций в секунду, а на задачах уравнений математической физики имел производительность близкую к миллиарду операций в секунду. С этой целью были разработаны спецпроцессоры, воспроизводящие команды БЭСМ-6 (процессор СВС), процессор быстрого преобразования Фурье (БПФ) и векторный процессор МВК "Эльбрус-2". Первые два процессора были сданы Госкомиссии совместно с МВК "Эльбрус-1" в 1980 году. Векторный процессор был запущен в производство в 1986 году и снят с производства ввиду устаревшей элементной базы. Процессор был разработан на элементно-конструкторской базе МВК "Эльбрус-2".

Описание принципов работы векторного процессора МВК "Эльбрус-2" приведено в соответствующей статье [6]. Оригинальные решения этой разработки,

включая "большую команду", используются в настоящее время в зарубежных проектах.

В статье о характеристиках надежности работы многопроцессорных комплексов [7] приведены методы испытания этих вычислительных средств и даны оценки надежности МВК "Элъбрус-2" на момент его испытаний. В этой же статье сформулированы основные условия, при которых многопроцессорные комплексы могут эффективно использоваться как для расчетов, так и в системах реального времени.

Статья "Анализ результатов испытаний МВК "Эльбрус-2" и дальнейшие пути его развития" [8] систематизирует результаты Государственных испытаний и дает рекомендации по дальнейшему развитию этого направления. Рекомендации базируются на основных общепринятых принципах создания и развития сложных вычислительных систем - "step by step".

Если проследить путь создания МВК "Эльбрус", то этот принцип четко прослеживается: в 1968 году Институт точной механики и вычислительной техники (ИТМиВТ) не имел ни интегральных схем, ни многослойных печатных плат для их соединения, ни высокочастотных кабелей и разъемов, ни систем автоматизированного проектирования.

Для построения МВК "Эльбрус" необходимо было решить сразу несколько проблем:

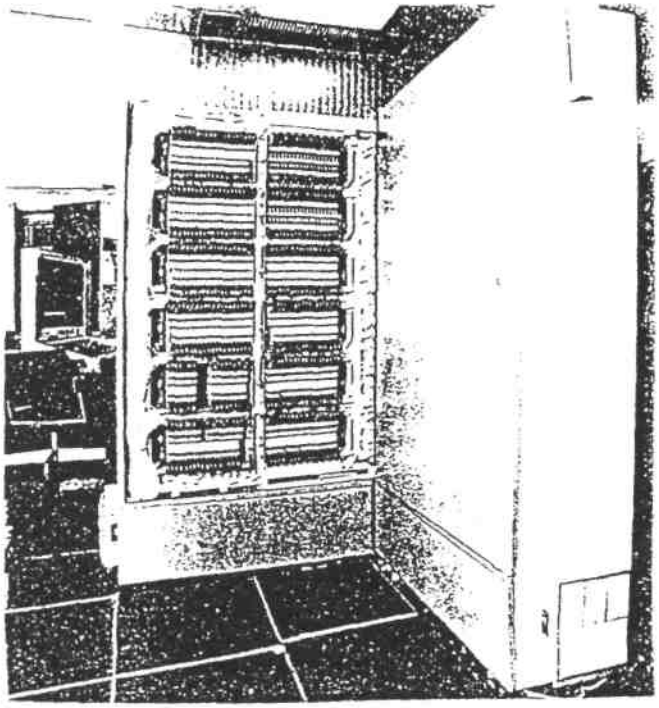
- разработать и наладить производство интегральных схем и печатных плат, создать системы автоматизации проектирования;
- создать высоконадежные конструкции и структуру построения многопроцессорных комплексов;
- решить вопрос пропорционального увеличения производительности с увеличением числа процессоров.

Было решено в качестве первого этапа разработать вычислительный комплекс С-300 и на нем решить часть конструктивно-технологических вопросов, включая разработку системы автоматизации проектирования и вопросы обеспечения надежности многопроцессорных комплексов (модульное резервирование). А на втором этапе, используя отработанные для С-300 конструктивно-технологические решения, создать МВК "Эльбрус-1", на котором решить принципиальные вопросы наращивания производительности комплекса с увеличением числа процессоров. Надо сказать, что в то время никто не верил в возможность пропорционального увеличения производительности за счет увеличения числа процессоров в комплексе. Фирма IBM на своих машинах убедительно демонстрировала падение производительности многопроцессорного комплекса при увеличении числа процессоров - введение четвертого процессора практически не увеличивала производительности всей системы. На третий этап оставалась реализация высокочастотной конструкции, системы охлаждения, использование быстродействующих интегральных схем и многокристальных сборок на них (переход от τ = 10-15 нс к τ = 2 нс) и высокочастотных соединений.

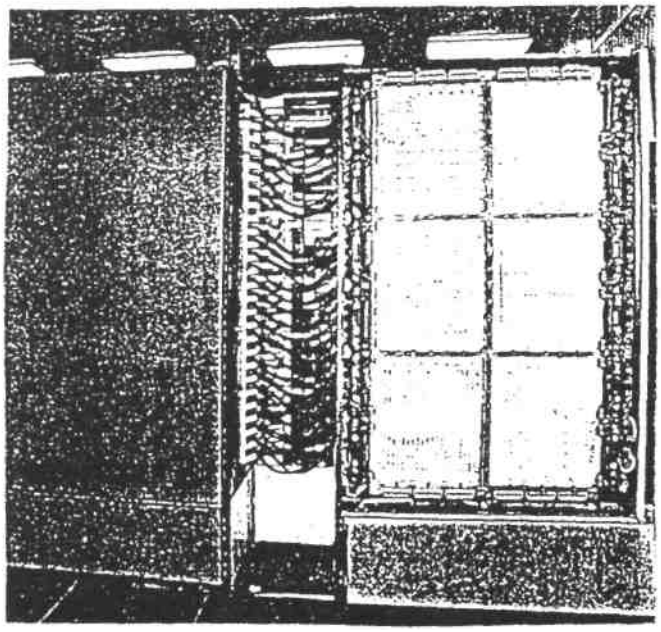
В статье о результатах испытаний даны рекомендации дальнейшего развития МВК "Эльбрус-2", в основе которых лежали те же основные принципы создания этого сложнейшего комплекса - этап модернизации за счет технологического прогресса. Представлялась возможность "свернуть" ячейки в интегральные схемы, что наряду с увеличением производительности процессора в 3-4 раза, существенно повышало технологичность его изготовления и снижало мощность потребления.

К сожалению, эти рекомендации были проигнорированы и в результате МВК "Эльбрус-2" является вот уже в течение более 15 лет последней разра-

боткой суперЭВМ в России, несмотря на то, что средств на модернизацию этого комплекса выделялось достаточно.



Со стороны ячеек



Со стороны плат второго уровня

Рис.1. Шкафы процессора.

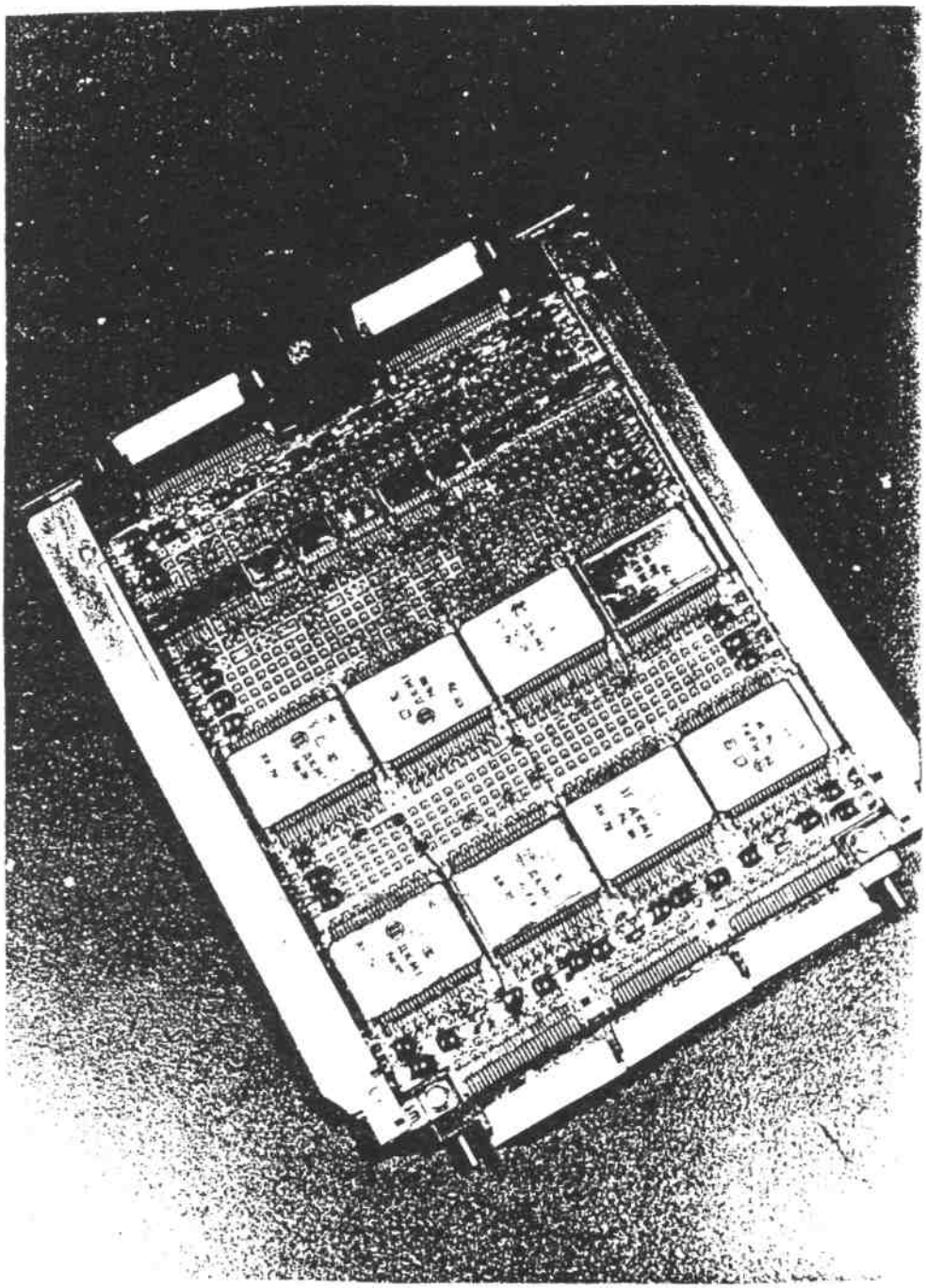


Рис.2. Типовая ячейка МВК «Эльбрус-2», состоящая из сборок мультичипов и микросхем ИС-100.

ДСП *Экз. №11*

МНОГОПРОЦЕССОРНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ КОМПЛЕКС (МВК) "ЭЛЬБРУС"

ЗАДАН Постановлениями ЦК КПСС и СМ СССР от 26 июня 1973 г. № 443-137 ■ от 12 мая 1974 г. № 362-126.

МВК "Эльбрус" относится к классу универсальных высокопроизводительных ЭВМ общего и специального применений и предназначен для использования в высокопроизводительных информапнонно—вычислительных и управляющих системах, а так -же научных и промышленных вычислительных центрах коллективного пользования в режиме разделения времени. МВК "Эльбрус" может использоваться как в универсальных, так и в специализированных системах реального времени.

ПРОИЗВОДСТВО МВК "Эльбрус-1" с 1979 г.; МВК "Эльбрус-2" с 1981 г.

СОСТАВ :

В состав комплектов входят центральные процессоры (до 10) к предусмотрена возможность использования наряду с универсальными специализированных процессоров : СВС - для реализации прикладных программ, написанных для ЭВМ БЭСМ-6, и БПФ - для быстрого преобразования Фурье; оперативная па - мять (до 32.); процессоры ввода-вывода (до 4) ; процессоры приема-передачи данных (до 16) ; синхронизатор центральный; устройства управления накопителями на магнитных барабанах и дисках (до 4); внешние запоминающие устройства; устройства ввода-вывода ,обеспечивающие подключение до 1016 внешних устройств и до 2580 линий связи.

ТИПОВЫЕ КОМПЛЕКТАЦИИ :

однопроцессорная, двухпроцессорная, четырехпропессорная и десятипроцессорная.

КООПЕРАЦИЯ :

Институт точной механики и вычислительной техники ям.С.А.Лебедева-головной.

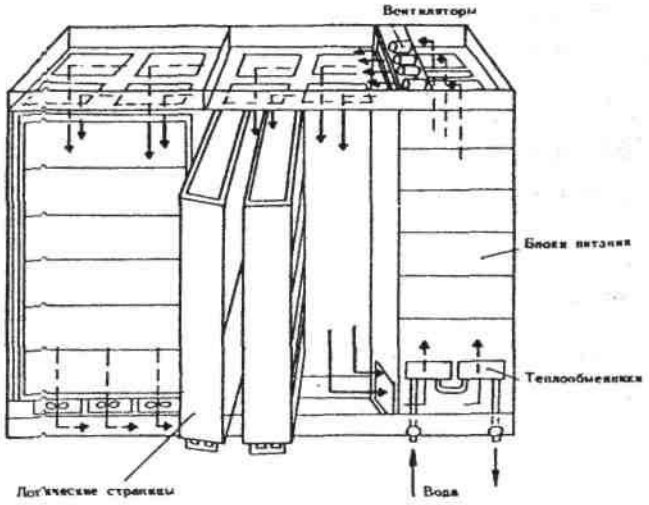
Производственное объединение "'Звезда".

Московский завод счетно—аналитических машин им. ВЛ.Калмыкова.

Пензенский завод вычислительных электронных машин.

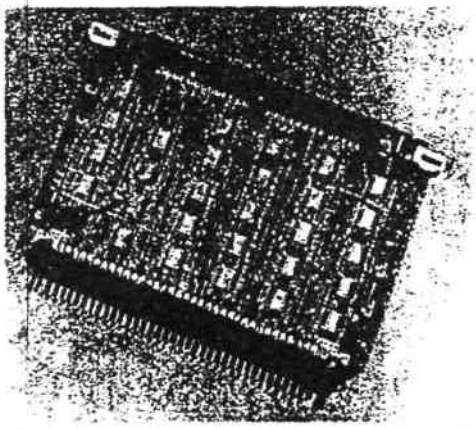
ГЛАВНЫЙ КОНСТРУКТОР В.СБурпев

Типовой шкаф МВК "Эльбрус-1" представляет собой автономную систему, на базе которой построены центральный процессор и процессор ввода-вывода. Она состоит из двух логических стоек и стойки питания-охлаждения.



Типовой шкаф МВК "Эльбрус-1"

Система охлаждения - двухконтурная замкнутая воздушно-жидкостная - отличается эффективностью, экономичностью, надежностью и независимостью типовых режимов от параметров воздуха машинных залов ВЦ.

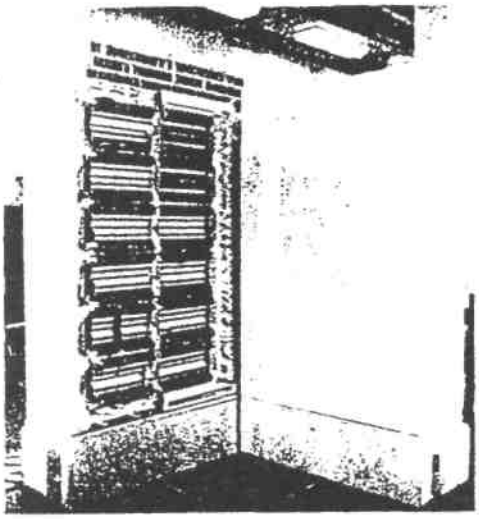


Типовая ячейка центрального процессора МВК "Эльбрус-1" (ИС "Логика-2", 15 нс)

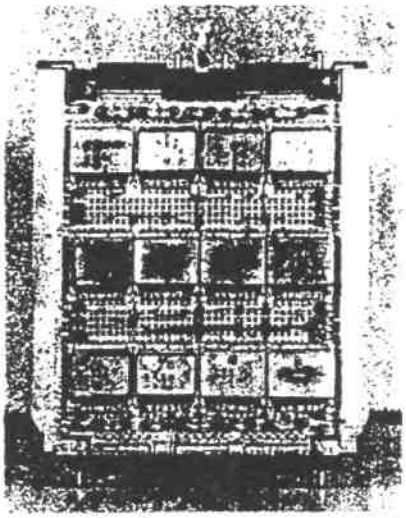
Типовой шкаф имеет размеры 3050 x 700 x 1980 мм.

Производительность МВК "Эльбрус-1" до 15 миллионов операций в секунду.

Емкость оперативной памяти до 1 миллиона 72-разрядных слов.



Центральный процессор МВК "Эльбрус-2"



Типовая ячейка центрального процессора МВК "Эльбрус-2" (МБИС, 2 нс)

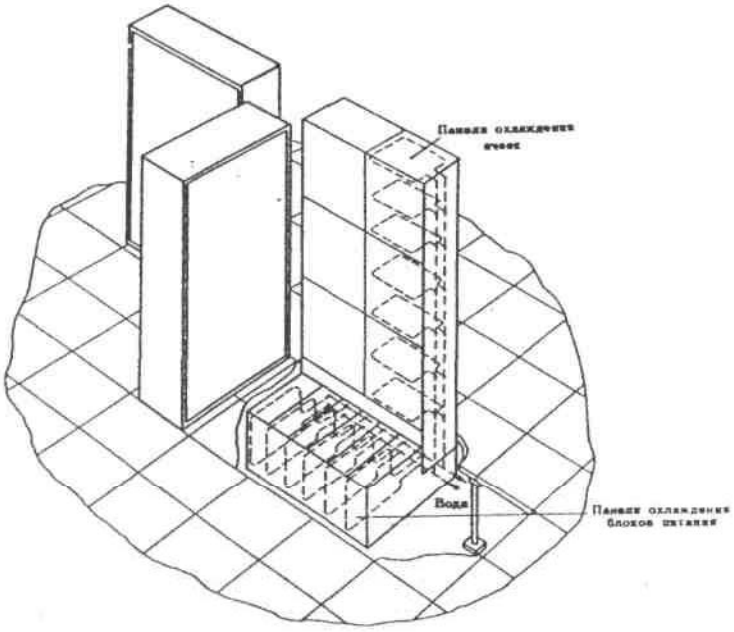
Модульный принцип построения многопроцессорного вычислительного комплекса "Эльбрус" обеспечивает :

- возможность создания различных по характеристикам вычислительных комплексов за счет изменения набора модулей, которые применительно к классу решаемых задач позволяют достичь наибольшей эффективности его работы ;
- высокие показатели надежности и достоверности выдаваемой информации за счет автоматического исключения неисправных модулей из состава комплекса.

Аппаратные средства комплекса "Эльбрус" реализуют :

- эффективное программирование на языках высокого уровня, сокращающее время разработки и отладки программ пользователя ;
- возможность общего системного программного обеспечения, являющегося универсальной общецелевой системой для использования в широких областях применения, включая создание программ, работающих в реальном масштабе времени ;
- обширную библиотеку прикладных программ, написанных для ЭВМ БЭСМ-6.

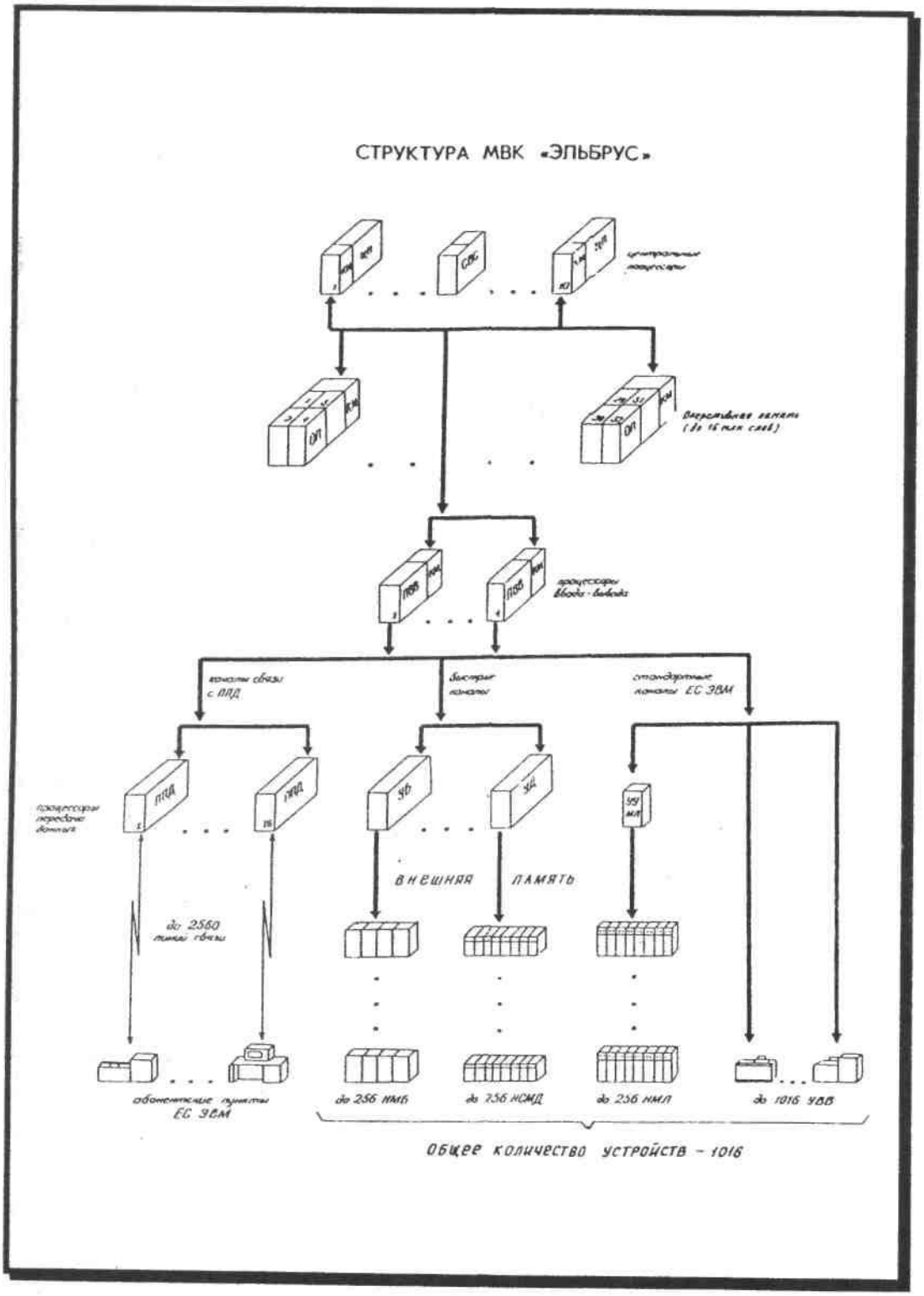
Центральный процессор МВК "Эльбрус-2" представляет собой автономную систему и включает три стойки для размещения функционального оборудования, шкаф для установки источников питания, размещенные под фальшполом, и систему жидкостного кондуктивного охлаждения.



Центральные процессор МВК "Эльбрус-2" с элементами системы жидкостного охлаждения

Жидкостная кондуктивиая система охлаждения, предусматривающая непосред - ственный отвод тепла от элементов к жидкому хладоносителю, циркулирующему по стойкам, позволяет существенно увеличить теплонапряженкость стоек и по сравнению с воздушное системой охлаждения обеспечивает более высокие показатели эффективности, экономичности и надежности.

Стойка центрального процессора имеет габариты 1020 x 350 x 1700 мм. Производительность МВК "Эльбрус-2" до 125 миллионов операций в секунду. Емкость оперативной памяти до 16 миллионов Т2-разрядных слов.


СТРУКТУРА МВК «ЭЛЬБРУС»
до 2560
внешняя
память
до 256 НМБ
до 256 НМД
до 256 НМЛ
до 1016 УВВ
Общее количество устройств – 1016

ТИПОВЫЕ КОМПЛЕКТАЦИИ МВК .ЭЛЬБРУС

Характеристика или наименование устройства	МВК 'Эльбрус-1"					МВК 'Эльбрус-2"			
	Шифр типовой комплектации					Шифр типовой комплектации			
	Э1-1	Э1-2 (Комп-лекта - ция№2)	Э1-2	Э1-4	Э1-10	Э2-1	Э2-2	Э2-4	Э2-10
ПРОИЗВОДИТЕЛЬНОСТЬ. млн.опер./с	1.5	2,5	3,0	6,0	15	12,5	25	50	125
(КОЛИЧЕСТВО ЦЕНТРАЛЬНЫХ	(1)	(1)	(2)	(4)	(10)	(1)	(2)	(4)	(10)
ПРОЦЕССОРОВ»									
ЕМКОСТЬ ОПЕРАТИВНОЙ ПАМЯТИ									
72-РАЗРЯДНЫХ СЛОВ;									
ПЧК, тыс.слов	128	128	256	512	1024	128	256	512	1024
ЭПП. млн.слов	-	-	-	-	-	2	4	8	16
ЕМКОСТЬ ВНЕШНЕЙ ПАМЯТИ:		'							
– НА МАГНИТНЫХ БАРАБАНАХ, млн.байт	8.5	8.5	17	34	136	8,5	17	34	136
– НА СМЕННЫХ МАГНИТНЫХ ДИСКАХ, млн.байт									
ЕС-5056М	34	34	68	136	272	34	68	136	272
ЕС-506I	87	87	175	350	700	87	175	350	700
– НА МАГНИТНОЙ ЛЕНТЕ.млн.байт	70	70	140	280	560	70	140	280	560
СИСТЕМА ТЕЛЕОБРАБОТКИ:									
– КОЛИЧЕСТВО ПРОЦЕССОРОВ	1	1	1*	2	4(1б)**	1	1*	2	4(16)**
ПЕРЕДАЧИ ДАННЫХ									
– КОЛИЧЕСТВО ОБСЛУЖИВАЕМЫХ ЛИНИЙ СВЯЗИ	48	48	48	96	192 (4096)**	48	48	96	192 (4096)**
ЭЛЕМЕНТНАЯ БАЗА (время задержки, мс)	ТТЛ - схемы (15)					ЭКЛ схемы (2)			
ЗАНИМАЕМАЯ ПЛОЩАДЬ, $м^2$:									
– ВЫЧИСЛИТЕЛЬНЫМ КОМПЛЕКСОМ	300	300	360	730	1270	420	640	1260	2290
– ЦЕНТРАЛЬНОЙ ВЫЧИСЛИТЕЛЬНОЙ ЧАСТЬЮ	100	100	140	360	460	100	130	260	430

* Поставляется с дополнительным устройством ГУС

**.Имеется возможность, расширения

Некоторые дилетанты в вычислительной технике, а может и "доброжелатели", считают, что МВК "Эльбрус" скопирован с американской вычислительной машины Burroughs. Я думаю, что эта книга наглядно показывает несостоятельность таких суждений. Мы подходили к идее создания такого комплекса самостоятельным путем **[1,2].** Безусловно, были внимательно изучены передовые проекты того времени, такие как английский проект Манчестерского университета MU-5, американские разработки Maltics, Burroughs и машины серии IBM. В то же время МВК «Эльбрус-1» и «Эльбрус-2» является самостоятельным отечественным оригинальным проектом в области архитектуры, схемотехники, системного матобеспечения и конструирования.

Представляется, что материал книги может быть использован для подготовки специалистов в этой области.

К сожалению нет статьи по конструкторско-технологическим решениям. В качестве иллюстрации на Рис.1 и 2 показаны два основных элемента МВК "Эльбрус-2". На страницах с 8 по 13 приведена так называемая "раскладушка", выпускаемая после завершения работы по теме для представления изделия руководителям высокого уровня. Она также в какой-то мере дает представление о конструкции МВК "Эльбрус-1" и "Эльбрус-2". В этом материале приведены официальные данные основных параметров изделий, подтвержденные Госиспытаниями, завершенными по МВК "Эльбрус-2" в 1985 году.

Автор чрезвычайно благодарен В.П.Торчигину, Ю.В.Никитину, Е.А.Суима, Г.Н.Давыдовой, Л.Н.Табаковой, Т.П.Щашуриной, которые вложили большой творческий труд в подготовку настоящей книги.

В.С.Бурцев

Литература

1. В.С.Бурцев. Перспективы развития вычислительной техники. Доклад на юбилейной научно-технической конференции, посвященной 25-летию ИТМиВТ АН СССР. М., 1973 (Препринт ИТМиВТ АН СССР).
2. Бурцев В.С. Тенденции развития высокопроизводительных систем и многопроцессорные вычислительные комплексы. М, 1977. (Препринт ИТМиВТ).
3. Бурцев В.С. Принципы построения многопроцессорных вычислительных комплексов "Эльбрус". М., 1977. (Препринт ИТМиВТ № 1).
4. В.С.Бурцев, В.П.Торчигин. Система команд. Вычислительный комплекс «Эльбрус» широкого назначения. Техническое предложение. 1973.
5. В.С.Бурцев, В.П.Торчигин. Основные особенности операционной системы МВК "Эльбрус".
6. В.С.Бурцев, Е.А.Кривошеев, В.Д.Асриэли, П.В.Борисов, К.Я.Трегубов. Векторный процессор МВК "Эльбрус-2". М., 1989 (Препринт ОВМ АН СССР).
7. В.С.Бурцев. Характеристики надежности многопроцессорных комплексов и анализ надежности МВК "Эльбрусе". М., 1987 (Препринт ОВМ АН СССР № 169).
8. Бурцев В.С. Анализ результатов испытаний МВК "Эльбрус-2" и дальнейшие пути его развития. М., 1988 (Препринт ОВМ АН СССР № 208).
9. В.С.Бурцев. Значение создания ENIAC в развитии информационно-вычислительных и управляющих систем в России. В кн.: В.С.Бурцев. Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ. М., 1997, с.5-17.

Перспективы развития вычислительной техники

В.С.Бурцев

В настоящее время происходит стремительное развитие вычислительной техники как в нашей стране, так и во всем мире. Недалеко то время, когда вычислительная техника будет внедрена во все области деятельности человека, то есть наступит время, когда основная часть всех общественно необходимых регулярных процессов циркуляции и обработки информации будет происходить через ЭВМ.

За истекший период развития вычислительной техники можно указать следующие основные направления ее использования:

Г. Автоматизированное управление отдельными объектами в большинстве случаев в интересах обеспечения одного или нескольких технологических процессов (управление станками, измерительными установками и так далее).

2. Предварительная обработка информации для ввода в большие вычислительные системы (машины-сателлиты) и небольшие инженерные и коммерческие расчеты.

3. Решение научно-исследовательских задач.

4. Автоматизированное управление в реальном масштабе времени сложными системами, состоящими из большого количества асинхронно работающих объектов, с целью объединения их в единый комплекс для решения общей задачи.

5. Системы управления предприятиями, объединениями, отраслями, работающие в рамках общегосударственной автоматизированной системы сбора и обработки информации для учета и планирования народного хозяйства в целом.

6. Автоматизированные вычислительные комплексы для инженерно-конструкторских и технологических разработок.

7. Большие информационные комплексы для научных исследований, инженерной деятельности и коммунального использования.

Ламповые машины (машины первого поколения) использовались в основном для решения проблемных задач научно-исследовательского плана.

Широкое использование вычислительной техники по первым двум направлениям стало возможным с момента создания полупроводниковых машин, не требую-

щих специального инженерного оборудования для их обслуживания, мобильных и простых в эксплуатации. Эти машины имеют сравнительно большую надежность (относительно тех объектов, которыми они управляют), а их математическое обеспечение, включая рабочие программы, достаточно просто.

Этими обстоятельствами объясняется резкое увеличение сбыта за последние годы в США мини- и малых машин, построенных на интегральных и больших интегральных схемах.

Машины средней производительности не обладают вышеперечисленными преимуществами по сравнению с большими вычислительными машинами. С другой стороны, эксплуатационные показатели их - стоимость эксплуатации на единицу произведенных вычислений - значительно уступают последним. Таким образом, среднему пользователю гораздо выгоднее закупить на определенное время физический ресурс большой вычислительной системы, имея в виду использование ее в режиме пакетной обработки, мультипрограммном режиме или режиме разделения времени. Отсюда объясняется существенное снижение роста спроса на машины среднего быстродействия. Можно предполагать, что в недалеком будущем большие вычислительные машины обеспечат все сферы использования вычислительных средств в направлениях 3-7.

Сравнительно медленное внедрение вычислительной техники по этим направлениям можно объяснить следующим:

1. Вычислительные средства не обеспечивают высокой надежности решения задач, которые сводятся к:

- гарантированному получению достоверных результатов за регламентированный временной интервал, причем для 4-го направления использования этот интервал может колебаться от секунд до микросекунд, для 5-го направления использования может изменяться от часов до секунд;
- гарантированному долговременному хранению больших объемов информации.

2. Создание и эксплуатация рабочих программ современных вычислительных машин требует больших затрат высококвалифицированного труда и не предусматривает широкого распараллеливания работ.

При использовании машин в направлениях 3-7 всегда стоит задача эффективной реализации во времени созданной программы (в особенности при эксплуатации в направлениях 4, 5). Поэтому использование языков высокого уровня, учитывая малую эффективность реализации их на существующих машинах, невозможно. С другой стороны, объем программ большинства задач, обеспечивающих использование в этих направлениях, исчисляется сотнями тысяч команд, в результате чего человек не в состоянии эксплуатировать та-кое математическое обеспечение, имея в виду постоянное совершенствование систем управления.

Эти два требования не столь категоричны для научно-исследовательских центров, так как, с одной стороны, всегда при помощи повторного просчета имеется возможность в значительной степени повысить достоверность результата, а регламентированное время получения последнего может исчисляться месяцами. С другой стороны, в большинстве случаев запрограммированная задача научного плана не требует большого числа повторений, что дает возможность вести программирование на языках высокого уровня. Исключение представляют большие проблемные задачи научного плана.

3. Не решена проблема совместимости машин. Структуры ЭВМ не обеспечивали ингтегрирования математического обеспечения из поколения в поколение даже на уровне программ пользователя. Более того, каждая новая конструкция ЭВМ требовала создания самостоятельного математического обеспечения, разрабатываемого практически от "нуля".

4. Малая емкость оперативной памяти и недостаточная скорость работы внешней памяти не позволяли обеспечить требуемых скоростей решения многих задач или делали постановку их вовсе невозможной.

Эти же причины в значительной степени затрудняли эффективную реализацию мультипрограммного режима и режима разделения времени.

5. Не решена Проблема комплексирования территориально разнесенных вычислительных средств посредством линий передачи.

Естественно, Не решив этих пяти основных проблем раннего этапа развития вычислительных средств, нельзя рассчитывать на серьезное внедрение вычислительной техники по основным направлениям ее использования.

Рассмотрим пути решения этих проблем.

1. Обеспечение высокой надежности решения поставленных задач

Наиболее естественный путь повышения надежности состоит в увеличении надежности составляющих элементов и в этом направлении, безусловно, необходимо вести большие работы, которые смогут за обозримый период увеличить надежность составных элементов на 1-3 порядка. Однако такое положение, как показывает опыт развития вычислительной техники, не решит поставленной задачи для больших ЭВМ. Для увеличения логической оснащенности вычислительных средств, а также значительного увеличения емкости оперативной памяти, внешних запоминающих устройств и устройств ввода-вывода, наверняка потребуется использование всех возможностей, предоставляемых элементной базой. Таким образом, будет правильным считать,, что увеличение надежности элементов решит проблему надежности только в части класса малых машин, где требования по надежности не так велики, а количество используемых элементов меньше на несколько порядков по сравнению с большими ЭВМ.

Кардинально Проблема надежности в настоящее время может быть решена только структурным способом.

Одним из таких способов является модульный принцип построения вычислительного комплекса, где каждый модуль охвачен полным аппаратным контролем, обеспечивающим высокую достоверность получаемых результатов. Такая вычислительная система, состоящая из одинаковых модулей процессоров, оперативной, внешней памяти и устройств ввода-вывода, принципиально может обеспечить Любую, наперед заданную вероятность решения задачи за регламентированный интервал времени, причем вероятность решения задачи определяется глубиной резервирования. Отключение дефектного модуля в такой системе производится автоматически по срабатыванию аппаратного контроля модуля. Несколько отличным способом обеспечивается надежность сохранности больших объемов информации. В зависимости от требований оперативности ее использования и объема хранения она может дублироваться на нескольких уровнях иерархии памяти (оперативное ЗУ, промежуточная память, внешняя память, память ввода-вывода).

Однако модульный принцип построения диктуется не только соображениями надежности.

Так, требование высокого быстродействия (200-500 нс) работы ОЗУ инженерно и технически невозможно совместить в одном блоке с требованием к емкости памяти 10^8 бит. Подобная память из-за конструктивно-технологических соображений будет состоять не менее чем из нескольких десятков блоков (модулей) памяти. Если на такую память работает только один процес-

сор, то коэффициент использования самого дорогостоящего оборудования -блоков ОЗУ - не будет превосходить 5-10%.

Таким образом, с целью оптимального использования рационально включать в систему несколько процессоров, работающих на одну общую память. Кроме того, многопроцессорный модульный принцип построения вычислительных средств позволяет строить комплексы различной производительности с оптимальным соотношением мощности процессоров и емкости оперативной и внешней памяти.

2. Трудности создания и эксплуатации рабочих программ

В настоящее время не вызывает сомнения тот факт, что большие программы представляют собой наиболее сложные в информационно-логическом отношении творения из всего, что когда-либо создавал человеческий ум. Трудности программирования уже достигли таких масштабов, что если не принять соответствующих мер при разработке структуры создаваемых машин, то будет существовать очень много полезных и актуальных задач, которые практически невозможно будет запрограммировать ввиду их алгоритмической сложности.

Решить эту проблему на настоящем этапе развития представляется возможным за счет существенного пересмотра традиционных принципов построения внутренней структуры управления и системы команд ЭВМ с целью достижения наибольшей эффективности выполнения программ, написанных на языке высокого уровня.

Основное отличие машинного языка от языка высокого уровня состоит в том, что программа, написанная на языке машины, помимо конкретного воплощения алгоритма содержит определенную последовательность данных, управляющих ресурсами вычислительной системы. Эта последовательность данных является ненужными накладными расходами на воплощение алгоритма в машинный язык по сравнению с изложением этого алгоритма на языке высокого уровня.

Во время программирования человек перерабатывает содержание алгоритма в термины машинного языка. Поэтому процесс программирования значительно упростится, если:

- машинный язык не требует введения в программу никакой другой информации, кроме информации о самом алгоритме;
- информация об алгоритме излагается наиболее компактным образом.

Можно сказать, что уровень программирования тем выше, чем ближе он к изложению алгоритма наиболее компактным способом без использования информации, управляющей ресурсами. И, наоборот, язык программы можно назвать в большей степени машинно-ориентированным, если программируя на нем, приходится вводить в программу информацию о физических ресурсах, необходимых для ее выполнения.

В языках существующих машин имеются две категории информации, управляющей распределением физических ресурсов. Первая из них выбирает тип памяти (регистры, оперативная или внешняя память) и физические адреса, по которым находятся операнды, вторая определяет на какой аппаратуре необходимо произвести обработку данных (АУ с фиксированной или плавающей запятой, полноразрядное или сокращенное и так далее).

Проследив развитие машинных языков и, соответственно, структур процессов, можно обнаружить уход от физических адресов, прежде всего при обращении к внешней памяти (что уже практически осуществляется), затем при обращении к оперативной и в конце концов при обращении к регистровой памяти.

Так, уже с введением ранних операционных систем или даже просто минимального объема обслуживающих программ из программ машин были исключены физические адреса внешних устройств. Это объясняется тем, что для этих устройств ввиду их малого быстродействия можно было обойтись интерпретационным программным способом, не затрагивая оборудования.

Труднее обстоит дело с адресацией величин, расположенных в основной памяти. Первые попытки аппаратными средствами добиться исключения физических адресов из машинных языков привели к базированию и введению математической памяти.

Надо сказать, что базирование не исключает информацию о ресурсах из программ, так как вся область, внутри которой описывается алгоритм, представляет ограниченное линейное адресное пространство, точно отображающее эквивалентную область физической памяти.

Таким образом, базирование не решает проблемы исключения физического ресурса из программы.

Введение математической памяти при правильной ее организации может способствовать решению этого вопроса. Однако было бы неверно адресовать данные в программах с ее помощью. Во-первых, это противоречит принципу лаконичности программ (математический адрес имеет большую избыточность информации для локального программного модуля). Во-вторых, в этом случае сама математическая память становится ресурсом.

Можно предполагать, что наиболее вероятным решением отображения математической памяти в физическую будет разбиение ее на страницы произвольной длины - сегменты, информация внутри которых имеет общий статус защиты и единую "судьбу" жизни на различных уровнях иерархии памяти. Так, например, внутри одной задачи могут быть сформированы программные сегменты, сегменты промежуточных вычислений и сегменты данных. Такая сегментация математической памяти значительно упростит работу операционной системы и будет свободна от недостатков принципа жестких страниц.

В настоящее время идет процесс исключения из машинных языков информации о прямоадресуемых регистрах. Найдены аппаратные решения динамического рапределения быстрых регистров, которые значительно повысят коэффициент использования этого дорогостоящего оборудования и тем самым увеличат производительность процессоров.

Вторая категория информации о ресурсах связана с обработкой данных. Во время разработки алгоритма человек использует элементарные операции (сравнение, сложение и тому подобное), реально выполняющиеся на физическом устройстве (на устройстве целочисленного сложения, устройстве сложения чисел с плавающей запятой и так далее). Алгоритмические операции определены для абстрактных объектов (числа, множества булевых переменных и других); реальные устройства работают над их кодовым представлением. Так, например, сложение двух величин может программироваться одинаково, вне зависимости от того, что складывается - целые, вещественные числа, любая их комбинация или даже выражения, описывающие алгоритм их вычисления. В реальной же программе на старых машинах указывается не абстрактная операция, а реальное физическое устройство (фиксированное сложение, плавающее сложение и так далее).

В ранее разработанных машинах не было другого выхода, так как в памяти содержался лишь код величины, не несущей информации о ее типе. В случае введения абстрактных операций необходимо вместе с каждой величиной иметь ее тип. Решение об использовании конкретного устройства принимается динамически во время выполнения программы в зависимости от типа посту-

пающей величины. В частности, если при реализации операции сложения в качестве операндов приходят имена, описывающие два массива данных, происходит аппаратное сложение этих массивов. Если же один из операндов есть имя процедуры, то происходит аппаратный запуск процедуры, а результат работы этой процедуры используется как операнд.

Как уже указывалось, вторым способом упрощения программ является более компактное изложение "чистого" алгоритма, другими словами, более оптимальное кодирование адресной информации и операторов.

Для исключения этой информации из программ целесообразно в язык машины ввести наиболее устоявшиеся в языках высокого уровня формы описания алгоритмов, такие как выражение и процедура. Аппаратная реализация такого машинного языка почти однозначно приведет к безадресному принципу указания операндов и адресации при помощи имен как внутри одной задачи, так и при обращениях к архивам.

Процесс декомпозиции программы с помощью системы процедур приводит к тому, что программа приобретает многоуровневую структурность, сильно облегчающую ее создание и восприятие. В результате введения процедуры довольно сложное понятие, предоставляемое ею, можно легко использовать, так как в месте использования нужно знать лишь интерфейс для ее запуска.

Последовательное проведение этих принципов в аппаратуре приводит к тому, что различные узлы оборудования рассматриваются как аппаратно реализованные процедуры. Каждая команда в программе - это процедура без параметров с очевидным значением. Считывание - это одноместная процедура. Арифметические операции - двухместные процедуры. Команды записи - это процедуры, изменяющие динамический контекст.

Уход на прерывание, например по переполнению, - это запуск глобальной программной процедуры из процедуры сложения, реализованной аппаратно.

Необходимо отметить, что на первом этапе такого подхода к проектированию ЭВМ высказывалось предположение, что увеличение логических возможностей устройств снизит их быстродействие. Однако опыт разработки показал, что новые принципы построения вычислительных систем, основанные на непосредственном восприятии аппаратурой устоявшихся методов организации вычислительных процессов, заложенных в языках высокого уровня и операционных системах, позволяют существенно повысить производительность вычислительных средств. Значительное сокращение времени на вычисления получается за счет исключения таких чрезвычайно часто встречающихся в вычислительных процессах подпрограмм, как подпрограммы преобразования типов и форматов данных, подпрограммы операционных систем, связанных с распределением оперативной памяти, существенного сокращения времени перехода с одной процедуры на другую и подстановки параметров, за счет более эффективного динамического распределения быстрых регистров аппаратными методами.

Несомненно, что аппаратная реализация этих принципов вызовет пересмотр существующих языков высокого уровня, что приведет к более сжатым и обозримым формам изложения алгоритмов, свободных от побочной информации.

Каковы же технические предпосылки создания таких высокоорганизованных процессоров?

Можно ожидать, что количество логических элементов, необходимое для их реализации, возрастет более чем в 10-20 раз. Несмотря на это, в процентном отношении стоимость процессора уменьшится по сравнению с возросшей стоимостью памяти (Рис.1).

Несколько слов об элементной базе логических устройств и возможных пределах ее быстродействия.

Рис.1.

Необходимая плотность компоновки элементов полностью определяется их внутренней задержкой τ_3 при условии, что допустимая длина связи не должна сильно ухудшать быстродействие элемента. Построенная экспериментальная кривая плотности компоновки элементов в плоскости $K_э$ в зависимости от их быстродействия (Рис.2) близка к квадратичной функции $K_э \equiv (1/\tau_3)^2$. С другой стороны, уровень разработки элемента определяется той работой $A_э$, которую он совершает за одно срабатывание, то есть произведением мощности, рассеиваемой одним вентилем $P_э$, на время задержки элемента τ_3 [$A_э = P_э\tau_3$].

Таким образом, меняя конструкцию полупроводникового прибора и оставаясь на том же технологическом уровне его изготовления, можно варьировать параметры $P_э$ и τ_3 . Уменьшая τ_3, необходимо во столько же раз увеличить мощность, рассеиваемую одним элементом. Учитывая, что плотность компоновки на плоскости обратно пропорциональна квадрату задержки элемента, получим зависимость мощности, выделяемой на один квадратный сантиметр, P от τ_3, которая будет обратно пропорциональна кубу τ_3:

$$P = K_э P_э \equiv (1/\tau_3)^2 A_э/\tau_3 \equiv (1/\tau_3)^3.$$

В свою очередь, величина P определяет возможный принцип охлаждения: воздушный - до 10^{-1} Вт /см^2, жидкостный - до 10 Вт /см^2 или испарительный - свыше этого предела.

Из. характеристик видно, что разработанные в МЭП интегральные схемы серии 100 даже при воздушном охлаждении позволяют коструировать ЭВМ с временем задержки на элемент в 2-4 нс. На этой же технологической базе при жидкостном охлаждении может быть достигнуто предельное быстродействие элементов, соответствующее $\tau_3 \approx 0{,}8$ нс.

В настоящее время ведутся научно-исследовательские работы как в схемотехническом, так и в технологическом направлениях, которые позволят уменьшить $A_э$ в 10-100 раз, что сократит τ_3 при жидкостном охлаждении до 0,1 нс.

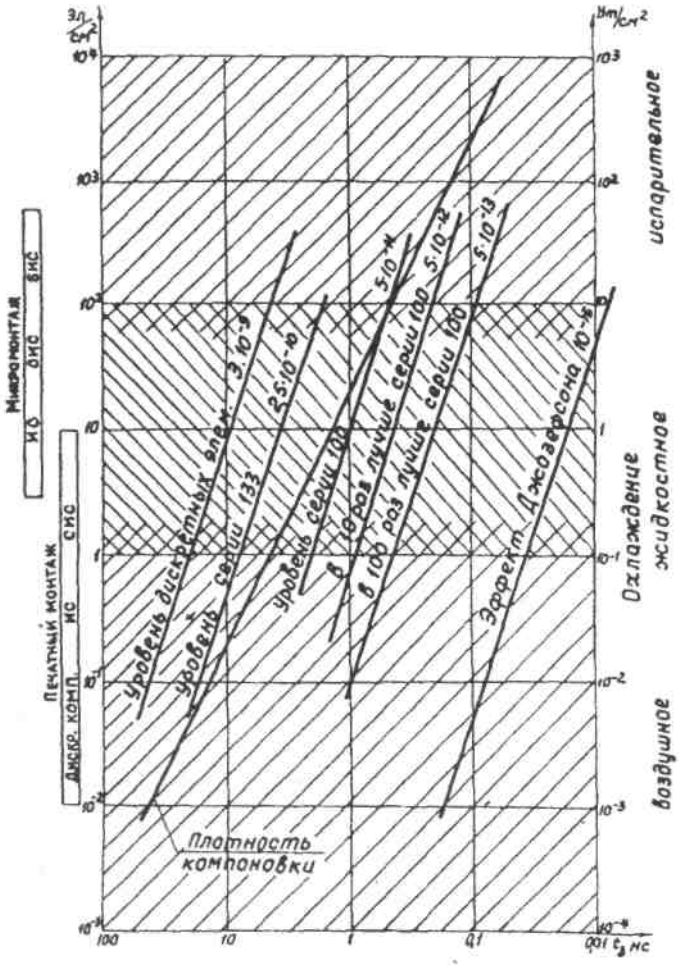


Рис.2.

Необходимо заметить, что значительное увеличение температурных градиентов приведет к существенному увеличению скорости деградационных процессов, то есть к резкому снижению долговечности и надежности элементов.

Решение этой проблемы может состоять либо в работе элементов при криогенной температуре, например 80°К, либо в открытии новых принципов реализации логических функций с существенно меньшими затратами работы на одну логическую операцию.

3. Совместимость

Эффективная реализация языков высокого уровня на ЭВМ в значительной степени решает проблему совместимости, однако не до конца.

Рабочие программы, написанные на языках высокого уровня, безусловно, могут быть перетранслированы на язык вновь создаваемых машин. Однако рабочая программа, написанная на языке высокого уровня, работает в системе, состоящей из самой машины, операционной системы и архива. Поэтому реализация перетранслированной программы на новой ЭВМ требует полной ин-

терпретации реакций старой операционной системы на новых средствах. Не лучше дело обстоит с архивом, так как его структура может не совпадать со структурой нового архива данных. Надо ожидать, что в недалеком будущем будут установлены базовые принципы построения операционной системы и архивов. После этого необходимо хотя бы в пределах страны, стандартизовать работу операционной системы и структуру архива. Последнее, кроме решения вопроса совместимости, откроет широкие возможности реализации этих принципов в структуре ЭВМ. До этого момента в аппаратуру могут внедряться только наиболее устоявшиеся алгоритмы работы операционной системы.

4. Малая емкость оперативной памяти и недостаточная скорость работы внешней памяти

Создание больших вычислительных систем, их структура и основные характеристики всецело зависят от емкости и скорости работы оперативной памяти. Требования разрабатываемых в настоящее время многопроцессорных систем следующие: емкость 10^8 бит и время цикла 200 нс. В недалеком будущем эти требования возрастут до 10^9 бит при времени цикла 80-50 нс. Создание таких ОЗУ на базе полупроводниковой технологии находится в настоящее время на уровне научно-исследовательских и опытно-конструкторских работ.

Необходимо отметить, что ОЗУ, построенные с использованием современной полупроводниковой технологии и схемотехники, будут обладать следующими существенными недостатками:

- большая потребляемая мощность - 50кВт для ОЗУ емкостью 10^8 бит, что значительно усложняет эксплуатацию таких устройств и снижает экономические показатели всего комплекса;
- стоимость такого ОЗУ превышает 2 млн.руб.;
- сравнительно невысокая надежность (по сравнению с процессором).

Ввиду этого создание ОЗУ, свободного от указанных недостатков, для больших вычислительных систем является первоочередной проблемой развития вычислительных средств. Эта проблема должна решаться путем совершенствования технологии и схемотехники полупроводниковых ИС, а также путем поиска новых принципов создания оперативной памяти. Наиболее конкурентноспособными с полупроводниковыми ЗУ надо считать оперативные оптоэлектронные ЗУ.

Основными преимуществами оптоэлектронных ЗУ являются:

- высокая плотность записи информации (10^5-10^7 бит/см^2);
- относительная технологическая простота изготовления блоков хранения информации и адресной коммутации элементов памяти.

Для реализации оперативного оптоэлектронного ЗУ в настоящее время отсутствуют какие-либо подходящие носители информации. Исследования по поиску таких носителей проводятся во многих направлениях: специальные магнитные пленки, аморфные полупроводники, термопластические слои и другие.

Таким образом, если проблема реализации логических устройств на настоящий момент решена и находится на уровне инженерно-технологического воплощения, то проблема создания оперативной памяти для больших вычислительных систем стоит чрезвычайно остро, и здесь должны быть развернуты поисковые научно-исследовательские работы.

Не лучше дело обстоит и с физической реализацией так называемой внешней памяти.

Во многом быстродействие вычислительных систем (в особенности при работе их в режимах мультипрограммной работы и режиме разделения времени)

зависит от оперативности "подкачки" данных в ОЗУ из внешней памяти. Быстродействие многих созданных систем для большого круга задач целиком определяется небольшой скоростью работы внешней памяти.

С целью исправления такого положения во многих существующих больших ЭВМ в качестве промежуточной памяти используются диски с головкой на тракт.

Для согласования скорости работы ОЗУ и внешних ЗУ в структуру многих комплексов будет введено промежуточное ЗУ большой емкости (Рис.3). Минимальные требования к объему такого ЗУ сводятся к 10^9 бит при времени цикла 1-5 мкс; в дальнейшем эти требования изменяются пропорционально изменениям параметров ОЗУ.

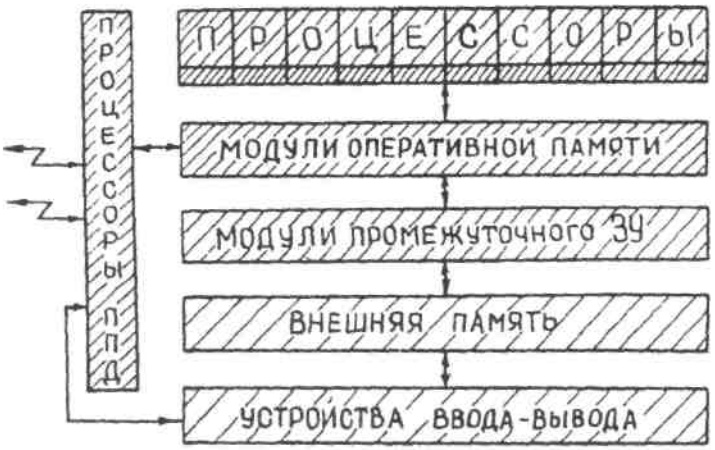


Рис.3.

В настоящее время такая память может быть реализована на ферритах с временем цикла 2-3 мкс. В недалеком будущем возможно построение аналогичной памяти на МОП-транзисторах.

Ведутся научно-исследовательские работы по созданию большой памяти на принципах оптоэлектроники, элементах с зарядовой связью и цилиндрических магнитных доменах (ЦМД). ЦМД позволяют на одной пластинке размером в 10^2 $см^2$ разместить, 10^7 бит информации при темпе ее передачи 10^7 бит/с.

Возможно, что для реализации такой памяти будут использованы биологические принципы хранения информации.

5. Проблема объединения территориально разнесенных вычислительных средств посредством линий передачи данных

В течение ближайших десятилетий во всех областях народного хозяйства будут внедряться и широко использоваться сети ЭВМ, состоящие из множества узлов обработки информации, объединенных линиями передачи данных.

Основными элементами сети должны явиться местные узлы, включающие в себя мини-машины и абонентские пункты, и центральные узлы, построенные на основе высокопроизводительных вычислительных систем.

Перспективное решение проблемы адаптации ЭВМ к таким вычислительным сетям состоит во включении в общий состав вычислительных комплексов специализированного процессора передачи данных, выполняющего основную часть функций взаимодействия с различными объектами через линии связи.

Такой процессор, эффективно воспринимающий язык описания вычислительных сетей, работая со своей автономной памятью, позволит без какого-либо изменения аппаратуры и программ основного вычислительного комплекса адаптироваться к различным конфигурациям вычислительной системы.

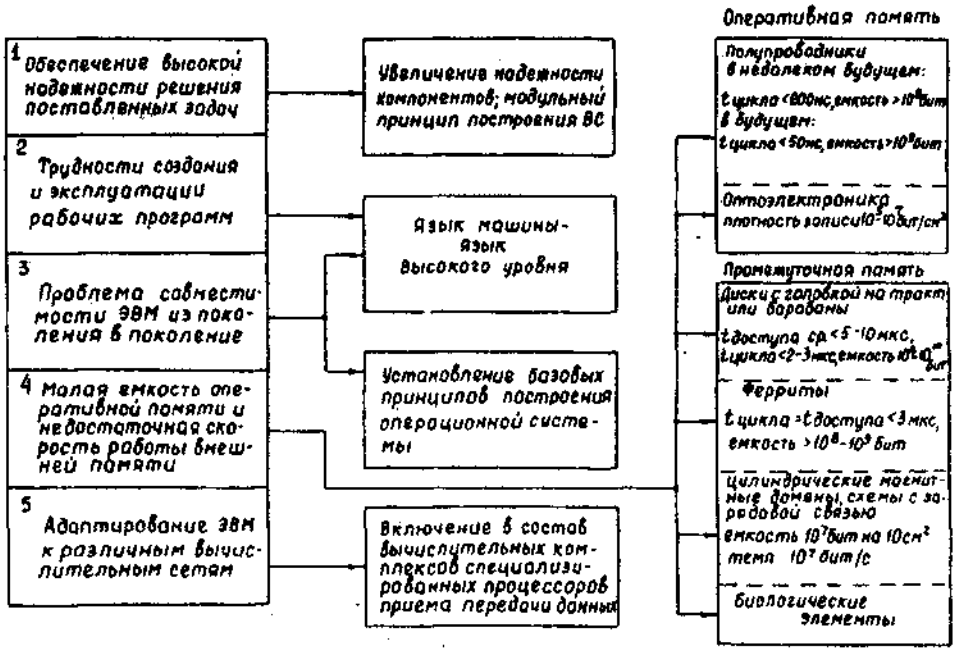


Рис.4.

Подводя итог, можно указать на следующие основные тенденции развития вычислительных систем и их структур (Рис.4): 1.' Модульный принцип построения.

2. Система команд процессора - язык высокого уровня.

3. Внесение в аппаратуру устоявшихся методов работы операционной системы и принципов построения архивов.

4. Значительное расширение емкости оперативной памяти до 1 млн.слов и введение в структуру промежуточной быстрой памяти емкостью не менее 10 млн. слов.

5. Включение в структуру срециализированных процессоров приема-передачи данных.

Тенденции развития высокопроизводительных систем и многопроцессорные вычислительные комплексы

В.С.Бурцев

Опыт использования вычислительных средств значительно расширил сферы применения сверхбыстродействующих вычислительных машин и комплексов. Решение таких задач, как автоматизация проектирования сложных объектов и систем в различных отраслях народного хозяйства; создание крупных информационных комплексов, работающих в режиме коллективного пользования; построение больших управляющих систем, состоящих из множества объектов, работающих в системе реального времени; задачи обработки больших объемов информации, получаемых со спутников; задачи моделирования физических процессов в интересах решения различных научных проблем и ряд других задач, предъявляют к вычислительным средствам практически неограниченные требования по быстродействию и емкости памяти. В то же время быстродействие создаваемых в настоящее время однопроцессорных комплексов близко к физическому пределу, эффективность использования самого дорогого оборудования памяти чрезвычайно низка, а живучесть и надежность таких систем, как правило, не отвечают необходимым требованиям.

В настоящей работе делается попытка показать, что перспектива развития быстродействующих вычислительных комплексов однозначно связана с созданием многопроцессорных систем, которые наряду с увеличением быстродействия комплекса обеспечат более эффективное использование памяти и значительно повысят структурную надежность комплекса. Несомненно, что многопроцессорная модульная структура ввиду целого ряда своих положительных качеств найдет широкое применение как в комплексах высокой производительности, так и при построении комплексов средней и малой производительности.

Оценивая физические пределы быстродействия однопроцессорной системы, мы имеем в виду использование полупроводниковых элементов, поскольку промышленный выпуск элементов, построенных на новых физических принципах, например с использованием криогенной техники, оптики, бионики и так далее, можно ожидать не ранее чем через 15-20 лет.

Физические пределы быстродействия вычислительной машины определяются быстродействием процессора и оперативной памяти. В то же время быстро-

Действие и процессора, и памяти характеризуется двумя показателями: *временем выполнения* операции и максимальным *темпом их выполнения.*

Под *временем выполнения* отдельной операции мы будем понимать время, которое затрачивается от момента выдачи задания на выполнение операции из заданной точки пространства до момента прихода результата в заданную точку пространства. Так, для оперативной памяти это время $T_{ОП\ ОЗУ}$ (для операции считывания) будет определяться разностью времен от момента подачи адреса на вход оперативной памяти до момента прихода выбранного числа на выходные шины.

Это же время может быть определено относительно входных и выходных клемм процессора

$$T_{оп\ пр\text{-}озу} = 2\ T_{св\ пр\text{-}озу} + T_{оп\ озу}, \quad (1)$$

где: Тсв пр-озу- время задержки в линии связи между процессором и ОЗУ.

В то же время максимальный *темп выполнения* операций для ОЗУ определится минимальным возможным временным интервалом выдачи адресов для считывания или записи, при котором операция выполняется без искажения информации. Аналогичные определения могут быть сформулированы и в отношении основных характеристик быстродействия процессора - *темп выполнения* операций и *время выполнения* операции.

Важным является то обстоятельство, что практически неограниченное увеличение *темпа выполнения* операций может быть решено довольно простым структурным способом, в то время как уменьшение *времени выполнения* операции имеет принципиальные физические пределы.

Наиболее распространенными структурными способами увеличения *темпа выполнения* операций являются магистральный принцип (принцип "трубопровода"), интерливинг и другие, заключающиеся в запараллеливании работы многих устройств в процессе обработки информации. Наиболее эффективно эти способы могут быть использованы при построении *специализированных* вычислительных средств, ведущих обработку по *фиксированной* программе. В пределе период темпа ограничен только временем переключения элемента ($\tau_э$).

Довольно часто, оценивая максимальную производительность процессора или памяти, приводят данные, характеризующие *темп выполнения* операций, опуская такой немаловажный для скорости реализации вычислительного процесса параметр, как *время выполнения* операции.

Для универсальных вычислительных машин, где последовательность операций не определена заранее, сокращение *времени выполнения* операции является существенным фактором, определяющим производительность комплекса.

Чем же определяется *время выполнения* операции и каковы его физические пределы для процессора и памяти?

При выполнении одной элементарной операции информация проходит через n последовательно соединенных логических элементов, а в памяти, кроме того, через сам запоминающий узел. Поэтому *время выполнения* операции для процессора $T_{оп\ пр}$ равно сумме задержек каждого элемента $\tau_{эi}$ и задержек в цепях связи между ними $T_{СВI}$;. Переходя к средним значениям $\tau_э$ и $\tau_{СВ}$, получим

$$T_{ОП\ ПР} = n(\tau_э + \tau_{СВ}), \quad (2)$$

где: $\tau_э$ - усредненное значение задержки логического элемента;

$\tau_{СВ}$ - среднее значение задержки в цепочке связи, приходящееся на один логический элемент.

Аналогично этому Топ озу определяется как

$$\text{Топ озу} = к(\tau_э + \tau_{СВ}) + \tau_{СВ\ ЗП} + \tau_{ЗП} \quad (3)$$

где: к - число последовательно соединенных логических элементов;

$\tau_{СВ\ ЗП}$ - задержка сигнала в цепях коммутации запоминающих элементов;

$\tau_{ЗП}$ - время срабатывания запоминающего элемента.

Время выполнения операции в системе процессор - оперативное запоминающее устройство Топ может быть оценено выражением

$$\text{Топ} = \text{Топ}_{ПР} + 2R\ \text{Топ}_{ПР\text{-}ОЗУ}, \quad (4)$$

где: R - учитывает, что только R-ая часть обращений за данными идет в оперативную память (остальные данные находятся в сверхоперативной памяти).

Принято, что на одну операцию процессора в среднем потребуется два обращения к памяти за кодами команд и операндами.

Оценивая минимальную величину Топ, необходимо рассмотреть принципиальные возможности уменьшения ее составных величин $\tau_э$, $\tau_{св}$, $Тсв_{ПР\text{-}ОЗУ}$, $\tau_{СВ\ ЗП}$, $\tau_{ЗП}$.

В машинах 1-3 поколений времена задержек в линиях связи практически не оказывали влияния на быстродействие системы процессор-ОЗУ, так как при существующем уровне технологии коммутации элементов и временах $\tau_э > 10$ нс, $\tau_{ЗП} > 100$ нс достаточно легко выдерживалось соотношение $\tau_{СВ} \ll \tau_э$ и $\tau_{СВ\ ЗП} \ll \tau_{ЗП}$.

При переходе к элементам $\tau_э < 1$ нс вопрос сокращения длин связи стал определяющим на пути дальнейшего повышения быстродействия устройств.

Задавая величину у, определяющую процент потери быстродействия элементов в системе за счет линий связи, можно получить соотношение между $\tau_э$ и плотностью компоновки элементов на плоскости (К)

$$\tau_{СВ} = \gamma\ \tau_э = \alpha\ t_0 \sqrt{\frac{1}{K}} \quad (5)$$

где: t_0 - задержка в линии связи на единицу длины;

α - коэффициент, учитывающий возможности трассировки связей в принятой конструкции и эффективность алгоритмов трассировки.

В существующих конструкциях а имеет величину 10-30, а средняя величина ε равна 5. В этом случае $t_0 = 0{,}075$ нс/см.

Из формулы (5) видно, что для уменьшения $\tau_э$ требуется существенное увеличение плотности компоновки (квадратичная зависимость). С другой стороны, увеличение плотности компоновки связано с преодолением целого ряда технологических трудностей и решением важной проблемы отвода тепла.

Необходимо отметить, что при одном и том же уровне технологии и одинаковых схемотехнических решениях уменьшение времени задержки влечет за собой пропорциональное увеличение величины мощности, расходуемой на один логический элемент. Поэтому основным параметром элемента, определяющим уровень его разработки, необходимо считать величину $А_э = Р$, $\tau_э$ -энергию, затрачиваемую элементом на одно логическое срабатывание ($Р_э$ - мощность, рассеиваемая одним логическим элементом).

Таким образом, мощность, выделяемая на единицу площади (рассматривается наиболее типовая плоская конструкция) в узлах ЭВМ, равна

$$Р = Р_э К = А_э К/ \tau_э.$$

Подставляя из формулы (5) значение К, выраженное через $\tau_э$, получаем:

$$Р = (\alpha t_0/ \gamma)^2 А_э\ 1/\tau_э^3 \quad (6)$$

Предельно допустимое значение P определяется предельными возможностями съема тепла.

Уменьшение $A_Э$ для полупроводниковой техники ограничено пределами уменьшения размеров компонентов, предельно допустимыми напряженностями полей в структурах и плотностями токов в линиях связи внутри приборов.

Подставляя достаточно оптимистические значения параметров с учетом схемотехнических возможностей снижения мощности рассеивания одним логическим элементом в формулу (6) ($P_э$ = 20 Вт/см^2, $A_Э$= 1 пДж, y = 0,1, α = 10), получим минимальное значение $\tau_Э \approx 0{,}2$ нс.

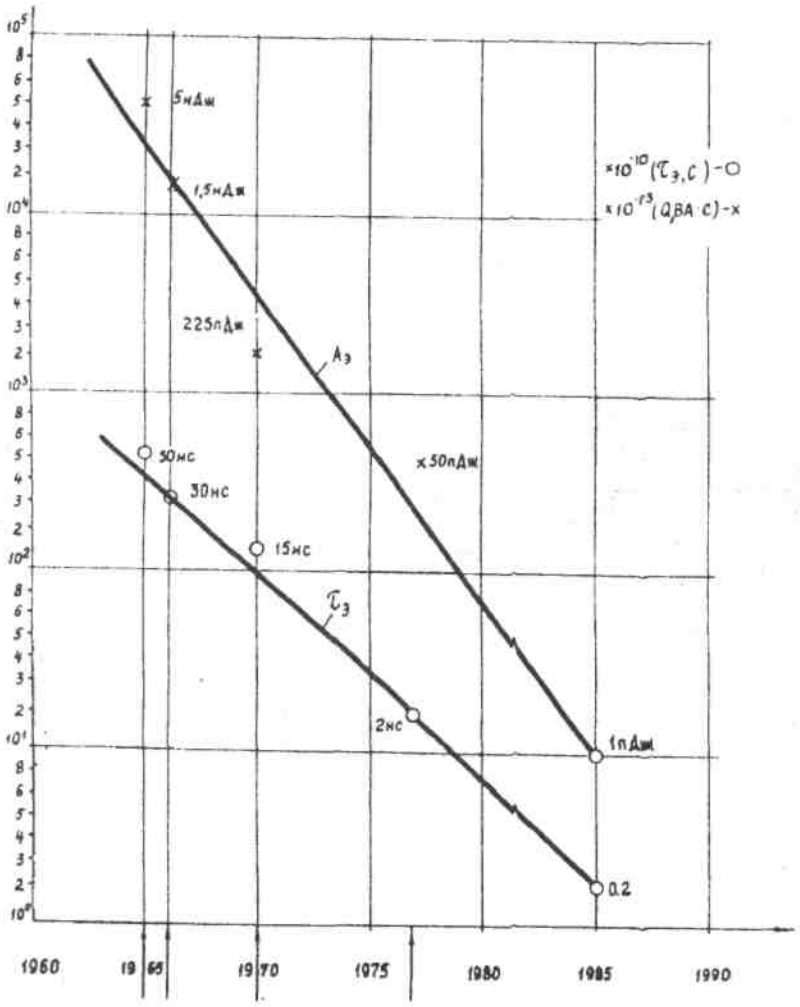


Рис.1. Улучшение энергетических и временных возможностей элементов отечественных ЭВМ.

На Рис.1 представлены графики изменения величин $\tau_Э$ и $A_э$ в процессе развития вычислительной техники в СССР (на полупроводниковой базе). Экстраполируя, можно оценить, что элементы с параметрами $A_э$ = 1 пДж, $\tau_Э$ = 0,2 нс будут работать в машинах в 1985 году.

Предельное быстродействие процессора с использованием таких элементов достигнет 100 млн. оп/с ($T_{ОП\ ПР}$ = 10 нс, при n = 40), причем, как следует из формулы (5), должна быть достигнута плотность компоновки элементов свыше 1000 вент./см^2.

Определяя предел уменьшения времени обращения к оперативному запоминающему устройству $T_{ОП\ ПР-ОЗУ}$, прежде всего следует обратить внимание на величину $T_{СВ\ ПР-ОЗУ} = L_{СВ\ ПР-ОЗУ}\ t_0$- Здесь $L_{СВ\ ПР-ОЗУ}$ - максимальная для данной конструкции длина связи между процессором и ОЗУ, а t_0 - время задержки в кабельных связях на единицу длины (t_0 = 5 нс/с). За все время развития высокопроизводительных машин суммарная площадь, занимаемая процессором и ОЗУ, не имела тенденции к уменьшению. Для различных машин $L_{СВ\ ПР-ОЗУ}$ составляет 10-15 м и соответственно $T_{СВ\ ПР-ОЗУ}$ = 50-75 нс. Объяснить это можно тем, что отношение емкости памяти к производительности машины Q/П на протяжении всех 25 лет практически не изменялось (Q/П = 40-160 тыс. слов/млн. оп./с).

Таблица 1. Изменение быстродействия и емкости памяти в быстродействующих ЭВМ за период с 1951г. по 1975г.

Название ЭВМ	Год выпуска	Быстродействие П оп/с	Емкость ОЗУ Q тыс.слов	Вр. выборки, мкс	Q/П
UNIVAC 1 (remindton rand sperry rand)	1951	0,008	1 (ртутные линии задержки)	120	125
IBM-709	1958	0,2	4-32	6	160
БЭСМ-2	1958	0,015	2	10	135
БЭСМ-6	1967	1	32-131	2	130
CDC-760	1970	12	65-512	1,7	42
IBM-370/195	1973	7	131-524	0,4	74
AMDAHL 470 v/6	1975	7-8	131-1000	0,2	125

В то же время за истекший период на три порядка возросло быстродействие машин, что потребовало соответствующих увеличений емкости ОЗУ (Таблица 1).

Увеличение быстродействия процессора осуществлялось в основном за счет увеличения скорости работы элементов, поэтому их объемы по сравнению с объемами ОЗУ значительно уменьшились, имея в виду повышение интеграции логических элементов за этот период на 2-3 порядка. Так, если в машинах раннего периода площадь, занимаемая процессором, - Snp и площадь ОЗУ - $S_{ОЗУ}$ были равными, то теперь $S_{ОЗУ}$ в 2-10 раз больше Snp.

Таким образом, в современных и будущих вычислительных системах величина $L_{СВ\ ПР-ОЗУ}$ будет определяться габаритами памяти, и ввиду постоянства отношения Q\П нет никаких оснований рассчитывать на ее уменьшение.

Если бы память в 10^6 слов представляла собой единый блок, то в линиях связи внутри этой памяти задержка сигнала $\tau_{св\ зп}$ была бы много больше, чем $T_{СВ\ ПР-ОЗУ}$. Поэтому и по ряду других конструктивно-технологических причин память такой емкости компонуют из отдельных запоминающих модулей емкостью в 16-64 тыс.слов, имеющих автономное управление.

Величину модуля выбирают обычно таким образом, чтобы удовлетворялось соотношение $\tau_{св\ зп} < 2T_{СВ\ ПР-ОЗУ}$.

Таким образом, минимальное значение $\tau_{св\ зп}$ можно оценить величиной 15-20 нс. Соответствующие этой задержке длины связи внутри запоминающего

блока обеспечиваются только в том случае, если плотность компоновки запоминающих элементов будет почти на два порядка выше, чем компоновка логических элементов в процессоре. Поэтому вопрос отвода тепла здесь также имеет принципиальное значение несмотря на то, что в режиме хранения запоминающий элемент потребляет значительно меньше энергии, чем логический.

Для полупроводниковой памяти середины 80-х годов можно считать вполне достижимым $\tau_{зп}$ = 10-20 нс. В этом случае минимальный период выполнения операции обращения к памяти будет $T_{ОП\ ОЗУ}$ = 30 нс (при К = 20), а время $T_{ОП\ ПР\text{-}ОЗУ}$ = 80 нс. Эта величина почти целиком определяется задержками в линиях связи ($T_{СВПР\text{-}ОЗУ}$ и $\tau_{св\ зп}$) и мало зависит от времени срабатывания элементов.

Подставляя полученные минимальные значения времен выполнения операций ($T_{ОП\ ПР}$ и $T_{ОП\ ПР\text{-}ОЗУ}$) в формулу (4) и принимая R = 0,1, получим, что Топ > 30 нс, а соответствующее быстродействие системы процессор-ОЗУ < 30 млн.оп/с.

Однако учитывая, что производительность определяется не только временем Топ. поскольку существуют участки программ, позволяющие использовать запараллеливание работы устройств, можно оценить предел производительности системы процессор-ОЗУ для полупроводниковой элементной базы в 40-60 млн.оп/с.

Достижение этого предельного быстродействия определяется не столько сокращением времени срабатывания логических элементов и элементов памяти, сколько решением вопросов сокращения связей при их коммутации (повышение степени интеграции) и непосредственно связанными с этими вопросами отвода тепла и уменьшения энергии, затрачиваемой на запоминание единицы информации и одно логическое срабатывание элемента.

Приведенный анализ убедительно показывает, что высокопроизводительные ЭВМ, разрабатываемые на полупроводниковой элементной базе, в недалеком будущем исчерпают имеющийся небольшой запас повышения производительности, дальнейшее же развитие высокопроизводительных систем должно основываться на структурных методах увеличения быстродействия комплекса.

В настоящее время известны два принципиально различных структурных способа повышения производительности вычислительного комплекса:

1) объединение нескольких машин в многомашинный комплекс;
2) создание многопроцессорного комплекса.

Можно привести четыре основных довода в пользу многопроцессорных систем:

1. Решение фундаментальных проблемных задач и организация решения задач в режиме коллективного пользования сводится к выполнению большого числа вычислительных процессов, использующих общие данные. Действительно, используются одни и те же системные программы (операционная система, система автоматизации программирования и так далее), пакеты прикладных программ и общие данные параллельно выполняемых задач. Имея в виду, что основную стоимость вычислительных машин составляют запоминающие устройства, дублирование памяти в многомашинном комплексе чрезвычайно нерентабельно.

2. Вычислительные процессы, использующие общие данные, должны определенным образом синхронизироваться. В многомашинном комплексе это требует синхронизации работы отдельных машин и их операционных систем, что вызывает недопустимые потери производительности всего комплекса.

3. Многопроцессорные комплексы повышают эффективность использования оперативной и внешней памяти за счет значительного снижения существую-

щей неравномерности использования памяти во времени задачами. Обоснование этого положения будет приведено ниже.

4. Многопроцессорный комплекс органически определяет модульный принцип построения вычислительной системы с обезличенным использованием однотипных модулей. Поэтому обеспечение высоких требований надежности вычислительного процесса и сохранности данных гораздо эффективнее реализуется на многопроцессорном комплексе, чем на многомашинном. Одновременно с этим многопроцессорный комплекс, построенный по модульному принципу, позволяет осуществлять постепенное наращивание вычислительных мощностей без изменения математического обеспечения и перерыва работы введенных средств. Структура такого комплекса выбирается с учетом специфики решаемых задач с целью наиболее эффективного использования оборудования.

Противники многопроцессорного направления ставят вопрос о распараллеливании алгоритмов. Действительно, есть такие задачи, отдельные алгоритмы которых принципиально не распараллеливаются, и не решен вопрос автоматического распараллеливания алгоритмов в языках.

Однако:

а) опыт эксплуатации больших вычислительных машин определил два основных режима коллективного их использования: пакетная обработка и режим разделения времени. Естественно, что при использовании вычислительной системы в этих режимах вопрос о распараллеливании алгоритмов не возникает;

б) опыт построения больших управляющих систем говорит о том, что процесс управления распадается на целый ряд автономных задач управления отдельными устройствами и задач обработки входных и выходных параметров многочисленных объектов;

в) опыт создания автоматизированных систем проектирования говорит о том, что качество и возможности проектирования во многом определяются производительностью, которая тратится на обсчет отдельных вариантов конструктивно-технологических решений с целью выбора лучшего;

г) не вызывает сомнения возможность распараллеливания многих известных задач прикладной физики и обработки большого количества измерений, в частности задач, решаемых в интересах геодезии.

Перечисленные выше сферы использования вычислительных систем имеют чрезвычайно большое значение в деле развития народного хозяйства. Все это открывает широкую дорогу использованию многопроцессорных систем в недалеком будущем.

Наиболее уязвимым местом многопроцессорной структуры безусловно является общая оперативная память. Возникает два основных вопроса:

1. Не придется ли в угоду постоянства традиционного соотношения Q/ПΣ (ПΣ - суммарная производительность процессоров) пропорционально количеству процессоров увеличивать емкость памяти.

2. Не приведет ли одновременное обращение к общей памяти нескольких процессоров к значительному снижению производительности системы.

Решение этих вопросов требует коренного пересмотра принципов построения процессоров и системы в целом. Однако опыт эксплуатации и конструирования высокопроизводительных машин создает определенные предпосылки к успешному разрешению этих вопросов. Фактически задача сводится к возможности рационального распределения *статического* и *динамического* ресурсов памяти. Под *статическим* ресурсом имеется в виду емкость памяти в режиме хранения. *Динамический* ресурс характеризуется вероятностью того, что при обращении к памяти по записи или считыванию одного из потребителей (процессором, мультиплексором и так далее) она окажется не занятой во

времени другим. Таким образом, использование *динамического* ресурса памяти со стороны потребителя тем больше, чем большее количество обращений в единицу времени они осуществляют к ней.

Прежде всего необходимо отметить, что увеличения емкости памяти, пропорционального количеству процессоров, не потребуется хотя бы из-за того, что системные программы и общие для выполняемых задач данные нет необходимости дублировать ни в оперативной, ни во внешней памяти. Однако в этом случае система команд должна обеспечивать повторную входимость программ.

Эффективное распределение *статического* ресурса памяти в многопроцессорных системах приобретает первостепенное значение, так как всякий преждевременный вызов данных в оперативную память ограничивает возможности работы других процессоров.

Таким образом, задача сводится к эффективному динамическому перераспределению операционной системой *статического* ресурса памяти между вычислительными процессами учитывая то, что статическое использование оперативной памяти задачами во времени чрезвычайно неравномерно. Большинство задач на подавляющем числе временных интервалов использует память небольшой емкости и только на отдельных пиковых участках времени эта емкость увеличивается.

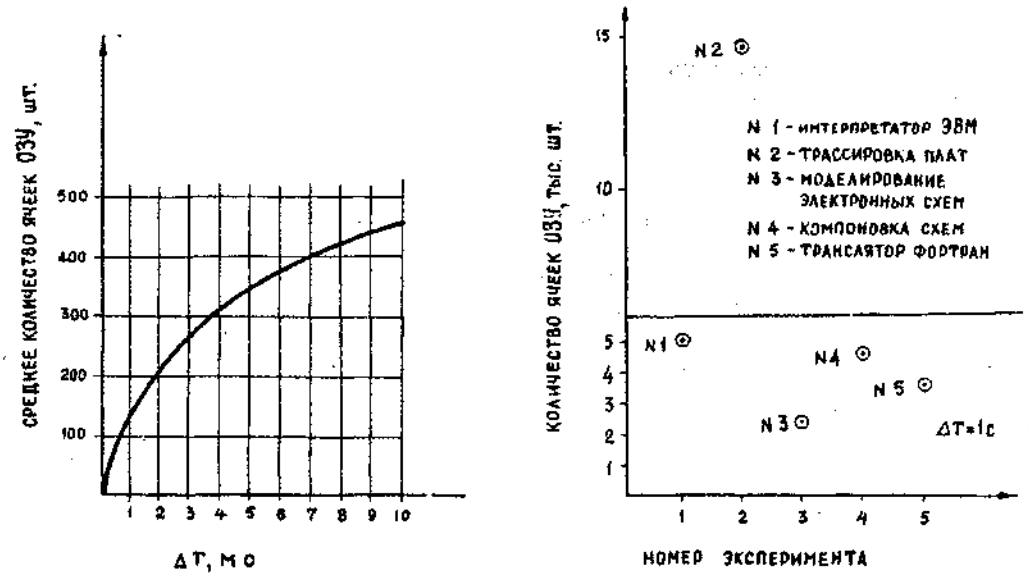


Рис.3. Использование ячеек памяти процессором

Так, анализ работы БЭСМ-6 на различных задачах (Рис.3) говорит о том, что средняя величина использования ОЗУ на интервале $\Delta T = 1$ с не превосходит 5 тыс. ячеек, в то время как имеется пик (в случае программы трассировки) до 15 тыс. ячеек. В то же время зависимость средних значений использования емкости памяти при изменении временного интервала ΔT от 1 мс до 10 мс говорит о том, что при увеличении ΔT происходит постепенное насыщение средней используемой емкости памяти.

Интервал времени переключения с процесса на процесс безусловно зависит от алгоритма задачи и емкости выделенной для ее решения памяти. Поэтому, чем меньше время, затрачиваемое на переключение с задачи на задачу, тем меньший

интервал непрерываемой работы ΔТ можно допустить, а следовательно, меньшая емкость оперативной памяти будет затрачиваться на решение одной задачи. Поэтому эффективность использования статического ресурса памяти во многом зависит от времени переключения системы с одного вычислительного процесса на другой, которое определяется скоростью переключения с задачи на задачу самого процессора и такими параметрами промежуточной и внешней памяти, как время доступа и темп обмена.

Так, система с быстродействием одного процессора, соответствующего процессору БЭСМ-6, имея среднее время переключений сотни микросекунд, могла бы на одну задачу в среднем тратить порядка тысячи ячеек (без учета системных затрат).

Этот прикидочный расчет определяет только порядок величин при заданных временах переключения в условии хорошо сбалансированного темпа обмена информацией оперативной и внешней памяти. Необходимо отметить, что при определении емкости памяти вычислительной машины пользователь обычно исходил из того требования, чтобы емкость оперативной памяти не снижала производительности ЭВМ при решении задач различных классов. В тех случаях, когда время решения какого-то класса задач можно было резко сократить за счет увеличения емкости оперативной памяти, принималось решение о ее наращивании несмотря на то, что средней коэффициент использования памяти был чрезвычайно мал. Некоторым оправданием этого положения было бы то, что во всех остальных случаях оперативная память использовалась в качестве буфера внешней памяти с целью снижения требований к ее основным параметрам: времени доступа и темпа обмена. Именно таким подходом можно как-то объяснить величину существующего соотношения Q/П. По иному обстоит дело в многопроцессорном комплексе, где в случае необходимости операционная система может отдать на требуемое время всю память в распоряжение одной задачи.

Все это говорит за то, что при определении емкости памяти многопроцессорного комплекса нет оснований ее статический ресурс увеличивать пропорционально числу процессоров. В то же время эффективность использования памяти во многом определяется методом ее распределения, заложенным в аппаратуре и операционной системе. Очевидно, тот метод будет эффективнее работать в многопроцессорном комплексе, который в большей мере позволит выполнить следующие требования:

а) должен быть вызван такой квант данных, вероятность использования которого в последующий момент наибольшая;

б) требуемый квант данных должен быть вызван в самый последний момент, и должна иметься возможность освобождения памяти в любой момент после ее использования;

в) должна иметься возможность перераспределять данные внутри памяти или сбрасывать их на внешнюю память с последующим вызовом;

г) должна быть предусмотрена возможность размещать большие массивы, представляющие единую линейную последовательность данных, в различных участках памяти отдельными квантами.

Ни одна из существующих однопроцессорных систем не удовлетворяет всем этим требованиям. Введение математической памяти в какой-то мере решает вопрос свободного распределения памяти (последние два пункта), однако ни в коей мере не позволяет удовлетворить требования первых двух пунктов, существенно влияющих на эффективность использования памяти.

При выборе емкости страницы математической памяти приходится учитывать то обстоятельство, что чем меньше страница, тем большая емкость памяти должна быть отведена операционной системе на их таблицы.

С другой стороны, так как страница определяет квант данных, которыми происходит обмен между внешней и оперативной памятями, увеличение емкости страницы резко снижает эффективность использования последней, имея в виду и то, что решение о вызове страницы в оперативную память принимается по малообоснованному критерию - хотя бы одно обращение к одному из данных внутри страницы. Статистические данные, полученные для большого количества разнообразных задач, показывают, что при емкости страницы в 256 слов только 25% оперативной памяти используется эффективно, в остальных случаях либо данные вызываются напрасно, либо вызов их оказывается преждевременным. Очевидно, источники информации о том, какие данные наиболее вероятно потребуются программе для работы и в каком объеме, содержатся в самом алгоритме. Поэтому метод подкачки данных квантами произвольных объемов, описанных при составлении программы на основании алгоритма решенной задачи, будет наиболее полно удовлетворять требованиям пунктов а) и б).

Вопрос распределения динамического ресурса оперативной и сверхоперативной памяти имеет первостепенное значение при построении многопроцессорных комплексов, так как его дефицит может привести к определенному ограничению их максимальной производительности.

Существенным в этом плане является значительное сокращение числа обращений к оперативной памяти за счет аппаратной реализации часто встречающихся в вычислительных процессах функций, ранее выполнявшихся программным способом. Аппаратная реализация языков высокого уровня, аппаратная реализация функций распределения оперативной памяти, функций программ диспетчера операционной системы, защиты данных, синхронизации работы процессоров при обращении к общим данным и так далее значительно сокращает количество обращений к оперативной памяти. Достаточно сказать, что только аппаратная реализация языков высокого уровня позволяет на отдельных участках программ в 2-3 раза сократить емкость памяти для ее написания и в несколько десятков раз сократить количество обращений к оперативной памяти при выполнении одного и того же вычислительного процесса.

Сверхоперативная память (СОЗУ) в многопроцессорных системах приобретает особое значение, так как наряду с сокращением времени обращения к данным она резко снижает средний темп обращения процессора к оперативной памяти. Наиболее естественным структурным решением было бы создание единой сверхоперативной памяти для всех процессоров. Однако такое решение не приемлемо в основном по двум причинам:

а) узким местом всей системы будет динамический ресурс этой сверхоперативной памяти;

б) недопустимо увеличивается время операции обращения к этой сверхоперативной памяти за счет потерь в линиях связи процессор - СОЗУ.

Длина этих линий связи будет пропорциональна площади, занимаемой всеми процессорами. Учитывая, что время операции полупроводниковой СОЗУ в недалеком будущем может определяться в 5-10 нс, объединение СОЗУ теряет смысл. В то же время построение индивидуальных СОЗУ для каждого процессора вызывает определенные принципиальные трудности при работе процессоров с общим данными. Поэтому структура сверхоперативной памяти в многопроцессорном комплексе существенно отличается от традиционных структур однопроцессорных систем.

С несколько иных позиций должна быть определена структура многопроцессорных вычислительных комплексов и в части подсоединения спецпроцессоров, таких как процессор ввода-вывода, осуществляющий управление внешними устройствами, процессор приема-передачи данных, управляющий терминальным оборудованием, и другие. Так, вопрос об использовании спецпроцессорами центральной памяти комплекса должен быть решен с учетом эффективного использования как статического, так и динамического ресурса этой памяти всей системой. Анализ структуры с этих позиций позволяет сформулировать следующие общие принципы использования оперативной памяти спецпроцессорами в многопроцессорном комплексе: те данные, к которым спецпроцессор обращается немногократно, целесообразно помещать в центральную оперативную память; данным же, требующим многократного обращения, должна быть выделена локальная память спецпроцессора.

При построении общей системы коммутации комплекса (процессоры - ОЗУ - спецпроцессоры) особое внимание должно быть уделено обеспечению максимального использования динамического ресурса, заложенного в этих устройствах. Так, например, ОЗУ больших систем по конструктивно-технологическим соображениям компонуется из нескольких однотипных модулей. Система коммутации комплекса должна быть построена таким образом, чтобы, используя это свойство, ОЗУ был обеспечен максимальный темп обмена с процессором.

Все вышеизложенное позволяет сделать следующие выводы:

1. Многопроцессорные комплексы позволяют:

а) значительно расширить физические пределы быстродействия вычислительных средств;

б) существенно увеличить эффективность использования оборудования вычислительных комплексов;

в) обеспечить заданную структурную надежность;

г) осуществлять постепенное наращивание вычислительных средств и адаптацию структуры комплексов под решаемые задачи.

2. Построение многопроцессорных вычислительных комплексов предъявляет новые требования как к организации работы входящих в них устройств, так и к системе управления в целом, включая операционную систему.

Принципы построения многопроцессорных вычислительных комплексов "Эльбрус"

В.С.Бурцев

При обсуждении проблем развития вычислительной техники неоднократно ставился вопрос о создании таких вычислительных комплексов, которые наряду с существенным повышением эффективности труда программистов, в особенности при создании сложных программ, обеспечили бы сохранность математического обеспечения, разработанного на их базе, при переходе в будущем на новые более перспективные конструкции вычислительных средств. Предлагаемая структура многопроцессорного вычислительного комплекса (МВК) "Эльбрус" будет, очевидно, в первую очередь обсуждаться именно с этих позиций. Тем не менее, при изложении принципов построения МВК "Эльбрус" следует остановиться на тех вопросах, которые во многом определяют правильность выбранных решений при создании комплекса:

- эффективность использования оборудования;
- возможность обеспечения предельной производительности;
- создание высоконадежных резервируемых структур, обладающих возможностью постепенного наращивания производительности с учетом адаптации к решаемым задачам.

Оценку принятых в МВК "Эльбрус" структурных решений, с точки зрения лучшей реализации вышеперечисленных вопросов, целесообразно провести в сравнении со структурами традиционных универсальных сверхбольших ЭВМ передовых зарубежных фирм IBM и CDC.

МВК "Эльбрус" построен по модульному принципу и в зависимости от комплектации может состоять из некоторого числа модулей процессора (1-10), модулей оперативной памяти (4-32), модулей процессоров ввода-вывода (1-4), различного количества модулей внешней памяти (барабанов, дисков, магнитных лент), модулей процессоров приема-передачи данных (1-16) и устройств ввода-вывода, подключенных либо непосредственно к ПВВ, либо через линии передачи данных посредством ППД (Рис.1). Каждый модуль комплекса, включая разнесенные по ним узлы центрального коммутатора, имеет 100%-ный аппаратный контроль и в случае появления хотя бы одиночной ошибки в процессе прохождения вычислительного процесса выдает сигнал неисправности данного модуля. По этому сигналу операционная система через аппаратно реализованную систему реконфигурации исключает неисправный модуль из работы. Отключенный модуль попадает в ремонтную конфигурацию, в которой посредством тест-диагностических программ и специальной аппаратуры ремон-

тируется, после чего может быть включен операционной системой в рабочую конфигурацию.

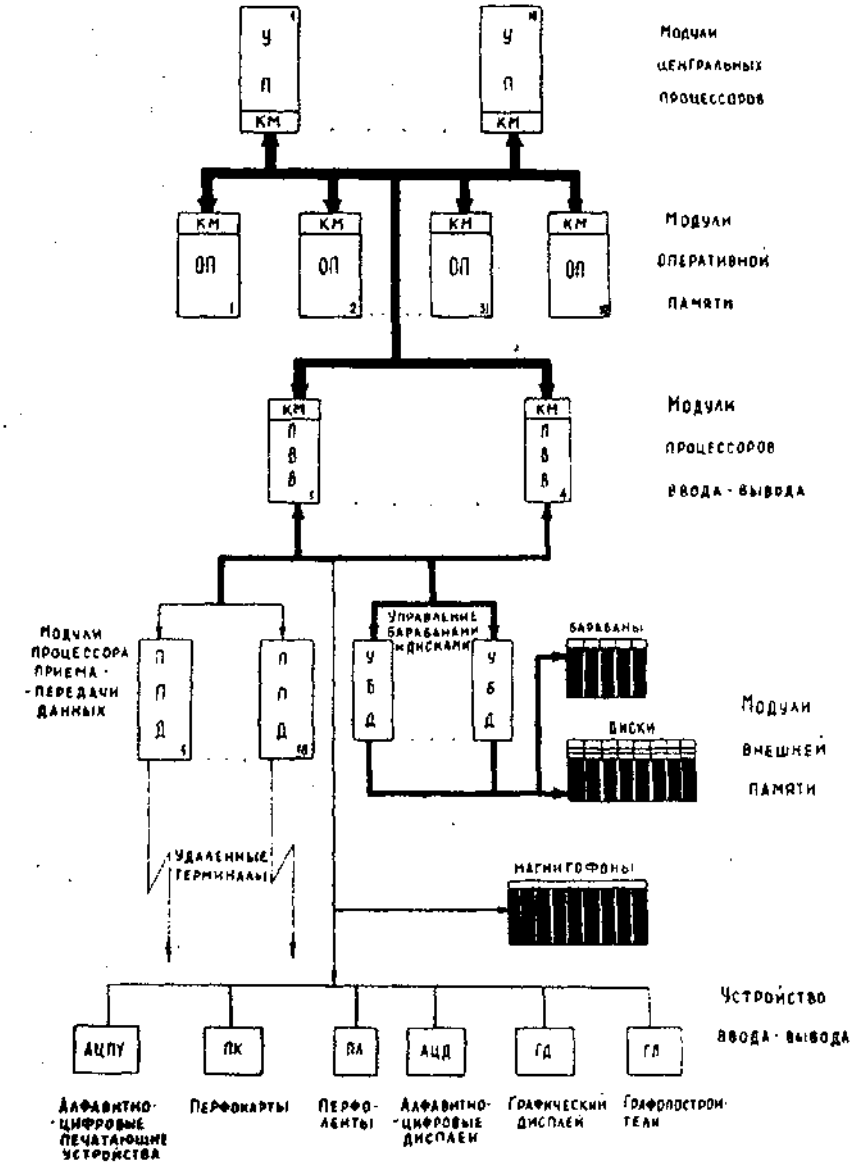


Рис.1. Блок-схема МВК «Эльбрус».

Такая система позволяет осуществить резервирование на уровне однотипных модулей устройств и обеспечить заданную надежность и живучесть комплекса значительно экономичнее, чем это делается в многомашинных комплексах, где резервирование осуществляется на уровне однотипных машин. Более того, модульный принцип позволяет в зависимости от специфики решаемых задач за счет различного соотношения в комплектации модулей процессоров, оперативной памяти, внешней памяти, процессоров передачи данных

и так далее, настроить комплекс на решение счетных, информационных или других задач. Аналогичных возможностей ни однопроцессорные, ни многомашинные комплексы не имеют.

Решение вопроса достижения предельного быстродействия в последнее время стоит особо остро, так как полупроводниковая техника ввиду принципиальных физических ограничений, таких как предельно допустимые напряженности полей в структурах и плотности токов в линиях связи внутри приборов, не сможет обеспечить время задержки элемента меньше 0,2 нс при 5 мВт мощности, выделяемой на один логический вентиль. В то же время задержки в линиях связи даже при очень больших плотностях монтажа элеме-тов не позволят пропорционально увеличению быстродействия элементов увеличить производительность однопроцессорного вычислительного комплекса. Оптимистические оценки в части достижения предельной производительности процессора при работе его с ОЗУ не превышают 60 млн. оп/с.

Поэтому наиболее кардинальным и эффективным структурным способом достижения максимального быстродействия вычислительной системы надо считать создание многопроцессорных комплексов.

Достижение этих же целей посредством многомашинных систем будет менее эффективно по следующим соображениям:

1. Решение фундаментальных проблемных задач, задач систем автоматизации проектирования крупных объектов и целый ряд других не менее крупных задач сводится к вычислению большого числа вычислительных процессов, использующих общие данные. Действительно, имея в виду работу вычислительных средств, хотя бы в режиме пакетной обработки, для решения этих задач используются одни и те же системные программы (операционная система, система автоматизации программирования и так далее), пакеты прикладных программ и общие данные одновременно решаемых задач. В многопроцессорном комплексе нет необходимости дублировать эти данные в памяти, в то время как в многомашинном комплексе все эти данные должны содержаться в каждой памяти машины.

2. Вычислительные процессы, использующие общие данные, должны определенным образом синхронизироваться. В многомашинном комплексе это требует синхронизации работы отдельных машин посредством их операционных систем, что, как показывает практика, вызывает недопустимые потери производительности всего комплекса.

В многопроцессорном комплексе эта синхронизация происходит внутри единой для всего комплекса операционной системы и сводится к синхронизации работы процессоров на единую память. Такая синхронизация в МВК "Эльбрус" в значительной степени решена аппаратными средствами.

3. Многопроцессорные комплексы существенно повышают эффективность использования оперативной и внешней памяти за счет усреднения во времени объемов памяти, занимаемых отдельными задачами.

Статистика показывает, что большинство задач на подавляющем числе временных интервалов требует памяти небольших объемов, и только на кратковременных пиковых участках времени этот объем увеличивается. Чтобы не снизить скорости решения задач из-за памяти, объем ее в однопроцессорных системах выбирался по пиковым участкам использования.

В многопроцессорных системах в случае затребования задачами небольших объемов памяти имеется возможность решать их одновременно, в то же время в случае необходимости на требуемый интервал времени вся память может быть отдана в распоряжение одной задачи.

Максимальное количество одновременно решаемых задач определяется средней величиной локальной памяти, отводимой для ее решения. Этот объем во многом определяется средним временем переключения системы с задачи на задачу. Объем локальной памяти должен быть таким, чтобы среднее время работы задачи без прерывания было Много больше времени переключения системы. Среднее время переключения системы с задачи на задачу определяется временем переключения самого процессора и такими параметрами промежуточной внешней памяти; как время доступа и темп обмена.

Поэтому в МВК "Эльбрус" приняты меры к сокращению времени переключения процессора и разработана промежуточная память на магнитном барабане, обеспечивающая высокий темп поступления информации 40 млн. бит/с и малое среднее время доступа менее 50 мс.

Безусловно, для реализации вышеописанных возможностей использования памяти в вычислительном комплексе должны быть предусмотрены эффективные средства динамического распределения памяти *{статического* ресурса памяти) между вычислительными процессами.

В то же время нельзя не отметить, что наиболее уязвимым местом многопроцессорной структуры, с точки зрения обеспечения высокой производительности комплекса, является общая оперативная память. Возникает весьма резонный вопрос, не приведет ли одновременное обращение к общей оперативной памяти нескольких процессоров к значительному снижению производительности комплекса в целом. Фактически задача сводится к возможности рационального распределения *динамического* ресурса памяти. Динамический ресурс характеризуется вероятностью того, что при обращении к памяти одного из потребителей (процессора, мультиплексора) она не окажется занятой по времени другим.

Таким образом, использование динамического ресурса памяти со стороны потребителя тем больше, чем большее количество обращений в единицу времени он осуществляет к ней.

Эффективным решением снижения интенсивности обращений к оперативной памяти со стороны потребителей является принцип аппаратной реализации наиболее устоявшихся во времени программных блоков, узлов и конструкций, таких как:

- типовые конструкции программ, созданных трансляторами (аппаратная реализация языков высокого уровня);
- функции программ распределения оперативной памяти и функции программ диспетчера (аппаратная реализация операционной системы);
- функции программ, обеспечивающих защиту данных и синхронизацию работ процессора при обращении к ним и так далее.

С точки зрения наиболее рационального использования динамического ресурса памяти, сверхоперативная память многопроцессорных комплексов приобретает особое значение, так как наряду с сокращением времени обращения к данным она резко снижает общее число обращений процессора к памяти.

Структура МВК "Эльбрус" в части подключения процессоров ввода-вывода, процессоров приема-передачи данных должна быть также решена с учетом наиболее эффективного использования как *статического,* так и *динамического* ресурса памяти всего комплекса.

При построении общей системы коммутации комплекса особое внимание должно быть уделено обеспечению максимального использования *динамического* ресурса, заложенного в конструкцию этих устройств. Так, например, ОЗУ больших вычислительных комплексов по конструктивно-технологическим соображениям компонуется из нескольких автономных по управлению одно-

типных блоков. Система коммутации комплекса должна быть построена таким образом, чтобы при использовании этого свойства ОЗУ был обеспечен максимальный темп обмена с процессорами.

Выше были изложены основные отличительные особенности, которые должны быть учтены при построении многопроцессорного комплекса "Эльбрус". Остановимся несколько более подробно на реализации этих особенностей в отдельных устройствах и блоках МВК "Эльбрус".

I. Особенности системы команд

Сравнение системы команд МВК "Эльбрус" с традиционными системами команд, принятыми в машинах фирм IBM и CDC, можно провести, анализируя следующие основные структуры данных:

МВК «Эльбрус» **CDC, IBM**

Первая структура

СЛОВО

ТЭГ	64 бита

Максимальный дискрет информации в слове БИТ
ТЕГ - определяет тип и формат данных в слове

СЛОВО

60.64 бита

Максимальный дискрет информации в слове БАЙТ

Вторая структура

БЕЗАДРЕСНЫЙ ФОРМАТ КОМАНДЫ RR

КОП

КОП - код операции

ФОРМАТ КОМАНДЫ ТИПА

КОП	R_1	R_2	R_3

R_1, R_2, R_3 - физические адреса быстрых регистров

Третья структура

ФОРМАТ КОМАНДЫ С АДРЕСОМ

КОП	ИМЯ (N,I)

N - уровень программы
I - порядковый номер слова

ФОРМАТ КОМАНДЫ RX, RS

КОП	R_1	R_2	R_3	АДРЕС

АДРЕС - физический или математический адрес

Четвертая структура

ОПИСАТЕЛИ

а) метка процедуры

НОМЕР БАЗЫ	NK	АДРЕС

АДРЕС - математический адрес данных процедуры

ОТСУТСТВУЕТ

б) дескриптор

ФОРМАТ	ГРАНИЦА	АДРЕС

АДРЕС - математический адрес начала сегмента
ГРАНИЦА - количество данных в сегменте

Укажем на следующие отличительные особенности структур МВК "Эльбрус".

Первая структура. Информация имеет более плотную упаковку с дискретностью до бита, в то время как традиционной является упаковка до байта.

Каждое слово имеет свой "тег", определяющий тип и формат данных. Наличие тега позволяет:

- осуществить эффективную защиту информации с дискретностью до бита;
- исключить из системы команд лишнюю информацию, указывающую на то, какими аппаратными средствами производить обработку этой информации (АУ с плавающей или фиксированной запятой, АУ для работы над целыми числами и так далее).

Так, например, в традиционной системе команд указывается, какое сложение, умножение или деление производить - с плавающей запятой или с фиксированной. В зависимости от этого выбираются аппаратные средства обработки. В системе команд МВК "Эльбрус" указывается только вид операции: сложение, умножение или деление. Аппаратура обработки выбирается автоматически в зависимости от типа данных операндов.

Вторая структура. Принята безадресная система команд, в результате чего отсутствует лишняя информация о распределении ресурса быстрых регистров. Из этого не следует делать вывод, что быстрые регистры отсутствуют в процессоре.

Эффективное динамическое распределение быстрых регистров реализует аппаратура. Необходимо отметить, что такое решение позволяет значительно увеличить количество быстрых регистров, так как нет требования сокращения их кода адреса с целью сокращения формата команды типа RR. Безадресный формат команд базируется на аппаратной реализации стека.

Третья структура. Сравнивая формат команды МВК "Эльбрус" с форматом команды типа RX и RS в машинах фирм IBM и CDC, помимо отсутствия информации о распределении быстрых регистров, можно видеть существенное смысловое различие в содержании разрядов адреса. Разряды адреса команды содержат указание на использование физического ресурса памяти либо указание адреса в линейном пространстве математической памяти. Разряды адреса команды МВК "Эльбрус" содержат закодированное определенным образом "имя" величины, определенное только в данном контексте программы и указывающее на лексикографический уровень данной программы, и порядковый номер слова в стеке данных этой процедуры (адресная пара N, 1).

Четвертая структура. Система команд предусматривает аппаратный "дистанционный" вызов процедуры. С этой целью введен специальный описатель процедуры - метка процедуры.

Информация, содержащаяся в метке процедуры, указывает: номер базы (номер в справочнике, указывающий на дескриптор, в котором, как правило, содержится адрес начала программы процедуры в оперативной памяти), номер команды (№К) и математический адрес, определяющий контекст процедуры или определенную доступность имен. Аппаратная реализация дистанционного вызова процедур существенно снижает расход памяти как по статическому, так и по динамическому ее ресурсу.

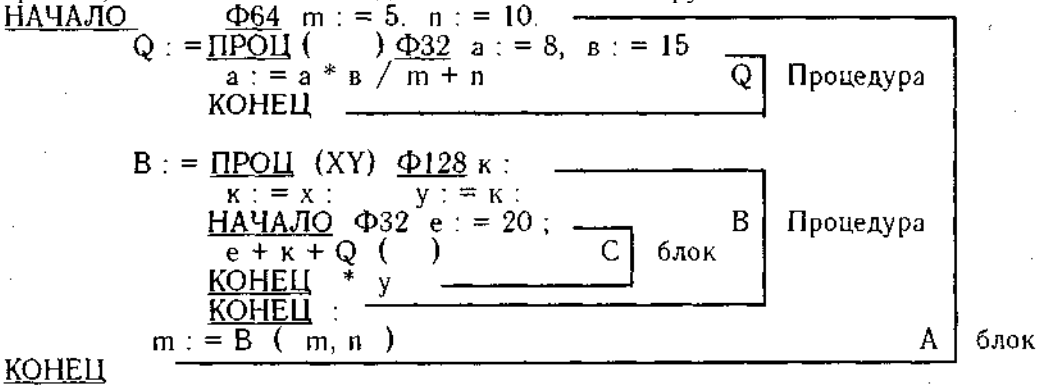
В традиционных машинах подобные аппаратные средства работы с процедурами отсутствуют.

Данные, находящиеся в обработке в вычислительной системе, представлены массивами, описанными дескрипторами. Все данные, за исключением кодов команд, предварительно погружаются в математическую память и в этой памяти описываются дескрипторами. Коды команд загружаются непосредственно в физическую память и соответствующим образом описываются дескрипторами.

В математической памяти данные описываются с дискретностью до бита, в физической - до слова. Данные массива, описываемого дескриптором, имеют единый статус защиты и одинаковый формат. В дескрипторе содержится информация о формате данных, уровне защиты, начальной границе массива и количестве данных в описываемом массиве. Эти описатели широко используютя операционной системой для эффективного распределения ресурса математической и физической памяти и организации защиты данных.

Традиционные машины таких средств распределения ресурса не имеют.

Покажем на примере организацию вычислительного процесса с помощью системы команд МВК "Эльбрус". Для этого рассмотрим прохождение программы, написанной в автокоде системы МВК "Эльбрус".



Символика написания не требует особых комментариев.

Символ "НАЧАЛО" означает начало описания блока, "КОНЕЦ" - завершение описания блока или процедуры. Символы Ф32, Ф64, Ф128 описывают форматы соответствующих локальных переменных. Символ ПРОЦ (X,У) означает начало описания процедуры с формальными параметрами X,У, ПРОЦ ( ) - та же процедура без параметров.

Структура программы предусматривает определенную контекстную защиту имен. Так, для блока С доступны данные процедуры В и блока А и недоступны локальные данные (а и в) процедуры Q. Для процедуры Q, находящейся на одном лексикографическом уровне с В, недоступны данные процедуры В и блока С.

В системе МВК "Эльбрус" программа А по вложенности уровней может быть представлена следующим образом.

На нулевом уровне находятся метки процедуры операционной системы. К этому уровню имеют доступ все программы. Этот уровень, таким образом, обеспечивает прохождение задачи, включая решение всех непредусмотренных пользователем ситуаций, возникающих в процессе ее реализации. В том случае, если уровень ОС не обеспечивает решения задачи, происходит выход на человека.

На первом и втором уровне располагаются описатели процедур выполняемой задачи, причем на первом уровне расположены дескрипторы заранее связанных программ, а на втором уровне располагаются описатели внешних имен, связь с которыми может быть установлена в процессе выполнения задачи. Таким образом, нулевой уровень выполняемой задачи (в данном случае блока А) в МВК "Эльбрус" будет третьим.

В блоке А описаны его локальные данные m, n, Q и В и указано единственное действие - m присвоить значение процедуры В, подсчитанное для параметров m и n.

Этот же пример программы в системе команд МВК "Эльбрус" будет иметь следующий вид (в скобках приведены соответствующие отображения на автокоде):

```
ВХОД     НЗЦ5     НЗЦ10    (начало          Ф64   m:=5  n:=10)
ФМ4.1.2.0                  (B: =      тело процедуры B)
ФМ4.1.4.0                  (Q: =      тело процедуры Q)
ЗГАЗ. 1. 3 ГАЗ. 3          }
МАРК ВВ3.1.ВВ3.2. ВХОД     }   (m        :  =B(m, n)
ЗП64                       }
ВЫХОД                      (КОНЕЦ)
```

Операторы m :=5 и n :=10 транслируются в команды НЗЦ5 и НЗЦ10, выполняющие непосредственную загрузку целых 5 и 10.

В следующую ячейку стека должна быть загружена метка, описывающая процедуру B. Это делается командой формирования метки ФМ. Параметрами этой операции являются имя процедуры и величина, определяющая контекстную доступность процедуры B к данным. В метке для организации контекстной доступности имен должен находиться математический адрес стека процедуры, в котором описана процедура B, в данном случае начала стека процедуры A. Уровень описывающей процедуры всегда на единицу меньше описываемой. Поэтому для формирования метки в качестве одного из параметров операции ФМ в тексте программы должен быть указан ее уровень - 4. Имя процедуры описывается указанием уровня 1 или 2 и порядковым номером расположения дескриптора, описывающего программу этой процедуры в стеках этих уровней. В данном случае имя - 1, 2 говорит о том, что дескриптор процедуры расположен в стеке первого уровня на втором месте от начала стека. В том случае, если вычисления начинаются не с первой команды программы, описываемой дескриптором, в команде ФМ указывается значение порядкового номера команды, с которой необходимо начать вычисления. На основании этих данных в стек блока A записывается метка процедуры B.

Аналогичным образом формируется и записывается в стек метка процедуры Q. Для выполнения m := B(m; n) в стек загружается адрес m и метка процедуры (ЗГАЗ.1 и ВВЗ.З). Затем выполняется команда маркировки стека, которая записывает в стек маркер стека процедуры B. Маркер стека является нулевой ячейкой стека процедуры B. До входа в процедуру в маркере содержится вся информация, необходимая для запуска процедуры; после запуска процедуры на этом месте формируется информация, необходимая для возврата управления в место ее запуска. И до и после запуска процедуры в маркере содержится математический адрес, определяющий адресное окружение выполняемой процедуры (статическая цепочка). После маркировки стека производится передача формальных параметров (m и n) из блока A в процедуру B. Это выполняется операция "взять величину" ВВ. В качестве операнда этой операции поставляется имя в виде адресной пары - 3.1. и 3.2. Первая цифра указывает уровень процедуры или фактически маркера стека этой процедуры, вторая -смещение этой величины в стеке относительно маркера.

Аппаратно такая адресация реализуется с помощью базовых регистров, в которых хранятся дескрипторы, описывающие стеки процедур, причем порядковые номера регистров соответствуют уровням процедур. Так, регистр № 0 содержит дескриптор, описывающий стек из описателей процедур и данных операционных систем, регистры № 1 и № 2 описывают стеки описателей процедур и внешних имен выполняемой задачи, регистр № 3 содержит дескриптор, описывающий стек данного блока A и так далее.

■ На Рис.2 изображено состояние стека и базовых регистров на момент входа в процедуру B. Для обращения по адресной паре берется начальный математический адрес дескриптора соответствующего уровня и прибавляется порядковый номер величины в стеке данной процедуры.

В момент входа в процедуру B заполняется базовый регистр № 4, после входа в блок C заполняется дескриптор регистра № 5. В блоке C для выполнения операции сложения по имени 3.4 в стек записывается метка Q процедуры, выполняется запуск процедуры, и уже результат этой процедуры используется как операнд. В процессе запуска процедуры (Рис.3) производится аппаратная корректировка содержимого базовых регистров: в четвертый регистр записывается дескриптор, описывающий стек процедуры Q, а пятый регистр стирается.

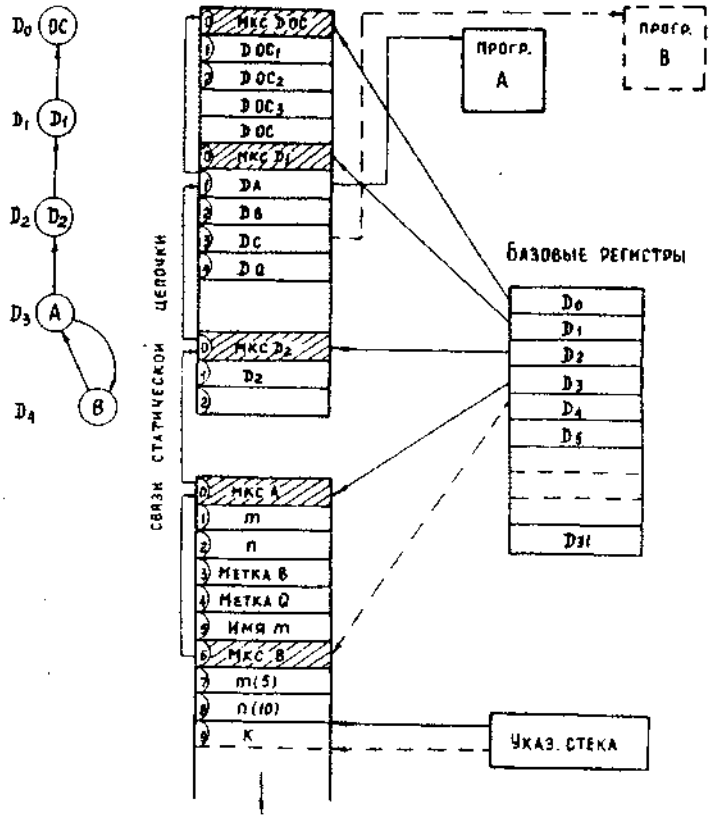


Рис.2.Аппаратная реализация стека (момент перехода к процедуре B).
Do - справочник программных сегментов операционной системы;
D_1 - справочник программных сегментов, скомпонованных в выполняемый файл;
D_2 - справочник программных сегментов других файлов.

Таким образом, исключается доступ процедуры Q в данные процедуры B и блока C, что соответствует контекстной защите имен. После завершения процедуры Q на место ее маркера подставляется результат и происходит корректировка базовых регистров с целью восстановления адресного окружения для продолжения работы блока C. Вся необходимая информация для этого содержится в ячейках, отведенных под маркеры процедур. Память, занимаемая стеком процедуры Q, освобождается.

После выполнения блока C результат его используется в процедуре B, а ячейки стека освобождаются. Аналогичным образом завершается выполнение процедуры B. Результат процедуры B записывается в m последней командой блока A.

Необходимо отметить, что в момент входа в процедуру или блок происходит первое обращение к сегменту команд через дескриптор, находящийся в стеке первого уровня. При этом возможны два случая:

1) сегмент находится в оперативной памяти;
2) сегмент находится в файле внешней памяти.

В первом случае адрес дескриптора используется для изменения базы в счетчике команд. Во втором случае адрес дескриптора указывает на справочник, по которому находится адрес этого сегмента на внешнем носителе, и с помощью операционной системы производится считывание необходимой информации. Адрес первой ячейки физической памяти считанного сегмента заносится в дескриптор, после чего выполняются операции, аналогичные первому сегменту.

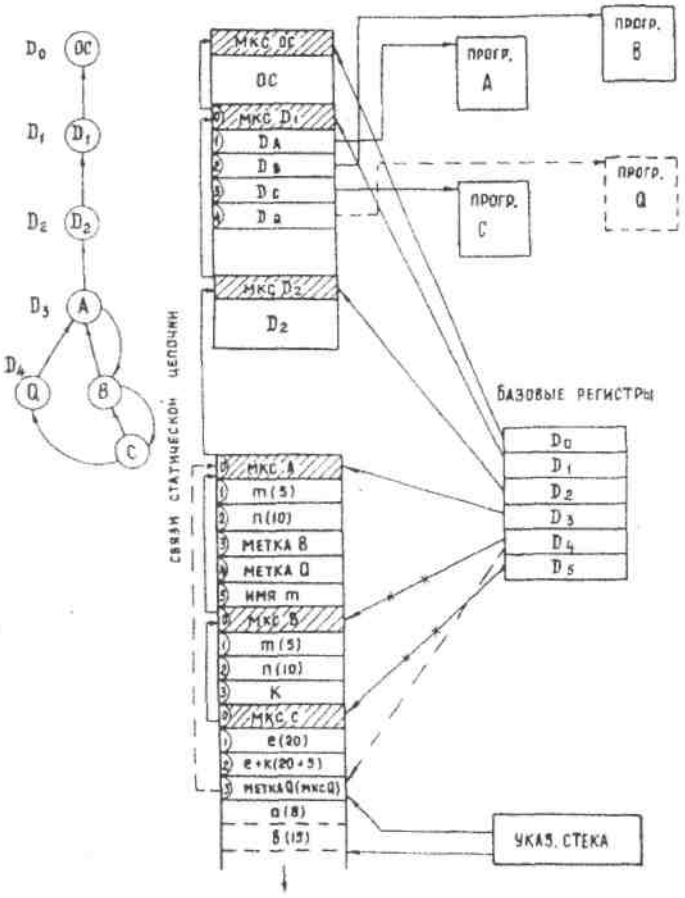


Рис.3. Стек на момент перехода к процедуре Q.

Сегменты использованных программ при необходимости могут забиваться другими данными, так как коды команды остаются неизменными в процессе вычислений и в любой момент имеется возможность их повторно считать.

Если в процедуре объявлен массив данных, то при входе в нее в стек загружается только специальный описатель этого массива. Выделение оператив-

ной памяти и формирование дескриптора, который помещается на место описателя, производится только в момент первого обращения к одному из этого массива элементов. Уничтожение массива (освобождение памяти) выполняется непосредственно перед выходом из процедуры.

Наличие неизменных сегментов программ и аппаратная реализация стека автоматически обеспечивает рекурсию вычислений, так как любое повторное вхождение в программу вызовет маркировку стека и выделение самостоятельного участка памяти для данных повторного использования программы.

Принятая система команд с ее аппаратной реализацией позволяет:

а) существенно сократить объемы программ (статический ресурс) - в 1,5-2 раза;

б) осуществить эффективное динамическое распределение оперативной памяти;

в) сократить в несколько раз число обращение к оперативной памяти (динамический ресурс) за счет аппаратной реализации типовых приемов программирования (стек, вход в процедуру, выход и так далее);

г) обеспечить рекурсии вычислений.

Для подтверждения этих положений приведем пример реализации типовых фрагментов программ в системе команд МВК "Эльбрус", БЭСМ-6 и IBM-360 в виде таблицы 1. Для языков высокого уровня приведены две графы: статическое и динамическое использование памяти. Дело в том, что в процессе прохождения программы в БЭСМ-6 и IBM-360 происходит обращение к административной части системы программирования за типовыми подпрограммами, поэтому в процессе выполнения программы существенно увеличиваются.

Таблица 1.

Алгоритм	IBM									БЭСМ-6					«ЭЛЬБРУС»				
	Алгол 4030		Фортран 1060		Фортран G		Фортран H		Асс	Алгол		Фортран		Асс	Алгол		Фортран		Асс
	С	Д	С	Д	С	Д	С	Д		С	Д	С	Д		С	Д	С	Д	
begin integer X.Y: procedure P(Z): integer Z: value Z: begin real t: t: = Z: end P(Y+Y) end	32	464	128	128	100	100	94	94	14	63	309	5	57	24	12	12	7	7	7
IT: = IX * IY + IZ	30	30	18	18	16	16	16	16	16	12	12	1	57	12	9	9	9	9	9
A (I): = I0	38	126	16	16	38	36	62	50	16	18	75	2	21	18	7	7	7	7	7
Σ	100	620	162	162	154	152	172	160	46	93	396	9	135	54	28	28	23	23	23

Примечания: Асс - Ассемблер. С - статика, Д - динамика

В МВК "Эльбрус" эти типовые приемы выполняются аппаратурой, поэтому разницы между написанной программой и выполняемой нет. Анализ этой таблицы позволяет в какой-то мере количественно оцепить эффективность решений, принятых в МВК "Эльбрус".

2. Синхронизация вычислительных процессов

Аппаратно вычислительный процесс отображается стеком, как правило, находящимся в оперативной памяти. В том случае, если на стеке работает про-

цессор (активный процесс), аппаратное отображение процесса распространяется на базовые регистры, указатель стека, сверхоперативную память и другую аппаратуру процессора. Процесс может находиться в состоянии ожидания какого-либо события (пассивный процесс) и аппаратно отображается только данными в оперативной памяти.

Принятая структура организации вычислительного процесса представляет широкие возможности распараллеливания вычислений.

Так, наряду с тем, что для каждой задачи или комплекса совместно оттранслированных программ может быть заведен самостоятельный стек, каждая процедура может быть реализована на отдельном стеке. В частности, в предыдущем примере, если бы в программе перед процедурой Q стояло указание об организации параллельного процесса, был бы организован новый стек и вычисления, предусмотренные процедурой Q, выполнялись бы параллельно с работами, производимыми на основном стеке (Рис.4), причем на вновь организованном стеке работу производил бы другой процессор, используя свои базовые регистры, указатель стека, сверхоперативную память и так далее.

Уже упоминалось о том, что однотипные модули, в частности процессоры, обезличены, то есть любой процессор работает на любом стеке или выполняет вычисления в интересах любого процесса. Кроме того, так как система многопроцессорная, одновременно в активном состоянии может находиться несколько процессов, работающих над общими данными.

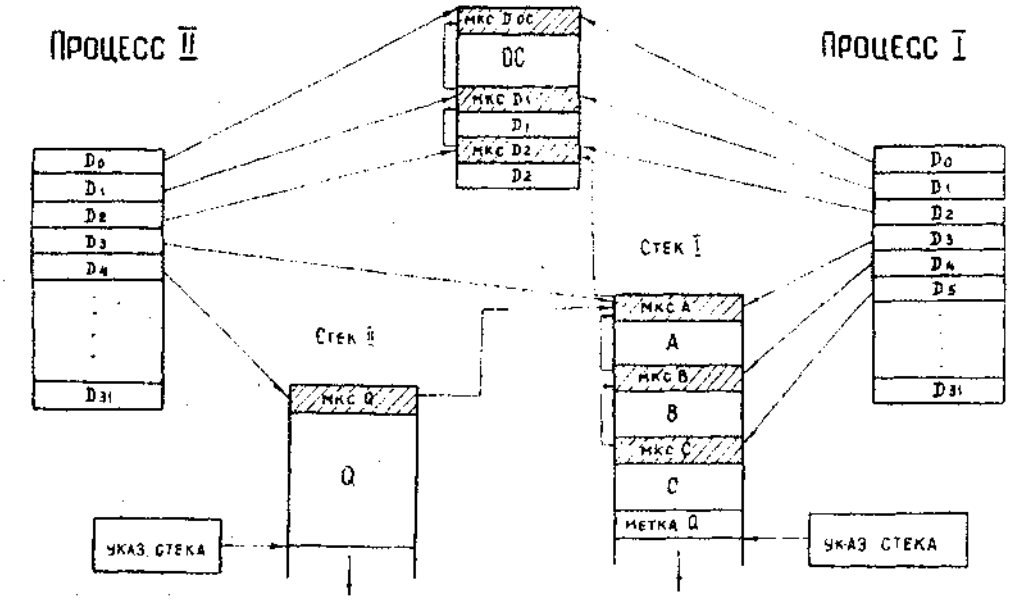


Рис.4. Вызов процедуры Q с образованием нового процесса II.

В МВК "Эльбрус" одним из аппаратных средств обеспечения синхронизации работы процессоров и их обезличенного распределения является применение "семафоров". В систему команд введены такие операции, как "открыть семафор" и "закрыть семафор". Прежде чем программа начнет работу с общими данными, она обращается к общей с другой программой ячейке, называемой семафором, с тем, чтобы его закрыть на время ее работы с этими данными. В том случае, если семафор открыт (нуль в соответствующем разряде ячейки), происходит закрытие

семафора, если же семафор оказывается закрытым, то процесс должен ждать открытия его.

В последнем случае процесс становится пассивным. Процессор освобождается от выполнения этого процесса и заносит в ячейку семафора адрес этого стека. Таким образом, на одном семафоре может образоваться очередь из нескольких ждущих стеков (Рис.5).

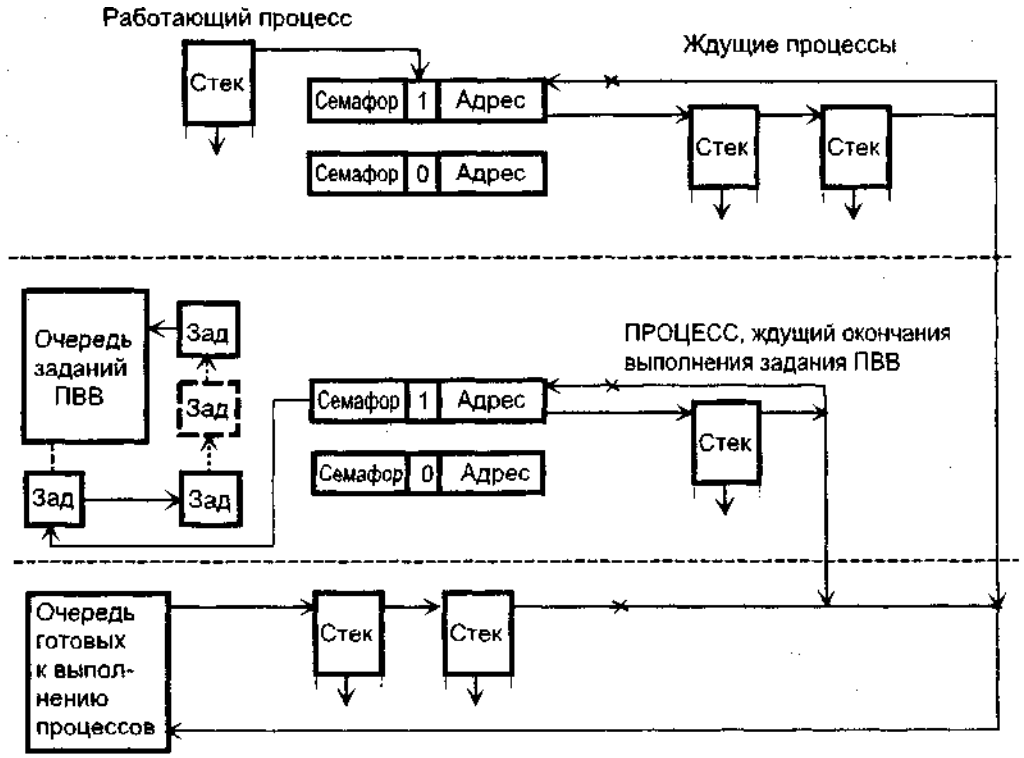


Рис.5. Синхронизация работы процессов при обращении к общим данным и ПВВ.

После завершения работы с данными, процесс, закрывший семафор, открывает его, устанавливая "0" в соответствующий разряд ячейки и время открытия $T_{откр}$ в поле ячейки семафора на место адреса стека текущего процесса. В том случае, если в поле адреса находится адрес ждущего стека или очереди стеков, процесс, открывающий семафор, ставит их в очередь готовых к выполнению процессов. Аналогичным образом осуществляется синхронизация работы процесса при обращении его к данным, находящимся во внешней памяти. Формируя задание процессору ввода-вывода, процессор закрывает семафор. Открытие этого семафора производится после выполнения соответствующего задания процессором ввода-вывода. Перед обращением к заказанным данным процесс обращается к семафору, и, если соответствующий разряд его не является нулем, "виснет" на этом семафоре. В очередь готовых процессов его переводят только после выполнения соответствующего задания процессором ввода-вывода в момент открытия семафора.

Таким образом, в пассивном состоянии процесс может находиться ввиду следующих трех состояний:

а) ждет окончания обращения другого процесса к общим данным;
б) ждет обмена необходимыми данными с оперативной и внешней памятью;

в) ждет освобождения процессора (очередь готовых процессов).

Как только какой-либо из процессоров освобождается от работы (процесс, над которым он работал, переходит в пассивный или заканчивается), он обращается в очередь готовых процессов и начинает выполнять процесс, стоящий в голове очереди готовых процессов. Необходимо отметить, что большинство процедур операционных систем выполняется на стеках тех процессов, которые их вызывали. Наряду с этим существуют самостоятельные процессы операционной системы, стеки которых подчиняются общей дисциплине. В зависимости от приоритета процесса его место в очереди готовых процессов может быть изменено.

3. Принципы организации сверхоперативной памяти

Проектируя сверхоперативную память в многопроцессорной системе, в целях эффективного использования статического ресурса этой памяти появляется желание обобщить ее для всех процессоров. Однако учитывая тот факт, что производительность каждого процессора во многом определяется быстродействием сверхоперативной памяти, становится ясно, что узким местом всей системы будет динамический ресурс этой памяти.

Есть еще и немаловажный фактор, препятствующий объединению сверхоперативной памяти (СОЗУ), - это резкое увеличение времени операции обращения к СОЗУ за счет больших потерь в линиях связи от процессоров к СОЗУ. Длина этих линий связи будет пропорциональна площади, занимаемой всеми процессорами. Поэтому, учитывая, что в будущем можно ожидать сокращения времени выполнения операции в СОЗУ лишь до 5 нс, объединение СОЗУ является бесперспективным.

Структура построения СОЗУ для каждого процессора в многопроцессорных системах по типу "кэш" вызывает определенные трудности. Дело в том, что при обращении к общим данным хотя бы двух процессоров возможна ситуация, когда в СОЗУ одного из процессоров будут находится старые данные, а в ОЗУ обновленные. Избежать этого можно только в том случае, если при каждой записи в ОЗУ записывающий процессор будет стирать записываемые данные в СОЗУ других процессоров. Последнее неминуемо приведет к значительному сокращению динамического ресурса СОЗУ процессоров.

В МВК "Эльбрус" принята распределенная по процессорам сверхоперативная память, организованная по функциональному принципу. Так, по функциональному назначению сверхоперативная память разбита на пять частей (Рис.6):

1. *Буфер быстрых, регистров.* Данные верхушки стека, находящиеся в обработке в процессоре - виртуальные быстрые регистры, выполняющие дисциплину стека.

2. *Буфер локальных данных.* Локальные данные процесса I - непрерывный участок памяти стека от указателя стека до данных процедуры A. Ввиду непрерывности отображаемого участка памяти возможна прямая адресация к этой сверхоперативной памяти.

3. *Буфер глобальных данных.* Общие данные процесса I и процесса II отображаются ассоциативной памятью.

4. *Буфер команд* процесса I и процесса II сокращает число обращений к ОЗУ за командами при повторном вызове процедур (в частности, для убыстрения выборки команд в цикле). Выполнен по ассоциативному принципу.

5. *Буферная память* для аппаратной опережающей подкачки данных массивов с целью максимального ускорения работы с ними. Выполнена по ассоциативному принципу.

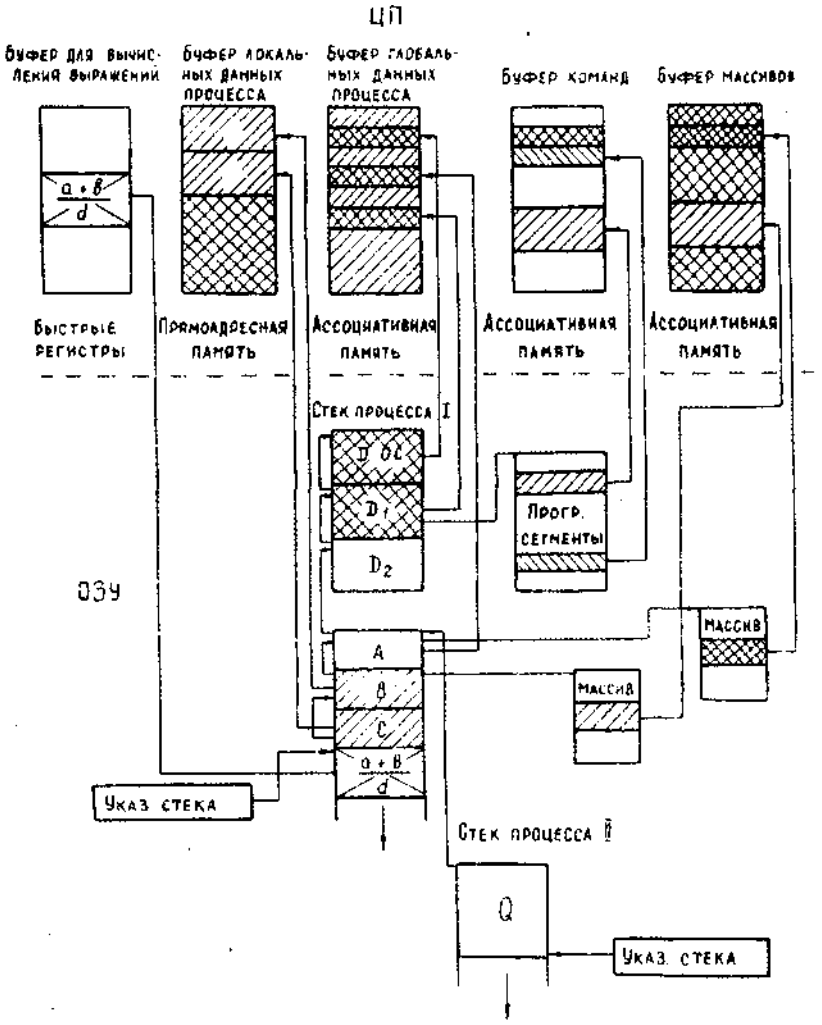


Рис.6. Организация сверхоперативной памяти процессора.

Рассматривая каждое из этих СОЗУ с точки зрения описанной выше конфликтной ситуации, возникающей в каждом процессе в случае построения СОЗУ по типу кэш, можно заметить, что память быстрых регистров, локальных данных и буфер команд свободны от этого дефекта. Первая и вторая ввиду локальности данных, третья вследствие неизменности кодов команд в процессе их выполнения. Конфликтная ситуация может проявиться в ассоциативной памяти глобальных данных и буфере массивов. При обращении к общим данным буфер массивов может быть безболезненно стерт, в то время как полный сброс из буфера глобальных данных замедлит вычислительный процесс. Поэтому при записи информации в эти регистры одновременно с данными в этот регистр пишется время их записи ($t_{зп}$). При открытии семафора в его разряды адреса пишется время открытия (Рис.5). По этим временам однозначно решается вопрос о том можно ли воспользоваться данными, находящимися в регистре или необходимо вызвать данное из ОЗУ. Если $t_{зп} < t_{откр}$, то данное

вызывается из ОЗУ, а данный регистр буфера глобальных данных обнуляется. В противном случае ($t_{ЗП} > t_{0ТКР}$) используются данные регистра.

Принятый в МВК "Эльбрус" принцип построения СОЗУ значительно экономичнее буферной памяти, построенной по типу кэш, так как позволяет:

а) сократить общее количество буферных ячеек;

б) многие из них сделать прямоадресуемыми (без ассоциативности), тем самым уменьшив оборудование и сократив время обращения;

в) распараллелить работу СОЗУ, что значительно повысит эффективность его использования;

г) исключить конфликтные ситуации, присущие СОЗУ, построенным по традиционным принципам.

4. Особенности использования оперативной памяти

Эффективность использования оперативной памяти, а соответственно и внешней памяти, во многом зависит от основных принципов ее распределения, заложенных в аппаратуре и операционной системе, самих системных программ и целого ряда других факторов, вызванных конструктивными соображениями организации работы вычислительного комплекса. Прежде всего отметим, что аппаратная реализация рекурсии позволяет обходиться одним комплектом программ в оперативной памяти. Это обстоятельство особенно важно для многопроцессорных систем. Если отсутствие рекурсии в однопроцессорных системах несколько снижало эффективность использования памяти при работе стандартных и других системных программ, то в многопроцессорной системе отсутствие рекурсии привело бы к многократному копированию в памяти большинства системных программ, отдельных программ прикладных задач и общих данных параллельно выполняемых задач.

Эффективность использования памяти во многом определяется методом ее распределения. Очевидно, тот метод будет эффективнее работать, который в большей мере позволит выполнить следующие требования:

а) должен быть вызван квант данных, вероятность использования которых в последующий момент наибольшая;

б) требуемый квант данных должен быть вызван в самый последний момент, и должна иметься возможность освобождения памяти в любой момент после ее использования;

в) должна иметься возможность перераспределять данные внутри памяти или сбрасывать их во внешнюю память с последующим вызовом;

г) должна быть предусмотрена возможность размещать большие массивы, представляющие единую линейную последовательность данных, в различные участки памяти отдельными квантами.

Так, принятый в операционной системе (ОС) IBM-360 метод распределения памяти не удовлетворяет ни одному из этих пунктов и представляет собой метод статического распределения памяти по задачам. Пункты требований а), б) и г) не удовлетворяются из-за принятого в ОС IBM-360 метода распределения памяти, а пункт в) ввиду того, что в поле памяти работы программы имеется информация о физических адресах данных и команд, исключающая возможность работы программы на другом физическом месте памяти.

По этим причинам в семействе IBM-370 в качестве средства, повышающего эффективность использования памяти, применена математическая память с подкачкой данных страницами фиксированного размера.

Использование математической памяти позволило в какой-то мере удовлетворить требованиям только пунктов в) и г). Принципиальным является то обстоя-

тельство, что введение математической памяти не решает задачи удовлетворения требований пунктов а) и б), существенно определяющих эффективность распределения оперативной памяти. Дело в том, что при выборе объема страницы математической памяти приходится учитывать то обстоятельство, что чем меньше страница, тем больший объем памяти должен быть отведен в операционной системе на их таблицу. С другой стороны, поскольку страница определяет квант данных, которыми происходит обмен между оперативной и внешней памятями, увеличение объема страницы резко снижает эффективность использования памяти. Для подтверждения этого приведем графики, характеризующие использование оперативной памяти в зависимости от величины страницы (Рис.7).

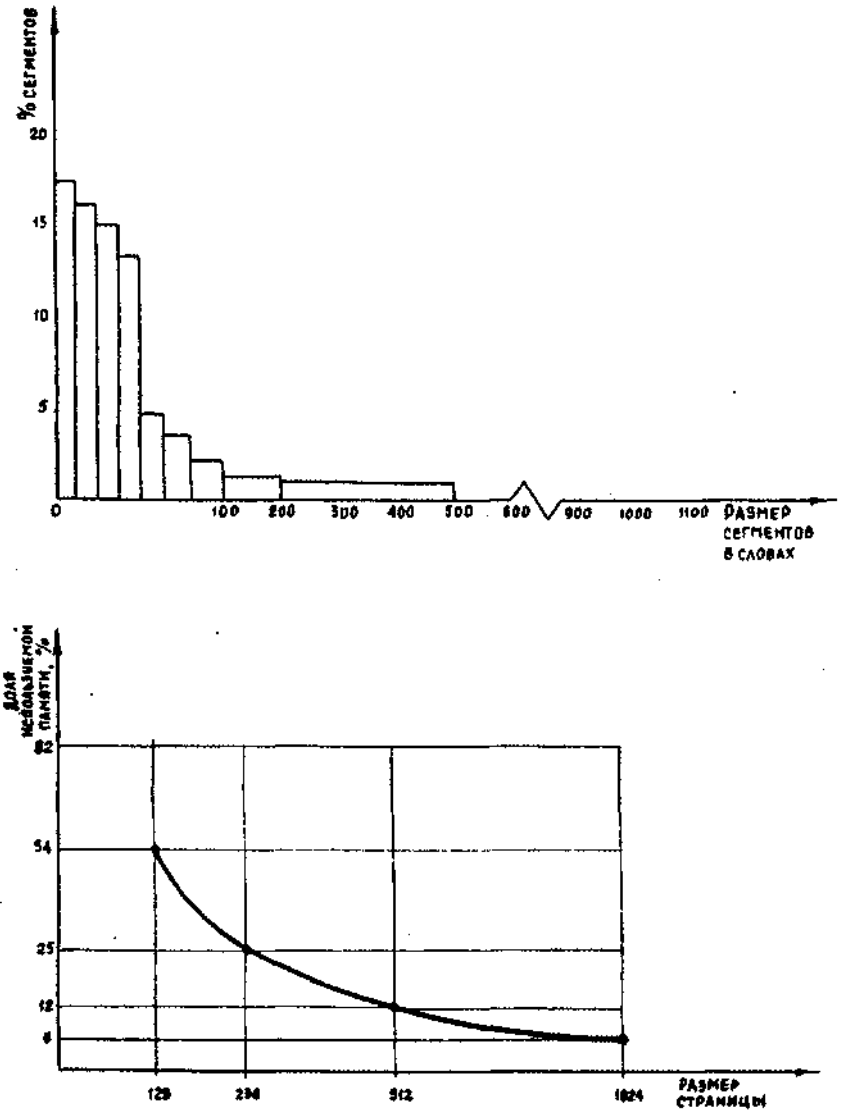


Рис.7. Эффективность использования ОЗУ в зависимости от размера страницы.

Как уже упоминалось, при расположении информации в физической памяти ее объем, выделяемый под данные, определяется только требованиями дескриптора, ввиду чего упаковка физической памяти от недоиспользования математической не страдает.

Таким образом, распределение физической памяти в МВК "Эльбрус" наиболее полно отвечает вышеизложенным требованиям, так как:

1) единственным источником информации о том, какие наиболее вероятно потребуются данные программе для работы и в каком объеме, является только сам алгоритм. При составлении программы трансляторы на оснований алгоритма объединяют однородные данные, описывая их одним дескриптором, поэтому принцип подкачки данных квантами, определяемыми дескрипторами, наиболее точно удовлетворяет требованиям пункта а);

2) описанный выше принцип организации вычислений с использованием стека наиболее полно удовлетворяет требованию пункта б).

В самом деле:

- память под выполнение процедуры (коды команд, локальные данные) выделяется только в момент непосредственного перехода к ее выполнению;
- память под промежуточные вычисления и сопутствующие им массивы данных выделяется в процессе выполнения программы;
- сразу же после завершения выполнения этой процедуры память освобождается - стек и ненужные массивы данных, описанные в нем, ликвидируются;

3) отказ от переиспользования математической памяти позволяет бесконфликтно сбрасывать данные на внешний носитель, восстанавливать их, производить необходимое перераспределение с целью повышения эффективности использования физической памяти;

4) "нарезка" математической памяти на страницы исключает необходимость отыскания больших свободных кусков физической памяти при загрузке в нее больших массивов данных.

Принятый в МВК "Эльбрус" принцип распределения памяти посредством сегментов произвольной длины приводит к появлению в памяти произвольных свободных мест различных размеров, заполнение которых требует определенного усложнения алгоритма работы операционной системы. Кроме того, введение дескрипторов и математической памяти безусловно усложняет доступ процессора к данным. Однако введение сверхоперативной памяти, работающей непосредственно в математических адресах, аппаратная реализация ассоциативных таблиц страниц, аппаратно-программный поиск по таблицам страниц и целый ряд других аппаратных средств существенно нивелирует эти недостатки.

Методы распределения физической памяти, принятые в МВК "Эльбрус", неразрывно связаны с принципами организации вычислительного процесса и защиты данных в системе. Эти методы базируются на аппаратной реализации устоявшихся приемов распределения памяти и, несомненно, дадут существенный эффект в использовании как статического, так и динамического ресурса памяти, что чрезвычайно важно для построения многопроцессорного комплекса.

5. Принципы управления внешними устройствами и терминальным оборудованием, работающим через линии связи

Требования к устройствам управления внешней памятью и устройствам ввода-вывода (внешние устройства) имеют много общего с требованиями к устройствам управления, обслуживающим терминальные устройства через линии передачи данных. Так, например, оба устройства управления должны работать в строго регламентированном реальном времени и обладать высоким

быстродействием на момент пиковой нагрузки, которая во времени чрезвычайно неравномерна. Кроме того, оба устройства решают автономную задачу, алгоритм которой практически не зависит от основных задач, решаемых центральной системой, ввиду чего может быть использована Локальная память. Круг задач, решаемых этими устройствами, ограничен той аппаратурой, которой они управляют.

Существенное их различие вытекает из количества типов объектов и их возможных комбинаций подключения. Так, номенклатура внешних устройств строго регламентирована, число возможных способов их коммутации сравнительно невелико.

Количество типов объектов, которыми приходится управлять, для обеспечения работы терминалов чрезвычайно велико - сюда входят модемы, линии связи, различные устройства ввода-вывода, микропроцессоры, вычислительные машины.

Реализация всех функций и того, и другого устройства может быть выполнена одним из центральных процессоров системы, однако, учитывая специфичность обработки информации (только логическая обработка малоразрядных данных), использование оборудования центральных процессоров будет неэффективным. Поэтому для МВК "Эльбрус" принято решение о включении в систему специальных процессоров: процессора ввода-вывода (ПВВ), управляющего внешними устройствами, и процессора приема-передачи данных (ППД), управляющего терминальным оборудованием через линии связи.

Остается нерешенным традиционный вопрос, на какой оперативной памяти целесообразнее работать этим спецпроцессорам: общей или локальной. Ответ на этот вопрос можно решить, рассмотрев эффективность использования статического и динамического ресурса памяти в том и другом случае.

С точки зрения более эффективного использования статического ресурса, имея в виду возможность перераспределения памяти, работа спецпроцессоров на общей памяти безусловно рациональнее. Анализ этих вариантов, с точки зрения рационального использования динамического ресурса, дает противоположное заключение. Действительно, всякое лишнее обращение к общей памяти спецпроцессора уменьшает ее динамический ресурс, в то время как при локальной памяти этого не происходит. Исходя из этого, можно сформулировать общие принципы использования оперативной памяти спецпроцессорами.

Те данные, к которым спецпроцессор в процессе своей работы обращается не многократно, целесообразнее помещать в общую память; данным же, требующим многократного обращения, должна быть выделена специальная память. Так, например, программа и память буфера сообщения ППД в процессе работы с терминалами требуют многократного обращения к ним, в то время как сформированное сообщение требует только одной записи в общую память. Поэтому под выполняемую в ППД программу и буфер должна быть выделена специальная память, а для хранения сообщений может быть использована общая и так далее.

Учитывая вышеизложенное, рассмотрим основные принципы работы ПВВ и ППД.

Как уже упоминалось, ПВВ управляет строго регламентированными типами объектов, число вариантов коммутаций которых невелико. Это дает возможность все функции Процессора и коммутацию объектов выполнить аппаратным способом. В этом случае нет необходимости в локальной памяти для команд, а малоразрядная обработка данных требует небольшой сверхоперативной локальной памяти.

В современных ЭВМ, как правило, функции диспетчеризации обращения к внешним объектам выполняются программным способом операционной систе-

мой. Алгоритм работы "диспетчера" операционной системы в настоящее время можно считать устоявшимся, и он сводится в основном к обработке очереди заявок задач по обращению к внешним объектам. Эта работа требует обработки большого количества прерываний от внешних объектов, связанных с индикацией их состояния, и выработки соответствующих управляющих команд в виде реакции на эти прерывания.

В МВК "Эльбрус" принято решение максимально разгрузить работу центральных процессоров от прерываний внешними устройствами посредством передачи функций диспетчеризации процессору ввода-вывода. Детальный анализ этих функций показал возможность простой аппаратной их реализации Работа операционной системы, выполняемая центральным процессором, заканчивается составлением заявки на обращение к внешнему устройству, в которой указывается адрес информации внутри объекта, адрес ОЗУ, куда необходимо поместить (или откуда взять) информацию и ее объем. Такая заявка центральным процессором ставится определенным образом в очередь к дескриптору, описывающему данное устройство. Этот дескриптор находится в таблице по индексу, соответствующему порядковому номеру, присвоенному этому устройству. Таблица устройств храниться в оперативной памяти (Рис.8).

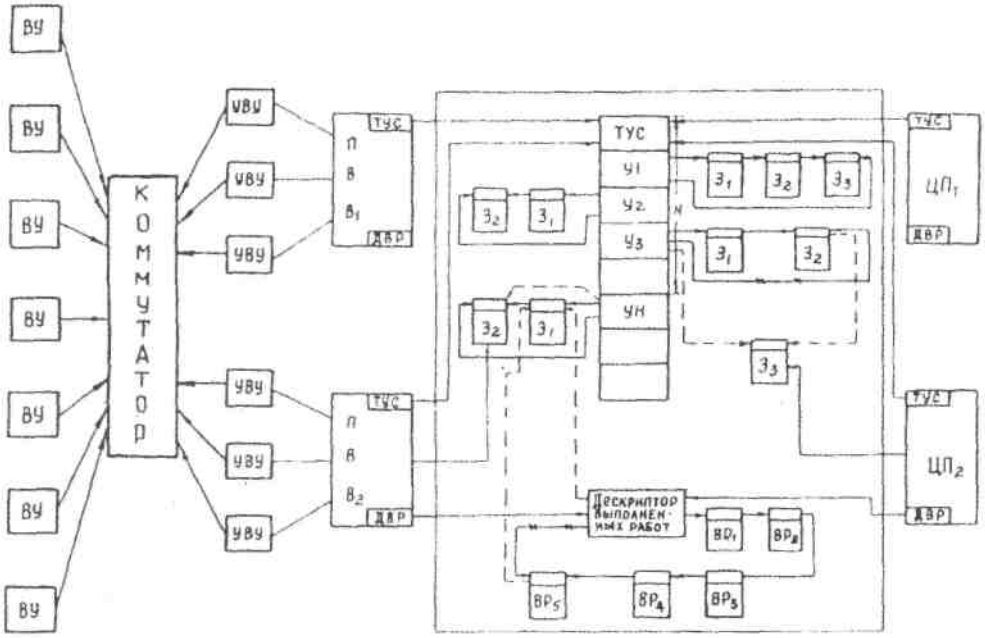


Рис.8. Принцип управления внешними устройствами.

Как только внешнее устройство заканчивает выполнение очередного задания, процессор ввода-вывода делает соответствующие отметки в заявке, выводит эту заявку из очереди к устройству и ставит ее в очередь выполненных работ, после чего ПВВ берет следующую по очереди заявку и организует ее выполнение на устройстве. Список выполненных заявок обрабатывается операционной системой на центральных процессорах. В случае, если заявка выполнена полностью без замечаний, ОС открывает соответствующий семафор и производит связанные с этим необходимые операции. В том же случае, если ПВВ указал на дефекты при вы-

полнении задания (сработал аппаратный контроль и так далее), ОС принимает решение о необходимости повторного выполнения задания. Как видно из Рис.8, одно и то же внешнее устройство может работать через любое из четырех ПВВ и любое управление внешним устройством УВУ. Выбор пути коммутации определяется аппаратно следующим образом.

Первоначально, когда к устройству N поступает первая заявка, ее обрабатывает любой свободный в данный момент ПВВ. Свободным ПВВ считается тогда, когда он выполняет заявку в части изменения режимов работы устройств. Работа ПВВ с устройствами в мультиплексном режиме не препятствует первоначальной обработке заявок. Последующие заявки к устройству N будут обрабатываться на том же ПВВ, так как сигнал о завершении работы с внешнего устройства приходит на тот ПВВ, который это устройство запустил. Устройство УВУ и соответствующий канал коммутации выбираются по мере его освобождения. Отдельные внешние устройства, такие как перфокарты, печать, магнитная лента, оказывающие меньшее влияние на живучесть системы, не имеют специальных коммутаторов и жестко закреплены за каждым ПВВ.

Принятая в МВК "Эльбрус" система управления, аппаратно реализующая все функции управления внешними объектами, включая функции диспетчеризации, резервирования и выбора исправного канала, позволяет существенно сэкономить ресурс оперативной памяти и избавить центральные процессоры от большого числа прерываний.

В отличие от ПВВ процессору приема-передачи данных (ППД) необходимо управлять большим количеством разнотипных объектов, способы коммутации которых практически безграничны. Причем, если введение новых типов устройств связано с их разработкой и освоением в производстве, что происходит не так часто, то изменение схем коммутации различных объектов даже в работающей системе может происходить ежедневно. Все это приводит к тому, что аппаратная реализация всех этих функций с помощью жестко закоммутированных программ процессора накладывает серьезные ограничения на работу системы в разветвленной сети терминальных устройств, работающих через линии передачи данных.

В МВК "Эльбрус" принят программный способ адаптации ППД к тем объектам, которыми он управляет, причем приняты все меры к тому, чтобы при изменении коммутации объектов не требовалось исправлять программу, а в случае введения нового устройства изменение программы ограничивалось бы добавлением программного модуля, описывающего вводимый в систему новый объект.

Достижение этих требований привело к следующим принципам организации математического обеспечения ППД. Существует справочник программных модулей, описывающий объекты, работающие в системе (включая модемы, линии передачи данных и так далее). Имеется таблица соединений объектов на данный момент.

При обращении к одному из терминальных устройств, на основании таблицы соединений, операционная система формирует из программных модулей объектов программу, обеспечивающую взаимодействие с требуемым терминальным устройством, учитывая его подсоединение в системе. Таким образом, при изменении коммутации объектов необходимо с одного из терминалов внести изменения в таблицу соединения объектов.

Ввиду того, что формирование рабочей программы и обращение к справочникам происходит однократно, а при работе с терминалом сформированная рабочая программа используется многократно, все справочники и таблицы расположены в общей памяти системы, а для хранения сформированных рабочих программ и их

операндов процессору приема-передачи данных выделена локальная память. В традиционных вычислительных комплексах аналогичные процессоры не имеют своего гибкого математического обеспечения, поэтому функции адаптации выполняет операционная система комплекса. Это, как правило, связано с полной или частичной ее генерацией.

Приведенный сравнительный анализ структуры многопроцессорного вычислительного комплекса "Эльбрус" позволяет сделать следующие выводы:

1. Многопроцессорная структура значительно увеличивает рентабельность и эффективность использования самого дорогостоящего оборудования - памяти.
2. Многопроцессорная структура является наиболее перспективной в части построения комплексов предельно большой производительности.
3. Принятый модульный принцип построения позволяет наиболее эффективно обеспечивать заданную структурную надежность комплекса и адаптацию его к решаемым задачам.
4. Требования обеспечения удобства и эффективности программирования, решения проблемы интегрирования математического обеспечения и развития языков высокого уровня не входят в противоречие с требованиями эффективного использования оборудования, обеспечения повышенной надежности и удобства эксплуатации. Эти требования имеют единую структурную основу.
5. Построение многопроцессорных вычислительных комплексов требует коренного пересмотра организации работ всех традиционных устройств и структуры комплекса в целом.

Неформальное описание системы команд

В.С.Бурцев, В.П.Торчигин

1. Организация памяти

1.1. Общие сведения о средствах адресации системы

Вычислительный комплекс «Эльбрус» предполагает сегментированное разбиение памяти по следующим функциональным областям:

1. Области памяти, отводимые для хранения кодов программ - программные сегменты.

2. Области памяти для хранения локальных данных и результатов промежуточных вычислений - сегменты стеков.

3. Области памяти для хранения локальных данных и массивов - сегменты данных.

Для обращения к величинам МВК располагает различными системами адресаций. Одни из них недоступны пользователю и используются непосредственно вычислительными средствами для оптимального размещения величин в физической памяти в процессе вычислений. Другие являются средствами адресации пользователя.

К недоступным пользователю относятся следующие системы адресации:

1. Адресация по физическим адресам, используемая аппаратными средствами для непосредственного обращения за словом в реальную память по физическому адресу.

2. Адресация в математическую память, адресное пространство которой составляет 2^{32} слов, в то время как максимальный объем реальной памяти не может превосходить 2^{19} слов. Таким образом, каждой задаче может быть выделена математическая память (МП) объемом 2^{32} слов.

МП используется аппаратно-программными средствами операционной системы с целью распределения реальной памяти и ее защиты. По требованию пользователя операционная система отводит ему необходимые области в математической памяти в виде сегментов, описываемых дескрипторами, и автоматически поставляет эти дескрипторы пользователю в процессе выполнения его программы.

В распоряжении пользователя имеются следующие средства адресации к величинам:

1. Адресация внутрь сегмента данных при помощи индексации. Непосредственно к программным сегментам пользователь адресоваться не может. В стековых сегментах пользователь может адресоваться только к определенным данным. Эти данные должны находиться в контексте выполняемой в это время процедуры. Обращение к этим данным происходит по адресным парам.

2. Для обращения за величиной в сегмент данных пользователь должен указать имя сегмента (его дескриптор) и величину смещения.

3. В вычислительном комплексе аппаратными средствами реализована безадресная система команд, которая предполагает, что операнды находятся в верхушке стека. Поэтому, при вычислении выражений пользователь, располагая определенным образом операнды в верхушке стекового сегмента, может не указывать адрес операнда в командной последовательности. После выполнения операции операнды вычеркиваются, а результат операции помещается в верхушку стека. У пользователя имеется возможность поместить в верхушку стека (загрузить в стек) любые имеющиеся в его контексте данные, а также записать полученный результат под любым именем, имеющемся в его контексте (записать из верхушки стека).

4. Адресация посредством адресных пар, которая фактически соответствует именам в языках высокого уровня. При помощи этой адресации пользователь может обращаться как к локальным данным выполняемой процедуры, расположенным в стеке, так и к глобальным данным, расположенным в других процедурах, доступных для данной.

5. Адресация к программным сегментам происходит посредством специального счетчика команд. Этот вид адресации предполагает обращение только в программные сегменты за вызовом кодов команд. Программные сегменты имеют единый байтовый формат величин, поэтому обращение к ним происходит по счетчику команд, код которого, как правило, меняется в зависимости от размера вызываемой команды. Непосредственно к программным сегментам пользователь обращаться не может. К программным сегментам может обращаться только аппаратура. Она индексирует программный сегмент индексом, который находится в счетчике команд. Затем счетчик команд увеличивается на количество байтов, равное длине (в байтах) выполняемой команды.

6. Для вызова необходимой информации из архива в распоряжение пользователя предоставляется алфавитно-цифровая система адресации, реализованная аппаратно-программным способом. Этот способ предполагает обращение за информацией, находящейся во внешней памяти. Он описан в разделе операционной системы. Обращение к архивной информации производится всегда путем явного вызова соответствующей системной программы.

Отметим, что любое обращение фактически производится через дескриптор, который содержит информацию о размере описываемого сегмента и о допустимом режиме работы с ним. Таким образом автоматически обеспечивается защита сегментов, имея в виду, что защита самих дескрипторов и других управляющих величин осуществляется аппаратурой с использованием специальной информации (теги), сопровождающей все слова памяти.

Обращение к величинам через дескрипторы несколько увеличивает время обращения к оперативной памяти, поэтому в МВК приняты следующие основные меры по сокращению этого времени.

1. Первые 16 ячеек верхушки стека выполнены на быстрых регистрах.

2. Имеются 32 регистра ассоциативного поиска слов по их математическому адресу.

3. Для 32 сегментов (размером менее 1024.слов) реализован аппаратный ассоциативный поиск физического адреса слова по его математическому адресу.

4. Все управляющие базовые регистры, включая 32 базовых регистра дескрипторов, описывающих локальные данные процедур в стеке, выполнены на быстрых регистрах.

5. Имеется буфер команд, выполненный на быстрых регистрах размером 8 слов.

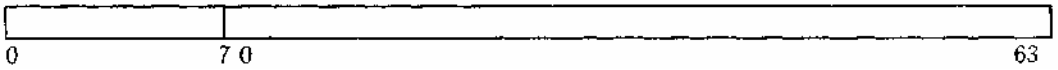
1.2. Структура математической памяти

Как уже указывалось, математическая память описывается 38-разрядным математическим адресом при условии указания размещения величины с точностью до одного бита (32-разрядным адресом при указании адреса машинного слова).

Основной единицей информации в этой памяти является величина. Величина - логическая единица информации, к которой можно адресоваться с помощью математического адреса. С другой стороны, величина - это единица информации, над которой производятся операции при выполнении команд программ.

Величины могут иметь следующие форматы: 1-разрядный, 4-разрядный, 8-разрядный, 16-разрядный, 32-разрядный, 64-разрядный и 128-разрядный.

Величины определенным образом упаковываются в физическую память. Единицей адресации физической памяти является слово, имеющее один формат 72 разряда. Слово состоит из двух частей: 64-разрядной информационной части и 8-разрядной управляющей (Рис.1).



управляющая часть информационная часть

(тег) Рис.1. Направление увеличения математических адресов.

Величины упаковываются в информационную часть слова по следующим правилам:

а) В одном слове могут находиться величины только одинакового типа и формата за одним исключением: целое и вещественное формата 32 разряда могут быть упакованы в одном слове в любом порядке.

б) Одна величина не может быть упакована так, чтобы одна ее часть находилась в одном слове, а другая часть - в другом слове. Другими словами, величина 2^n разрядов (n= 0,2,3,4,5,6) должна иметь математический адрес, в котором n младших разрядов равны нулю. Математический адрес 128-разрядных величин (n=7) может начинаться с любого номера слова.

в) В управляющих разрядах слова хранится информация о типе и формате величин, находящихся в слове (Рис.1).

Математическая память разбита на страницы. Страницы могут быть различных размеров. Размеры страниц кратны 16 словам. Блок из 16 слов, расположенный в математической памяти так, что адрес начала его в 10 младших разрядах 38-разрядного адреса имеет нули, в дальнейшем для краткости будем называть строкой. Максимальный размер страницы равен 1024 словам или 64 строкам. Таким образом начало страницы любого размера имеет математический адрес, младшие 16 разрядов которого равны нулю.

Начало любого сегмента в математической памяти совпадает с началом некоторой страницы. Для размещения сегмента используется страница минимально возможного размера. Сегменты размером более 1024 слов размещаются в математической памяти на смежных страницах, при этом все страницы, кроме последней, должны иметь максимальный размер.

Начало страницы, а следовательно и сегмента, в физической памяти может размещаться в любой физической строке, где физический адрес начала может быть любым физическим адресом (младший разряд физического адреса определяет слово). Страница располагается в смежных ячейках физической памяти.

Поэтому сегмент размером более 1024 слов может присутствовать не полностью в физической памяти, а его страницы могут находиться в различных областях физической памяти (Рис.2).

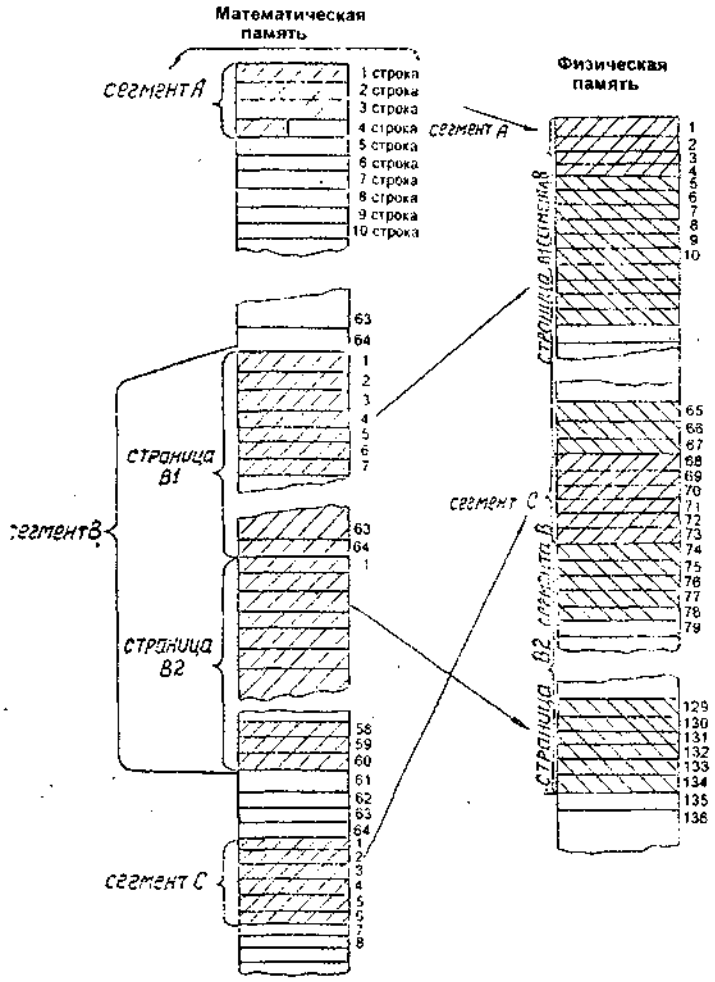


Рис.2.

1.3. Типы и форматы величин

Как уже указывалось, каждое слово имеет тег, описывающий характер информации, хранящейся в слове. В одном слове может храниться либо одна величина, либо несколько величин одинакового типа и формата, либо половина величины формата 128. Кроме того, имеются величины, в которых хранится управляющая информация, необходимая для внутренних нужд процессора. Такие величины в дальнейшем будем называть управляющими словами. Пользователь не имеет прямых средств для изменения информации в управляющих словах. Управляющие

слова используются устройством управления для автоматической корректировки базовых регистров при вызове процедур.

По функциональным особенностям все величины можно разбить на 3 класса: класс значений, класс адресов и управляющие слова.

а. Класс значений имеет следующие типы величин:

1. целое форматов 16, 32 и 64
2. вещественное форматов 32, 64 и 128
3. пусто форматов 32, 64
4. бит формата 1
5. цифра формата 4
6. байт формата 8
7. набор формата 64
8. индексное слово формата 64
9. интервал формата 64
10. дескриптор формата 64
11. метка go to формата 64
12. дополнительные типы формата 64

б. Класс адресов имеет следующие типы величин:

1. имя
2. косвенное слово
3. метка процедуры

в. Управляющие слова имеют два типа:

1. маркер стека (МКС)
2. управляющее слово возврата (УСВ)

Ниже приводится описание перечисленных типов величин и управляющих слов.

а. Типы класса значений:

1. Целое

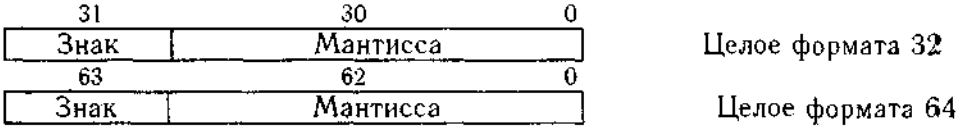


Рис.3. Представление целых чисел.

Значения типа целое имеют 2 формата: 32 разряда и 64 разряда (Рис.3).. Эти значения представляют собой целые числа, которые записаны в прямом коде со знаком.

2.Вещественное

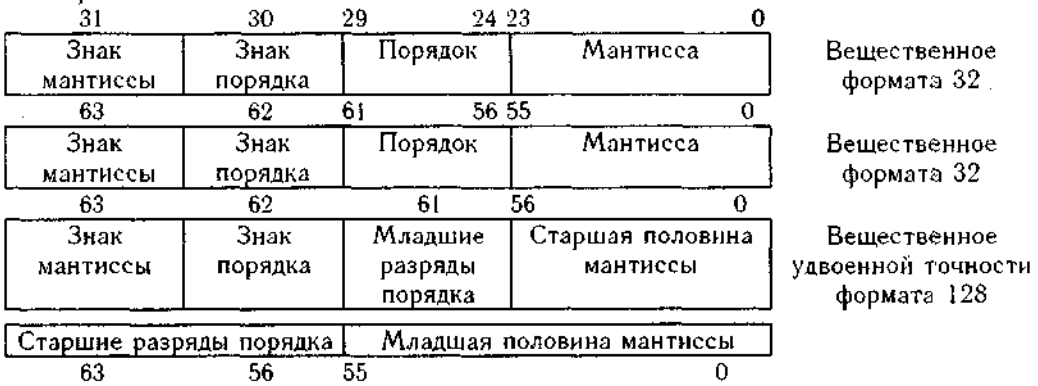


Рис.4. Представление вещественных чисел.

Значения типа вещественное имеют 3 формата: 32 разряда, 64 разряда и 128 разрядов (Рис.4). Вещественное число состоит из мантиссы (в прямом коде) со знаком и порядка (в прямом коде) со знаком. Вещественное число равно $M*16^{П-N}$, где М и П-целые числа, записанные в разрядах мантиссы (М) и порядка (П), N - количество шестнадцатиричных цифр в мантиссе.

3.Пусто

Значения типа пусто имеют 2 формата: 32 разряда и 64 разряда (Рис.5). Эти значения означают отсутствие информации (например, неинициализированные данные). При попытке считать значение типа пусто вырабатывается прерывание. Значение типа пусто следует отличать от технической пустышки, которая используется аппаратурой при упаковке величин в стеке и недоступна программисту.

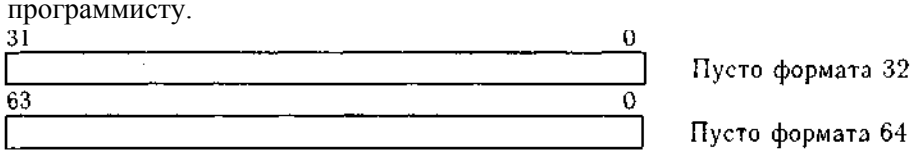


Рис.5. Форматы элементов памяти типа пусто.

4. Бит

Значения типа бит имеют форматы в 1 разряд и обычно представляют значения логических величин (истина или ложь). Значения типа бит широко используются в логических частях программы.

5. Цифра

Значение типа цифра имеет формат в 4 разряда и обычно представляет десятичную цифру, но вообще может представлять любой 4-разрядный символ. Цифровые массивы используются для представления десятичных чисел.

6. *Байт*

Значение типа байт имеет формат в 8 разрядов и обычно представляет код алфавитно-цифрового символа. Байтовые массивы используются для представления как оттранслированных программ, так и программ на исходном языке. Кроме того, байтовые массивы широко используются для хранения алфавитно-цифровой информации.

7. Набор

Значение типа набор имеет формат в 64 разряда. Значение типа набор представляет собой совокупность одинаковых по формату элементов (так называемых элементов набора).

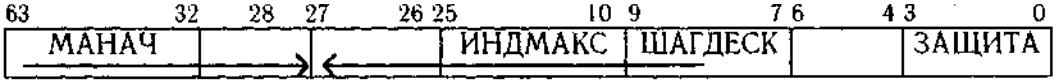
Отметим, что в отличие от самого набора элементы набора не являются величинами, и непосредственное обращение к элементам набора недопустимо. Для обработки элементов набора в системе команд имеются специальные операции.

8. Дескриптор

Значение типа дескриптор (Рис.6) имеет формат в 64 разряда и вводится для описания области памяти. Как уже указывалось, пользователь получает дескрипторы от операционной системы и не имеет средств вводить дескрипторы, описывающие новые сегменты, минуя операционную систему. Однако пользователь имеет возможность из любого своего дескриптора, описывающего некоторую область О; получать дескрипторы, описывающие подобласти области О (см. тип величины "интервал").

В дескрипторе указывается:

МАНАЧ - математический адрес начала описываемой области
ИНДМАКС - максимальное значение индекса
ШАГДЕСК - величина, определяющая на сколько разрядов происходит перемещение в описываемой области памяти при изменении индекса на 1. ШАГДЕСК может быть равен 2^n разрядов (n = 0,1 ,...,7). Значение n записывается в 3-х разрядах.
ЗАЩИТА - информация, определяющая режим доступа к описываемой области (разрешение считывания, записи, исполнения, записи и считывания адресной информации)



МАНАЧ - математический адрес начала массива
63-32 с точностью до слова,
63-26 с точностью до бита

ИНДМАКС - максимальный индекс или размер.
Граница между полями МАНАЧ и ИНДМАКС определяется аппаратурой по содержимому поля ШАГДЕСК.
Например, для слов (ШАГДЕСК = 6) имеем:
МАНАЧ = 63-32
ИНДМАКС = 28-10
для битов (ШАГДЕСК = 0)
МАНАЧ = 63-26
ИНДМАКС 23-10

ШАГДЕСК - шаг дескриптора = 2^n бит (n = 0,1,2,...,7)

ЗАЩИТА - информация о режиме доступа

Рис.6. Структура дескриптора.

Максимальный размер области, описываемой дескриптором, равен 2^{19} слов (~ 0,5млн).

Описание дескриптором области памяти игнорирует тот факт, что эта область (как и всякая другая) разбита на слова. Ответственность за правильную адресацию к величинам в области, описываемой дескриптором, возложена на программиста.

Обращение к величинам в области, описанной дескриптором, производится путем индексации дескриптора. Существует 3 способа индексации дескриптора (индексация целым, индексация индексным словом, индексация интервалом (раздел 1.4), но в каждом из способов тем или иным образом задается целый положительный индекс И, с помощью которого происходит вычисление смещения (СМ), относительно начала области, описываемой дескриптором.

При индексации целым или индексным словом величина смещения СМ определяется следующей формулой:

СМ = ШАГДЕСК * И, где И - значение целого, либо значение содержимого поля ИНДТЕК в индексном слове

При индексации интервалом величина смещения СМ определяется формулой:

СМ = ШАГИНТ *ИНДНАЧ

Независимо от способа индексации величина смещения сравнивается с размером области Р, описываемой индексируемым дескриптором. Р определяется следующей формулой:

Р = ШАГДЕСК * (ИНДМАКС + 1)

Если величина смещения СМ больше размера области Р - прерывание.

При операциях с интервалом контролируется следующее условие: область, описываемая интервалом, должна находиться внутри области, описываемой индексируемым дескриптором. Если это условие не выполняется - прерывание.

9. Индексное слово

Значение типа "индексное слово" имеет формат в 64 разряда и применяется при индексации дескриптора. После индексации дескриптора текущим значением индекса (ИНДТЕК) автоматически происходит модификация индексного слова путем прибавления к текущему значению ИНДТЕК величины приращения А. Таким образом, при каждой последующей индексации дескриптора индексным словом текущий индекс ИНДТЕК изменяется на А (Рис.7).

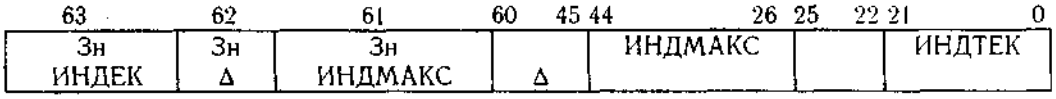


Рис.7. Структура индексного слова.*

Δ - величина приращения, ИНДМАКС - предел, ИНДТЕК (или ИНДНАЧ) - текущее значение.

Обычно индексное слово применяется в циклах для последовательной адресации к элементам массива. Поле массива ИНДМАКС в индексном слове используется для определения момента выхода из цикла во время выполнения операции КЦ - конец цикла. Если (ИНДМАКС - ИНДТЕК) А< 0, происходит выход из цикла. Использование в этом режиме индексного слова описано в разделе "Операции индексации" настоящей главы. Впоследствии в индексное слово добавлен признак НИ, указывающий на то, что в поле ИНДТЕК содержится немодифицированный индекс.

10. Интервал

Значение типа интервал имеет формат в 64 разряда и применяется для получения дескриптора подобласти из области, описываемой индексируемым дескриптором (операция "Взять подмассив" (ВП), Рис.8), а также для преобразования массивов в наборы и наоборот (раздел 1.4.2).

Как уже указывалось, пользователь получает готовые дескрипторы от операционной системы и не имеет средств вводить дескрипторы, описывающие новые сегменты, минуя обращения к операционной системе. Однако пользователь имеет возможность из любого своего дескриптора, описывающего некоторую область О, сформировать дескрипторы, описывающие подобласти области О. Для этого в верхушку стека должны быть помещены 2 величины: дескриптор, описывающий область О, и интервал, характеризующий необходимую подобласть О*.

---

* В процессе разработки и отладки комплекса поразрядное разбиение различных слов претерпели изменения, которые не сказались на принципах организации системы команд и вычислительных процессов в целом. Эти изменения были направлены на оптимизацию по быстродействию тех или иных операций, на экономию аппаратных средств, а также изменение аппаратных затрат на создание оперативной памяти.

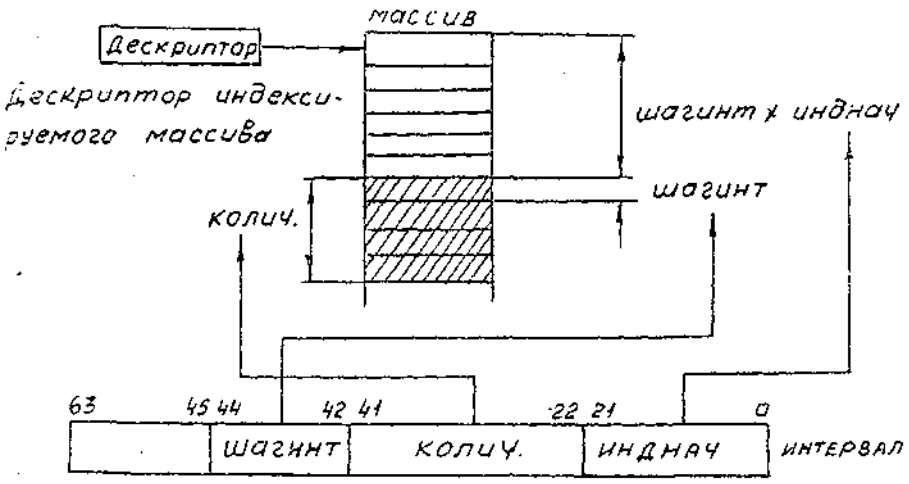


Рис.8. Выполнение операции ВП - взять подмассив.

Формирование дескриптора, описывающего подобласть О*, осуществляется однобайтовой операцией ВП - взять подмассив. В результате этой операции исходные величины вычеркиваются из стека и в верхушку стека помещается сформированный дескриптор Д*, имеющий следующие поля:

МАНАЧ* = МАНАЧ + ШАГИНТ*ИНДНАЧ
ШАГДЕСК* = ШАГИНТ ИНДМАКС*= КОЛИЧ,

где:

МАНАЧ	-	содержимое соответствующего поля исходного дескриптора,
ШАГИНТ, ИНДНАЧ, КОЛИЧ	-	содержимое соответствующих полей интервала,
МАНАЧ*, ШАГДЕСК*, ИНДМАКС*	-	содержимое соответствующих полей сформированного дескриптора.

Информация о режиме доступа переписывается в формируемый дескриптор в поле ЗАЩИТА* из поля ЗАЩИТА исходного дескриптора.

//. *Метка операций перехода (метка go to, Рис.9).*

Значение типа метка go to имеет формат в 64 разряда и применяется для передачи управления по метке с помощью операций переходов (раздел 1.10.3 и 1.10.4). В метке go to содержится информация о программного сегменте, в который передается управление (поле БАЗА) и адрес команды внутри программного сегмента (поле N^). Кроме того, метка содержит информацию о лексикографическом уровне (поле LL) и адресном пространстве процедуры, в которую передается управление (поле АДРЕС).

Существует 6 типов меток, отличающиеся тегами:

1) привилегированная метка,
2) метка процедуры,
3) метка процедуры с запретом записи адресной информации,
4) сквозная или техническая метка,
5) сквозная метка с запретом записи адресной информации,
6) метка go to (признаки сквозной метки и запрета записи адресной информации отсутствуют).

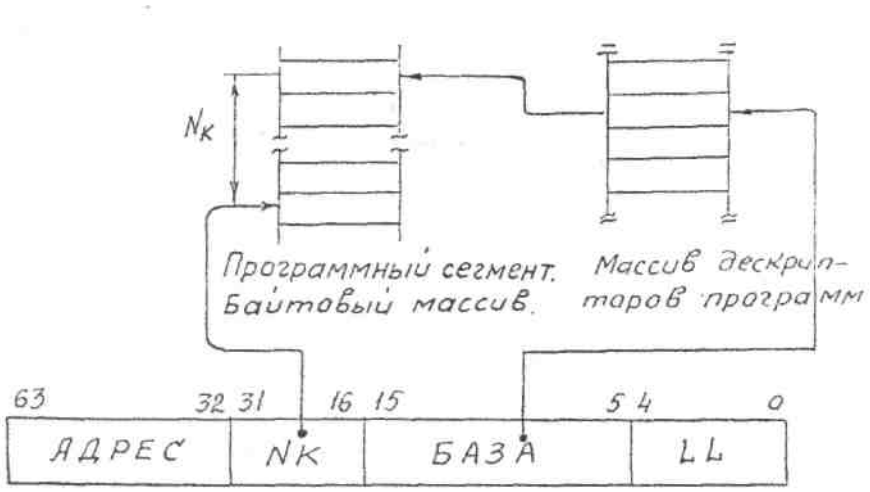


Рис.9. Структура метки.
АДРЕС - математический адрес начала области данных,
N_k - номер байта относительно программной базы,
БАЗА - информация об адресе программного сегмента,
LL - уровень тела процедуры, в которой описана данная метка.

Кроме метки go to введено понятие процедурная метка, в которых поле адрес указывает на МКС уровня LL-1. Различают 6 типов процедурных меток, отличающихся своими тегами:

1. техническая метка
2. техническая привилегированная метка
3. нормальная метка
4. нормальная метка с блокировкой записи адресной информации
5. привилегированная метка
6. привилегированная метка с блокировкой внешних прерываний.

б. Величины класса адресов:

/. Адресная пара.

В областях данных все адресные пары занимают по одному слову, то есть имеют формат в 64 разряда (Рис.10).

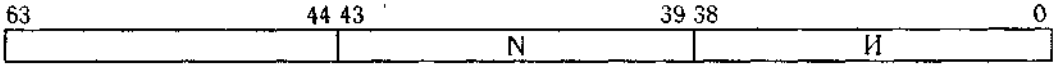


Рис.10.
N - номер лексикографического уровня, И - индекс.

Адресная пара может индексироваться. При этом в верхушке стека должны находиться адресная пара и информация об индексе. При индексации адресной пары (N, n) происходит индексация дескриптора, расположенного в базовом регистре с номером N. Выполнение этой операции описано в разделе "Операции индексации" настоящей главы.

2. Косвенное слово.

Величины типа косвенное слово имеют формат в 64 разряда (Рис.11). Любая величина может описываться косвенным словом, в котором указывается:

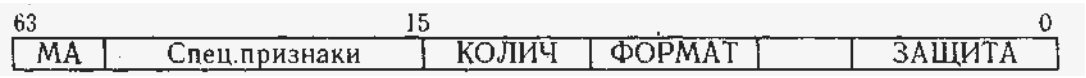


Рис. 11. Структура косвенного слова.
МА - математический адрес с точностью до бита, К - признак конца косвенности, У - признак указателя, В - признак векторного косвенного слова (специальные признаки), КОЛИЧ - количество элементов в векторном косвенном слове, ФОРМАТ - формат величины, ЗАЩИТА - информация о режиме доступа.

а) МА - величина в памяти, на которую указывает косвенное слово.

б) Формат величины - двоичный логарифм формата (записывается в трех разрядах поля ФОРМАТ).

в) Информация, определяющая режим доступа к величине (записывается в четырех разрядах поля ЗАЩИТА).

г) Косвенное слово может описывать массив величин одного и того же типа и формата, умещающийся в один набор (так называемое векторное косвенное слово). Информация о количестве элементов в наборе помещается в поле КОЛИЧ. Признак векторного косвенного слова помещается в разряд В.

д) Имеются 2 вида косвенных слов, различающихся признаком указателя У:

1) У = 0 - ссылка (сквозной указатель),

2) У = 1 - указатель.

В системе команд имеется специальная операция, которая преобразует вид ссылки в вид указатель. В дальнейшем термин косвенное слово будем применять и к ссылкам, и к указателям в тех случаях, когда различие между ними несущественно.

Различие между этими видами проявляется только в 2-х операциях ВОА и ВАОА (раздел 2.1.2). Введение признака указателя позволяет эффективно обрабатывать адресную информацию.

е) Признак конца косвенности КК свидетельствует о том, что данное косвенное слово является предпоследним в цепочке последовательных вызовов. Программист имеет возможность устанавливать признак КК по своему усмотрению.

Впоследствии структура косвенного слова претерпела значительные изменения: добавлен признак С, соответствующий тому, что поле адрес указывает на элемент в стеке; поле математического адреса перенесено в правую половину слова.

В конечном счете структура дескриптора, косвенного слова и имени полностью унифицированы. Они имеют различные теги. Для дескриптора дополнительно введен признак П. При индексации дескриптора с П=1 выдается прерывание.

3. Метка процедуры.

Величины типа метка процедуры имеют формат в 64 разряда. Метка процедуры служит для вызова по этой метке процедуры. Структура метки процедуры такая же, как и структура метки go to (Рис.9), но в поле LL записывается номер лексикографического уровня вызываемой по этой метке процедуры. Вводятся 2 вида меток, различающиеся своими тегами:

- обычная метка;
- сквозная метка.

В результате операций формирования меток получаются обычные метки, но в системе команд имеется специальная операция, которая преобразует обычные метки в сквозные. В дальнейшем термин метка процедуры будет применяться как к сквозным, так и к обычным меткам в тех случаях, когда различие между ними несущественно.

Различие между метками процедуры сквозной и обычной проявляется только в операциях загрузки стека (раздел 1.8).

Как обычные, так и сквозные метки имеют 2 варианта, различающиеся своими тегами. Первый вариант - процедура, вызываемая по метке, может записывать адресную информацию.

Второй вариант - процедура, вызываемая по метке, и все процедуры, вызываемые через эту процедуру, не могут записывать адресную информацию.

Изменение варианта метки может производиться только операционной системой.

Для нужд операционной системы введен еще один тип метки - привилегированная метка, имеющая определенный тег. По таким меткам операционная система может обращаться к процедурам, которые недоступны пользователям.

В дальнейшем, были реализованы следующие 6 типов процедурных меток:

1. *техническая метка (аналог сквозной)*
2. *техническая привилегированная метка*
3. *нормальная метка*
4. *нормальная метка с блокировкой записи адресной информации*
5. *привилегированная метка*
6. *привилегированная метка с блокировкой внешних прерываний.*

в. Управляющие слова:

/. *Маркер стека (МКС).*

МКС является управляющим словом формата 64 (Рис.12). По содержимому своих полей МКС похоже на дескриптор, так как оно описывает область в стеке ниже самого себя, отведенную для формальных и локальных параметров некоторой процедуры (поле РАЗМЕР).

Рис.12. Структура МКС.

АДРЕС - математический адрес предыдущего МКС по статической цепочке,
БЛ - признак блокировки записи адресной информации,
РЕЖ - признак привилегированной процедуры,
NIL - признак конца статической цепочки,
РАЗМЕР - размер области памяти в стеке, отведенной для процедуры,
LL - номер лексикографического уровня процедуры.

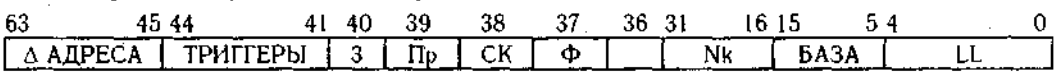
Кроме того, МКС содержит информацию о лексикографическом уровне этой процедуры (поле LL) и адресном пространстве этой процедуры (поле АДРЕС).

Признак NIL ставится в 1, если данное МКС является конечным в статической цепочке. Признак блокировки записи адресной информации БЛ ставится в 1, если МКС принадлежит процедуре, которой запрещена запись адресной информации. Признак РЕЖ ставится в 1, если запускаемая процедура недоступна пользователю.

В дальнейшем признак NIL был исключен. Добавлено 2-битовое поле ЗН, имеющее следующий смысл:

0	-	*выход из процедуры без значения*
1	-	*процедура должна оставлять в вершине стека значение (I слово)*
10	-	*процедура должна оставлять в вершине стека указатель (2 слова, например, дескриптор и индекс)*

2. Управляющее слово возврата (УСВ).



А АДРЕСА	-	*адрес данного УСВ - адрес МКС запустившей процедуры,*
ТРИГГЕРЫ	-	*информация о состоянии запустившей процедуры,*
З	-	*признак завершенности входа в процедуру,*
Пр	-	*признак, прерывания,*
СК	-	*признак того, что процедура запущена по сквозной метке,*
Ф	-	*признак процедуры-функции,*
Nk	-	*номер байта возврата относительной программной базы,*
БАЗА	-	*информация о базе программного сегмента,*
LL	-	*номер лексикографического уровня запустившей процедуры.*

Рис.13. Структура УСВ.

УСВ является управляющим словом формата в 64 разряде (Рис.13). УСВ располагается в стеке рядом с МКС, и вместе они образуют так называемую связующую информацию, позволяющую восстановить первоначальное состояние системы после возврата из процедуры. УСВ в некотором отношении является меткой возврата, так как оно содержит информацию об адресе возврата (в полях БАЗА и Nk). Кроме того, УСВ содержит информацию, необходимую для восстановления лексикографического уровня, окружения и режима процедуры, в которую производится возврат (в полях LL, Δ-адреса, ТРИГГЕРЫ).

Признак З указывает на то, что формирование связующей информации при входе в процедуру закончено. Признак прерывания указывает на то, что при выходе из процедуры должен вырабатываться сигнал прерывания.

Признак СК указывает на то, что процедура была запущена по сквозной метке.

Признак Ф указывает на то, что связующая информация относится к процедуре-функции, которая должна поставлять значение.

В дальнейшем признак Ф исключен. Вместо него в МКС введено поле ЗН. Добавлен признак ОВП - разрешения внешних прерываний в запустившей процедуре.

1.4. Адресация при помощи индексации

Перед выполнением операции индексации в верхушку стека должны быть загружены дескриптор или адресная пара и информация об индексе. При индексации адресной пары (N,И) происходит индексация дескриптора, размещенного в базовом регистре с номером N, причем в качестве индекса используется индекс, загруженный в стек, увеличенный на величину И индексируемой адресной пары. С учетом этого замечания дальнейшее описание относится к индексации и адресной пары, и дескриптора.

Информация об индексе может задаваться одним из следующих 3-х способов:

- целым числом или правым битовым набором, который автоматически преобразуется в целое;
- интервалом;
- индексным словом, либо непосредственным адресом индексного слова (адресная пара или косвенное слово).

1.4.1. Индексация целым числом

Индексация целым числом Ц (Рис.14) может выполняться следующими операциями:

ИНД - индексация;
ИНДЗ - индексация с загрузкой.

Результатом операции ИНД является косвенное слово в верхушке стека, содержащее МА элемента массива. Информация в поля ФОРМАТ и ЗАЩИТА результирующего косвенного слова переписывается из исходного дескриптора соответственно из полей ШАГДЕСК и ЗАЩИТА. Математический адрес элемента массива вычисляется по формуле:

МА = МАНАЧ + ШАГДЕСК x Ц.

Операция ИНДЗ эквивалентна последовательному выполнению операций ИНД и ВЗА.

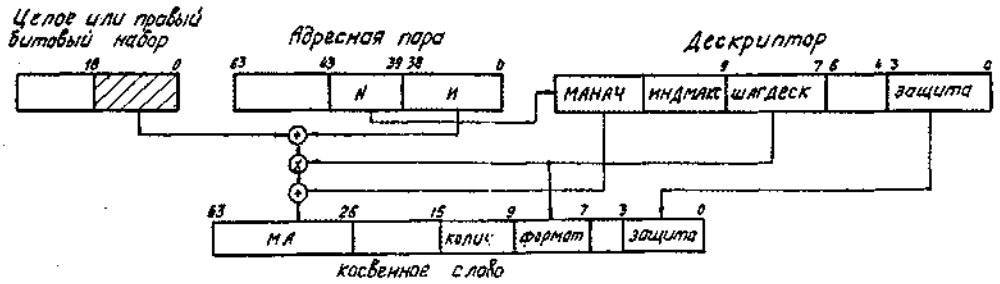


Рис.14.

1.4.2. Индексация интервалом

Индексация интервалом (Рис.15) выполняется теми же операциями, что и индексация целым числом: ИНД - индексация; и ИНДЗ - индексация с загрузкой.

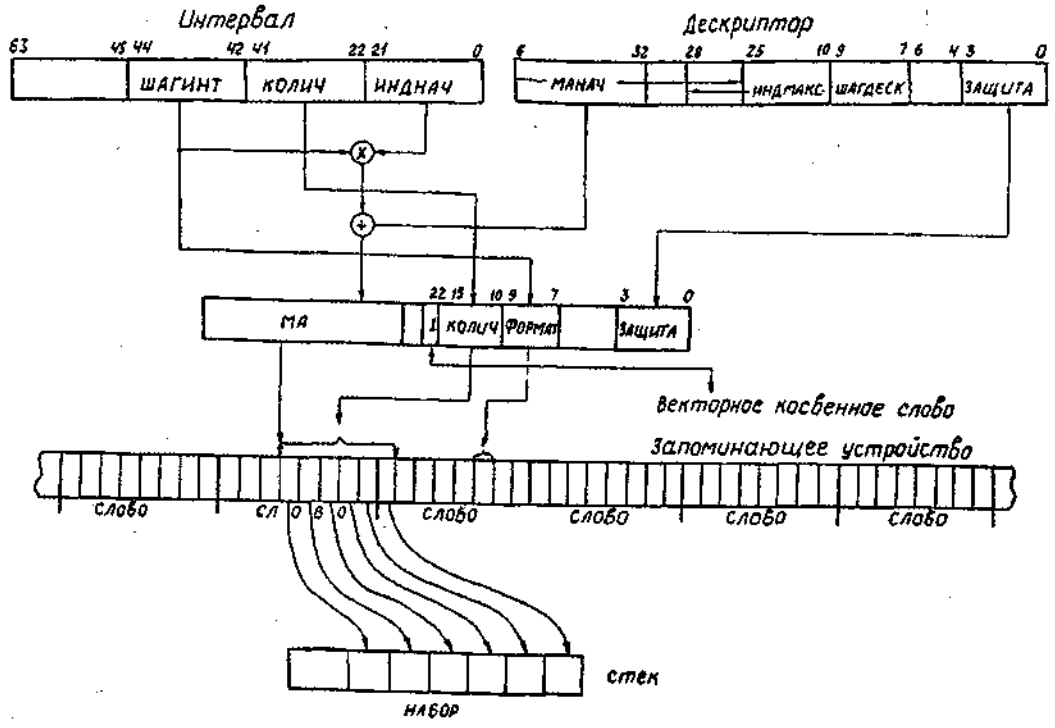


Рис.15

При индексации интервалом с помощью операции ИНД интервал описывает подмассив, который должен умещаться в один набор.

В результате операции получается косвенное слово, описывающее этот подмассив (так называемое векторное косвенное слово). Содержимое полей ФОРМАТ и КОЛИЧ в этом косвенном слове соответственно равно содержимому полей ШАГИНТ и КОЛИЧ в интервале. Поле МА в результирующем косвенном слове вычисляется по следующей формуле:

МА = МАНАЧ + ШАГИНТ x ИНДНАЧ.

Информация о защите переписывается в косвенное слово из индексируемого дескриптора. Индексация интервалом с помощью операции ИНДЗ эквивалентна последовательному выполнению операций ИНД и ВЗА. При загрузке в стек подмассив, описываемый векторным косвенным словом, превращается в левый набор. При записи из стека набора по векторному косвенному слову записывается подмассив. (Информация о количестве записываемых элементов массива и их формата берется из векторного косвенного слова. Запись может производиться в 2 смежных слова).

Таким образом векторное косвенное слово представляет средства для преобразования подмассива в набор и наоборот.

1.4.3. Индексация индексным словом

Индексация индексным словом (Рис.16) используется при организации циклов вместе с операцией КЦ - конец цикла.

Индексация индексным словом может выполняться следующими операциями:

ИНД - индексация;

ИНДЗ - индексация с загрузкой;

ИНД+ - индексация со сложением;

ИНД+3 - индексация со сложением и загрузкой.

При индексации индексным словом в верхушке стека, кроме индексируемого дескриптора, находится адресная информация, указывающая расположение индексного слова (без лишней косвенности), либо само индексное слово.

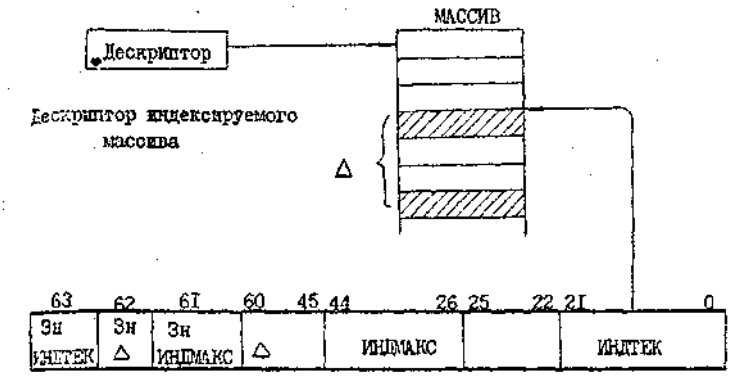


Рис.16. Индексация индексным словом.

Индексация индексным словом в операциях ИНД и ИНДЗ выполняется так же, как и индексация целым числом. В качестве целого используется текущее значение ИНДТЕК индексного слова.

Индексация индексным словом в операциях ИНД+ и ИНД+3 выполняется так же, как выполняются соответственно операции ИНД и ИНДЗ, но, кроме того, происходит модификация индексного слова и запись его на прежнее ме-

сто. Модифицируется поле текущего значения индекса ИНДТЕК следующим образом:

ИНДТЕК:= ИНДТЕК + Δ

Поле ИНДМАКС в этих операциях не используется (см. операцию КЦ).

1.5. Адресация к стековым сегментам

Стековые сегменты применяются для хранения результатов промежуточных вычислений. Дескриптор стекового сегмента, данные которого находятся в обработке (текущего стекового сегмента), хранится в специальном стековом регистре. Текущий стековый сегмент посредством специального счетчика, называемого указателем стека (УС), разбивается на 2 части: заполненную и свободную. УС указывает границу между этими областями (Рис.17).

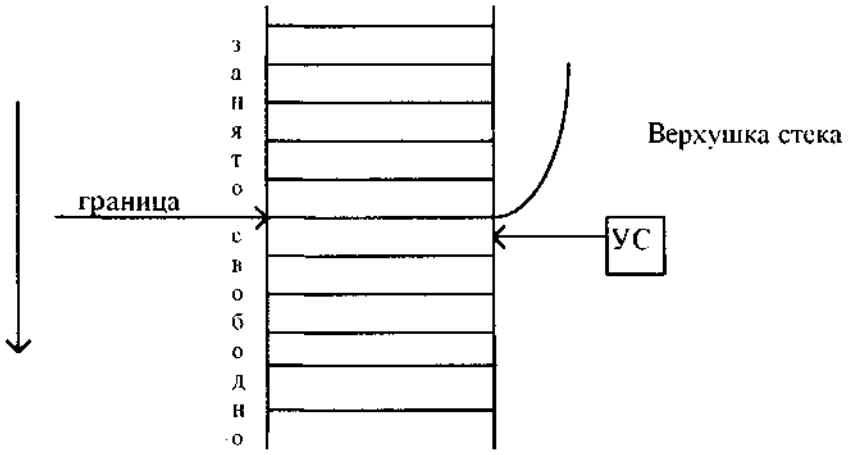


Рис.17. Магазинный принцип организации.
(стрелка слева показывает направление увеличения математических адресов).

Заполненная часть сегмента называется стеком, а некоторая непрерывная область заполненной памяти, которая граничит со свободной областью, называется верхушкой стека. Как правило, начальное нулевое положение УС совпадает с началом сегмента.

Математический адрес первой свободной ячейки в стековом сегменте получается путем сложения МАНАЧ дескриптора, описывающего текущий стековый сегмент, с содержимым УС.

Все операции в ВК производятся над величинами, расположенными в верхушке стека. После выполнения операции область памяти, занимаемая этими величинами, отводится в область свободной памяти.

Результат операции записывается в первую ячейку свободной памяти, и УС снова корректируется на начало области свободной памяти. Применение рассмотренного механизма позволяет использовать безадресную систему команд, так как МА величин определяется адресом, получаемым при индексации дескриптора текущего стекового сегмента содержимым УС. Дескриптор текущего стекового сегмента всегда находится в стековом регистре и поэтому явно не указывается. Содержимое УС находится в счетчике УС и поэтому также не указывается. Таким образом пара, по которой определяется МА первой свободной ячейки, известна аппаратуре.

В системе первые 16 ячеек в верхушке стека с помощью аппаратных средств помещаются в быстрых регистрах. Поэтому выполнение безадресных команд, как правило, не требует обращения к ОЗУ. Это обстоятельство позволяет существенно увеличить скорость выполнения программы.

Для удобства обработки в стеке каждая величина размещается в одном слове. Исключение составляют целые и вещественные величины формата 32: в одном слове могут размещаться две любых таких величины в любом порядке.

Величины формата 16 при загрузке в стек автоматически преобразуются в величины формата 32. Поэтому УС может передвигаться с точностью до полуслова, то есть изменение УС на единицу вызывает перемещение границы между областями свободной и занятой памяти на одно полуслово. Так как счетчик УС имеет 32 разряда, стековый сегмент может быть равен по объему всей математической памяти.

1.6. Адресация к программным сегментам

Выполняемая в любой момент времени программа представляет собой байтовый массив, расположенный в программном сегменте, и состоит из последовательности операций над верхушкой стека и операций передачи управления. Каждая команда состоит из одного или нескольких байтов. Первый байт в команде всегда отводится под код операции, последующие, если они имеются, для записи величин (литералов) и/или для идентификации вариантов операции. Многие операции, у которых имеются величины, имеют два варианта:

1) некоторые величины операций указаны в команде - статический вариант;
2) все величины считаются загруженными в верхушку стека. Обращение к

командам осуществляется с помощью двухбайтового счетчика
команд, для которого единицей отсчета является 1 байт. В ходе выполнения программы содержимое этого счетчика автоматически меняется в соответствии с форматом команд, обеспечивая последовательное выполнение команд Любая команда может начинаться с любого байта в слове. Нулевое состояние счетчика команд соответствует началу программного сегмента.

В процессоре имеется регистр, в котором хранится дескриптор программного сегмента, находящегося й работе.

Математический адрес команды получается из пары (дескриптор программного сегмента, содержимое счетчика команд) путем сложения МАНАЧ дескриптора программного сегмента с содержимым счетчика команд. Так как счетчик команд имеет 16 разрядов, размер программного сегмента не может превосходить 2^{16} байтов ~ 8000 слов.

Впоследствии было принято решение программные сегменты не загружать в математическую память. Это связано с тем, что программные сегменты не изменяются во время выполнения программы. Поэтому, если возникает необходимость убрать программный сегмент из физической памяти, его можно не откачивать в файл откачки (как это делается с сегментом данных), а просто уничтожить. Копия этого сегмента остается в кодовском файле. Поэтому, когда снова возникает необходимость поместить программный сегмент в физическую память, он считывается из кодовского файла.

Передача управления в различные части программы осуществляется операциями переходов путем простого изменения счетчика команд или по меткам (раздел 1.10.3 и 1.10.4). Метка, по которой передается управление, должна находиться в верхушке стека. В метке содержится информация об адресе дескриптора (в поле БАЗА), Описывающего программный сегмент, в который передается управление, и номер байта относительно начала этого сегмента (в поле Nk). Этот номер засылается в счетчик команд.

1.7. Адресация с помощью адресных пар

В ВК имеется 32 базовых регистра (дисплей-регистры), в каждом из которых может содержаться некоторый дескриптор. Обращение к областям, описываемым этими дескрипторами, производится при помощи адресной пары (N, И), где N - номер базового регистра, И - индекс, которым индексируется содержащийся в этом регистре дескриптор. Поскольку УС может передвигаться с точностью до полуслова, в базовые регистры помещаются дескрипторы, у которых поле ШАГДЕСК соответствует 32 разрядам. Такая возможность, кроме сокращения количества обращений к памяти при индексации дескрипторов, позволяет осуществить эффективное выполнение программ, написанных на языках высокого уровня.

Действительно, при трансляции с языков, имеющих блочную структуру (АЛГОЛ 60, PL/1, СИМУЛА, АЛГОЛ 68), именам переменных в входном языке соответствуют адресные пары в оттраснслированной программе *(N* соответствует лексикографическому уровню процедуры, в которой описано имя, а *И* соответствует относительному адресу переменной в области памяти, отведенной для этой процедуры).

Известно, что области памяти, отводимые для формальных и локальных параметров начатых, но незаконченных процедур, наиболее удобно хранить в стеке. В ВК предусмотрены аппаратные средства, обеспечивающие загрузку в соответствующие базовые регистры всех доступных для выполняемой процедуры областей памяти. При переходе от одной процедуры к другой с помощью аппаратных средств производится автоматическая перезагрузка базовых регистров в соответствии с изменением адресного пространства новой процедуры.

Следует отметить, что адресное пространство, доступное процедуре, существенно меньше, чем вся математическая память.

Поэтому количество разрядов в командах, отводимое для записи адресной пары, также существенно меньше количества разрядов, требуемых для записи полного МА. Таким образом, сокращается длина адресной части команд. Это обстоятельство, наряду с применением безадресных команд, позволяет значительно сократить объем программ по сравнению с объемом программ, написанных обычными методами программирования.

Кроме того, в системе команд предусмотрены средства, позволяющие автоматически загружать в базовые регистры дескрипторы любых сегментов, а не только областей, находящихся в стековых сегментах. В этом случае обращение к переменным в этих сегментах также может производиться с помощью адресных пар.

1.8. Операции загрузки в стек

Каждой загружаемой в стек величине отводится очередное слово из области свободной памяти стека. Из этого правила имеется одно исключение: целым и вещественным величинам формата 32 отводится половина слова. В одном слове упаковываются две любых таких величины в любом порядке. (Величины формата 16 при загрузке в стек преобразуются в величины формата 32.)

Информация может загружаться в стек либо непосредственно из командного потока (непосредственная загрузка), либо из области данных. В первом случае загружаемая величина находится в самой команде, во втором случае величина загружается по адресу, находящемуся либо в команде, либо в верхушке стека (загрузка по адресу).

1.8.1. Непосредственная загрузка

а) Универсальная непосредственная загрузка.

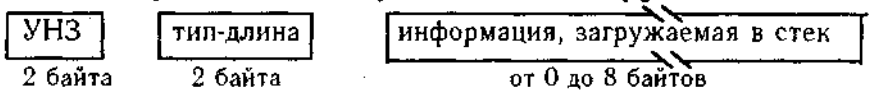


Рис.18. Вид команды УНЗ - универсальная непосредственная загрузка.

Непосредственная загрузка осуществляется командой УНЗ - универсальная непосредственная загрузка. Команда состоит из двух байтов кода операции, двух байтов тип-формат записываемой информации и от 0 до 8 байтов загружаемой информации (Рис.18). Заданная в команде информация должна умещаться в 1 слово. Эта информация загружается по адресу, задаваемому УС Слову, в которое загружается информация, приписывается тег в соответствии с информацией в поле тип-формат.

Исключение представляют целые формата 16 и 32 и вещественные формата 32 и 128. Перед загрузкой целое формата 16 преобразуется в целое формата 32.

Если загружается целое формата 32 или вещественное формата 32 во вторую половину слова, а в первой половине уже находится целая или вещественная величина формата 32, то всему слову приписывается ТЕГ, выделяющий одну из 4-х возможных комбинаций: целое-целое, целое-вещественное, вещественное-Целое и вещественное-вещественное.

Универсальная непосредственная загрузка вещественного двойной точности (128р) выполняется 3-мя командами. Двумя командами УНЗ загружаются 2 битовых полных набора (безразлично правых или левых), затем выполняется команда преобразования типа (ПУТ).

Особый случай представляет загрузка величины типа пусто. При загрузке величины типа пусто информационное поле в команде может отсутствовать.

б) Сокращенная непосредственная загрузка.

Предусмотрены команды, являющиеся частным случаем команды УНЗ, но имеющие меньший формат. Это команды: . 1) НЗА - непосредственно загрузить короткий адрес,

2) НЗДА - непосредственно загрузить длинный адрес,
3) НЗО - непосредственная загрузка целого нуля,
4) Н31 - непосредственная загрузка целой единицы,
5) НЗЛО - непосредственная загрузка логического нуля,
6) НЗЛ1 - непосредственная загрузка логической единицы,
7) НЗБ - непосредственная загрузка байта,
8) НЗКЦ - непосредственная загрузка короткого целого,
9) НЗЦ - непосредственная загрузка целого,
10) ЗБО - загрузка бита отношения.

Первые две команды предназначены для загрузки коротких и длинных адресных пар (Рис.19).

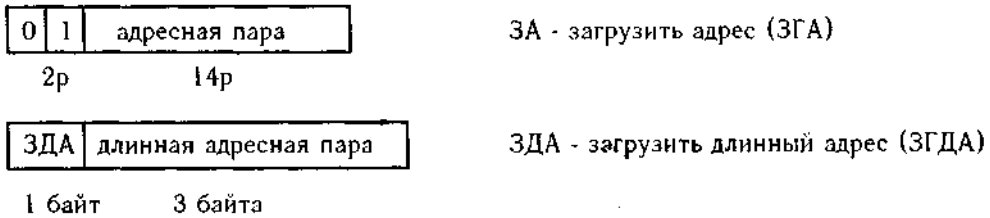


Рис.19. Вид команд для загрузки адреса.

Команда НЗА - двухбайтовая, а команда НЗДА - четырехбайтовая.

При загрузке в стек командами НЗА, НЗДА, НЗЛО, НЗЛ1, НЗБ, ЗБО информация занимает все 64 разряда информационной части, а в служебную часть слова устанавливается соответствующий тег.

Если в момент загрузки УС указывает на вторую половину слова, то в нее загружается техническая пустышка, которая пропускается при считывании из стека путем увеличения УС на единицу.

При загрузке командами НЗО, Н31, НЗЦ, НЗКЦ информация в стеке занимает половину слова и записывается как целая величина формата 32. Всему слову приписывается тег в зависимости от того, какой встретился вариант компоновки: целое-целое, целое-вещественное, вещественное-целое или вещественное-вещественное.

Команды 3-6 и 10 являются однобайтовыми, и информация о загружаемой величине содержится в коде операции.

Команды 5-6 загружают в стек величины типа бит, равные соответственно 0 и 1.

Команда 7 является двухбайтовой (Рис.20) и загружает в стек свой параметр.

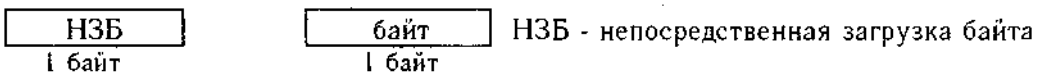


Рис.20. Вид команды.

Команды 8 и 9 предназначены для загрузки в стек целых форматов 32, не превосходящих по модулю соответственно $2^7 - 1$ и $2^{15} - 1$ (Рис.21).

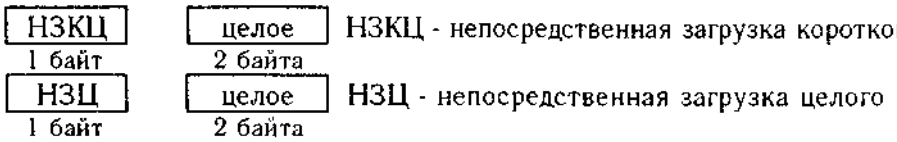
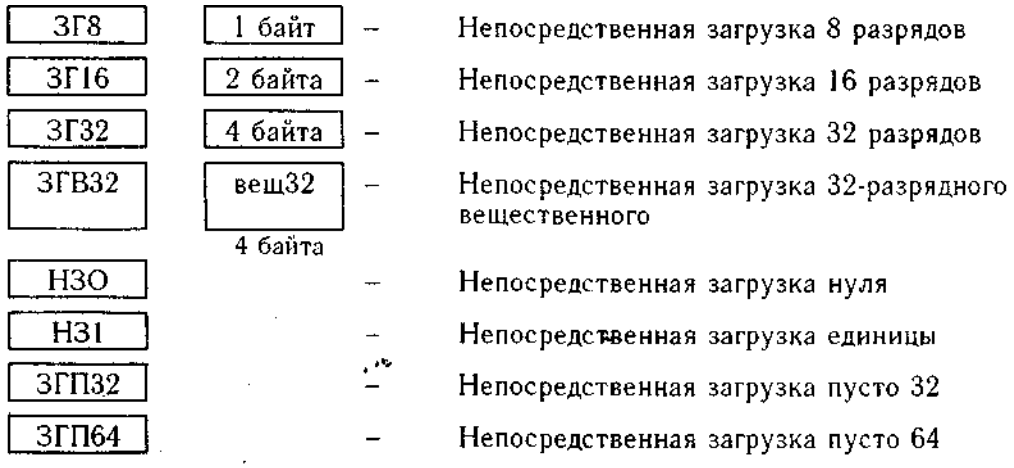


Рис.21. Вид команд НЗКЦ и НЗЦ.

Обе команды загружаются в стек целое формата 32.

В непривилегированном режиме запрещена загрузка адресной и управляющей информации, а также слов со спецтегом СТЗ.

Оставлены следующие укороченные команды непосредственной загрузки:



1.8.2. Загрузка в стек по адресу

Для загрузки в стек из области памяти имеются три группы команд.

К первой группе относятся команды, обеспечивающие загрузку в стек величин по адресу, находящемуся в самом командном потоке непосредственно за кодом выполняемой операции. (Этот адрес представлен адресной парой.)

Во второй и третьей группах загрузка величин в стек производится по адресной информации, находящейся в верхушке стека (динамический вариант загрузки).

Различие между второй и третьей группами состоит в том, что результатом операций третьей группы является адрес результата аналогичной операции второй группы.

К первой группе относятся операции:

1) ВЗ - взять значение,

2) ДВЗ - длинная команда взять значение.

Первая команда ВЗ имеет в качестве параметра короткую адресную пару, вторая ДВЗ - длинную адресную пару (Рис.22). Выполняются эти команды одинаково следующим образом.

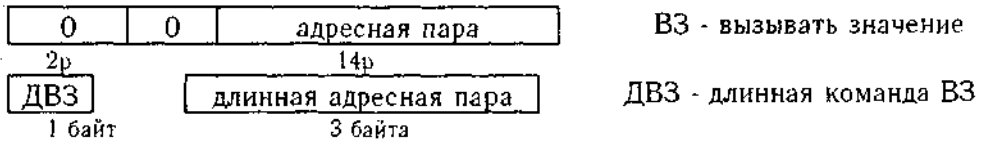


Рис.22. Вид команд ВЗ и ДВЗ.

По адресной паре (N, И) путем индексации дескриптора базового регистра под номером N и индексом И находится МА величины. Эта величина загружается в верхушку стека. При этом осуществляется контроль правильности обращения. Попытка обращаться в середину формата величины, определяемая путем анализа МА и тега слова, в котором находится величина, вызывает прерывание.

Загруженная в верхушке стека величина может принадлежать либо к классу значений, либо к классу адресов. Если она принадлежит к классу значений, то операция на этом заканчивается. Если она принадлежит к классу адресов, то в зависимости от ее типа предпринимаются следующие действия:

1. Если загруженная в верхушку стека величина является адресной парой, выдается прерывание.

2. Если загруженная в верхушку стека величина является косвенным словом, то происходит считывание по этому косвенному слову, и полученная величина загружается на место косвенного слова (делается еще один шаг по косвенности).

3. Если загруженная в верхушку стека величина является меткой процедуры (технической меткой), то происходит автоматический запуск процедуры по этой метке, причем после завершения процедуры в стеке оказывается поставляемый этой процедурой результат (запуск и выполнение процедуры также рассматриваются как один шаг).

Если в результате выполнения очередного шага в верхушке стека оказывается величина класса значений, то операция на этом заканчивается. Если же оказывается величина класса адресов, то делается следующий шаг, и описанный процесс повторяется до тех пор, пока в верхушке стека не окажется величина класса значений или не произойдет прерывание

Если в цепочке последовательных вызовов в верхушке стека окажется величина типа косвенное слово с признаком конца косвенности КК, равным 1, то следующая величина, загружаемая по этому косвенному слову, должна принадлежать к классу значений, иначе - прерывание.

Ко второй группе относятся следующие операции:

1) ВЗА - вызвать значение по адресу,
2) ВВА - вызвать величину по адресу,
3) ВВАШ - вызвать величину по адресу одношагово.

Операции второй группы - однобайтовые. Перед выполнением этих операций в верхушке стека может находиться любая величина. Если эта величина принадлежит к классу значений, то не предпринимается никаких действий, и эта величина является результатом операций. Если величина принадлежит к классу адресов, то делается один шаг, как это было описано в операциях ВЗ и ДВЗ. В дальнейшем эти три операции выполняются по-разному.

Для операции ВВАШ результат первого шага будет результатом операции независимо от типа величины, загружаемой в верхушку стека.

При выполнении операции ВЗА анализируется тип результата после первого шага, и если результат принадлежит к классу значений, выполнение операции на этом заканчивается. Если результат принадлежит к классу адресов, то делается следующий шаг, и так далее до тех пор, пока в верхушке стека не окажется величины класса значений. Таким образом, начиная с первого шага, выполнение операции ВЗА производится так же, как и операций ВЗ и ДВЗ (адресные пары допускаются только перед началом выполнения).

При выполнении операции ВВА анализируется тип результата после первого шага, и если результат принадлежит к классу значений, либо является указателем, либо обычной меткой, выполнение операции на этом заканчивается. Если в верхушке стека оказывается величина адресного типа - сквозная метка, либо сквозной указатель, делается следующий шаг до тех пор, пока в верхушке стека не окажется величина класса значений, либо указатель, либо обычная метка. Эта операция дает возможность при необходимости загружать в стек адресную информацию, доступную пользователю.

Если во время выполнения операций ВЗА, ВВА, ВВАШ в верхушке стека окажется косвенное слово с признаком конца косвенности КК=1, то величина, загружаемая в стек по этому косвенному слову, должна принадлежать к классу значений, иначе - прерывание. Во всех случаях, если при выполнении этих операций в стек загружаются величины типа управляющих слов, возникает прерывание.

К третьей группе относятся операции: ВАЗА - взять адрес значения по адресу, ВАВА - взять адрес величины по адресу, ВАВАШ - взять адрес величины по адресу одношагово, которые выполняются так же, как и операции соответственно ВЗА, ВВА, ВВАШ, но в отличие от них эти операции загружают в стек не сам конечный результат, а предпоследнее в цепочке последовательных вызовов косвенное слово, указывающее на эту величину.

Если при выполнении операций третьей группы окончательный результат поставляет процедура, то - прерывание, если перед выполнением операций третьей группы в стеке уже находится величина класса значений - также прерывание. Если при выполнении операции ВАВАШ в верхушке стека находится адресная пара, то результатом операции будет косвенное слово, соответствующее этой адресной паре. Если при выполнении операций третьей группы в верхушке стека окажется косвенное слово с признаком конца косвенности КК=1, то выполнение операций на этом заканчивается.

Операции третьей группы являются двухбайтовыми (Рис.23).

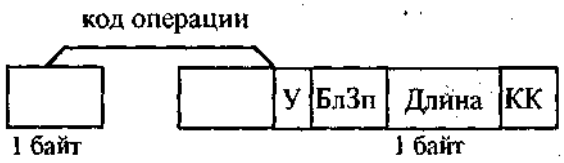


Рис.23. Вид команд ВАЗА, ВАВА, ВАВАШ.
У - *признак указателя, БлЗп - признак блокировки записи адресной информации, Длина - информация о формате значения, КК - признак конца косвенности.*

Во втором байте указывается информация, которая может быть записана в результирующее косвенное слово.

Признак указателя У и признак конца косвенности КК всегда записываются из 2 байтов операций.

Признак блокировки записи адресной информации БлЗп в результирующем косвенном слове устанавливается в 1, если его имеет предпоследнее косвенное слово в цепочке последовательных вызовов, либо второй байт команды.

Поле формата в результирующем косвенном слове заполняется из соответствующего поля предпоследнего косвенного слова. Если результирующее косвенное слово формируется из адресной пары, то поле формат заполняется из второго байта операций.

Ниже приводятся примеры выполнения рассмотренных операций. Пусть имеется цепочка косвенных слов и меток, показанная на Рис.24а.

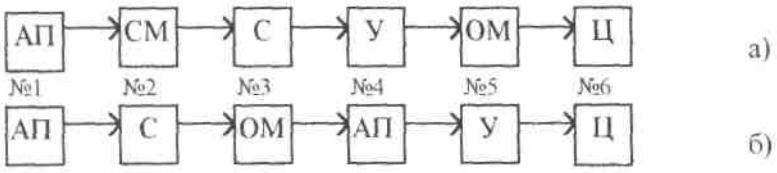


Рис.24. Вид цепочек последовательных меток.
АП - адресная метка, СМ -сквозная метка, С - ссылка, У - *указатель, ОМ - обычная метка, Ц - целое значение.*

Тогда результатом операции:
ВЗА будет целое № 6,
ВВА будет указатель № 4,
ВВАШ будет сквозная метка № 2, ВАЗА будет прерывание, ВАВА будет ссылка № 3, ВАВАШ будет ссылка №1, соответствующая адресной паре № 1.

Если имеется цепочка, показанная на Рис.246, то результатом вышеприведенных операций соответственно будут: прерывание, обычная метка № 3, ссылка № 2, прерывание, ссылка № 2, косвенное слово типа ссылка, соответствующее адресной паре № 1.

В дальнейшем оставлены только две операции КЦ и КЦС.

Для команды КЦ в вершине стека находится косвенное слово (аналог операции КЦАБ). Для команды КЦС в вершине стека находится индексное слово (аналог операции КЦИС).

В дальнейшем изменены мнемоника, состав команд и уточнены алгоритмы их выполнения. Оставлены команды:

ВЕЛ - аналог ВЗ, ДВЕЛ - аналог ДВЗ, СЧВЕЛ - аналог ВЗА, СЧВЕЛШ -аналог ВВАШ, САВЕЛ - аналог ВАВА.

Добавлена команда ПФП - передать формальный параметр, которая выполняется следующим образом. Если в начальный момент в вершине стека не находится косвенное слово или имя, то прерывание. Производится считывание по адресу в вершине стека и проверяется тип считанного значения.

Если считанное значение является косвенным словом или именем, то оно помещается в вершину стека в качестве результата. Операнд вычеркивается из стека.

Если считанное значение является технической меткой, то эта метка является результатом операции (в отличие от команды САВЕЛ, при которой запускается процедура по технической метке).

1.9. Запись в память из стека

Операции записи являются однобайтовыми. Они используют 2 величины, находящиеся в верхушке стека: записываемая величина и адрес записи. В качестве адреса записи может выступать либо адресная пара, либо косвенное слово. Запись производится "одношагово", то есть непосредственно по адресу записи.

Принято, что запись величины производится без какого-либо контроля забиваемой при записи информации. Перед операцией записи проверяется соответствие между МА, по которому производится запись, и форматом, указанным в операции записи (МА должен быть кратен формату, указанному в операции). Если формат записываемой величины меньше формата, указанного в операции записи, то величина записывается без изменения ее формата. При этом в информационную часть слова, в которое записывается величина, записывается тег величины и контролируется тип записываемой величины с типом, указанным в операции записи. В том случае, если информация о типе и формате записываемой величины имеется и в коде операции, и в адресе записи, информация о типе величины берется из кода операции, а информация о формате из адреса записи. В частности, если в коде команды тип величины не указан, контроль на соответствие типа не производится.

Из пяти операций записи 2 операции (ЗЦ32, ЗЦ64) выполняются с контролем типа и формата величины и три операции (ЗП32, ЗП64, ЗП128) выполняются только с контролем формата.

Все операции записи имеют 4 варианта исполнения, так как записываемая величина может находиться выше или ниже адреса записи, и запись может производиться с сохранением или без сохранения записываемой величины. Вариант с сохранением оставляет в стеке записываемую величину и вычеркивает из стека адрес записи. Вариант без сохранения вычеркивает из верхушки стека обе величины.

Эти варианты имеют следующее обозначение (в примере взята операция ЗП64);

ЗП64СА - запись с сохранением, когда в верхушке стека находится адрес записи;

ЗП64БА- запись без сохранения, когда в верхушке стека находится адрес записи;

ЗП64СВ- запись с сохранением, когда в верхушке стека находится величина;

ЗП64БВ- запись без сохранения, когда в верхушке стека находится величина.

Итак, если в качестве адреса записи стоит адресная пара, и если формат записываемой величины больше формата, указанного в операции записи, то операции записи выполняются следующим образом:

1. Операция ЗП32Ц контролирует тип и формат записываемого числа. Если число не целое и не правый битовый набор, который автоматически преобразуется

в целое, - прерывание. Если число целое, но формат больше (формат 64), то предпринимается попытка преобразовать это число в целое формата 32. При невозможности - прерывание.

2. Операция ЗП64Ц контролирует тип записываемого числа.

3. Операция ЗП32 используется для записи как вещественных, так и целых чисел. При этом придерживаемся того же принципа, что и при выполнении арифметических операций, то есть целое - это частный случай вещественного, поэтому при любой возможности записать целое пишется целое.

Если формат целого числа больше 32, то предпринимается попытка преобразовать это число в целое формата 32. Если попытка неудачна, то оно преобразуется в вещественное формата 32 с соответствующим округлением и прерыванием. Если записываемое число является вещественным формата, большего чем 32, то предпринимается попытка преобразовать его в формат 32. При удаче производится округление, при неудаче - прерывание. Если записываемая величина отлична от правого битового набора, целого или вещественного, и превышает формат 32 - прерывание.

4. Операция ЗП64 реализована аналогично операции ЗП32 с той лишь разницей, что формат записываемого числа равен 64. При этом целые всегда записываются как целые. Запись вещественных чисел производится по тем же правилам, что и в операции ЗП32.

5. Операция ЗП128 производит запись без всякого контроля формата записываемого числа.

Если в качестве адреса записи стоит косвенное слово, в котором уже имеется информация о формате записываемого формата в поле формат, то формат результата берется из косвенного слова, а информация о том, должен ли быть этот элемент целым, - из операции записи. Таким образом, запись по косвенному слову без контроля типа может производиться любой из операций ЗП32, ЗП64, ЗП128.

1.10. Формирование интервалов, индексных слов и меток

1.10.1. Операции формирования интервалов

Для формирования интервала введена однобайтовая операция:

ФИНТ - формирование интервала (динамический вариант) и 4-х байтовая операция;

ФИНТЛ - формирование интервала литерально (статический вариант).

Перед выполнением операции ФИНТ в верхушку стека должны быть помещены 3 целых числа формата 64 или битовых набора (при выполнении операции они автоматически преобразуются в целые), соответственно равных КОЛИЧ, ШАГИНТ, ИНДНАЧ (Рис.25).

Рис.25. Состояние верхушки стека перед выполнением операции ФИНТ.

После выполнения операции исходные величины вычеркиваются из стека, и в верхушку стека помещается сформированный Интервал. Операция ФИНТЛ имеет следующий вид (Рис.26).

Рис.26.

В операции ФИНТЛ поля интервала задаются в команде.

1.10.2. Операции формирования индексных слов

Для формирования индексного слова введена однобайтовая операция ФИС - формирование индексного слова. Перед выполнением операции ФИС в верхушку стека должны быть помещены 3 целых числа формата 64 или правых байтовых набора, соответственно равных.

Рис.27. Состояние верхушки стека перед выполнением операции ФИС.

После выполнения операции исходные величины вычеркиваются из стека и в верхушку стека помещается сформированное индексное слово (Рис.27).

В дальнейшем изменен порядок операндов в стеке на следующий ИНДМАКС (в вершине стека), ДЕЛЬТА, ИНДТЕК.

1.10.3. Операции формирования меток операции переходов

а) Формирование меток для операций переходов и процедур.

Передача управления по меткам осуществляется операциями переходов и операцией вызова процедуры "Вход".

Различие в выполнении этих операций состоит в том, что в случае перехода на базовом регистре номера LL устанавливается дескриптор, указывающий начало данных той процедуры или блока, к которому мы перешли (в поле МАНАЧ дескриптора передается поле АДРЕС метки), а информация, не относящаяся к этому блоку или процедуре и находящаяся ниже этих данных в стеке, вычеркивается из него, в то время как при выполнении операции "Вход" мы образуем новую область данных для той процедуры, в которую мы входим. Поэтому в случае операции "Вход" в базовый регистр с номером LL записывается дескриптор, МАНАЧ которого формируется из текущего значения УС, а математический адрес поля АДРЕС метки устанавливается в базовый регистр LL-I (раздел 2.1).

Ввиду этого различия формируется два типа меток соответственно двумя группами операций формирования меток. Сами метки различаются только тегами. Названия операций отличаются добавлением буквы П для операций образования процедурных меток.

Для передачи управления по меткам введены следующие команды формирования меток:

1) ФМ, ФМП - формирование метки,
2) ФМБ, ФМБП - формирование метки со сменой базы,
3) ДФМ, ДФМП - динамическое формирование метки,
4) ДФМБ, ДФМБП - динамическое формирование метки со сменой базы,

5) УФМ, УФМП - универсальное формирование метки.

В общем случае при формировании меток тем или иным образом указывается:

1) номер уровня той процедуры LL, в которую передается управление по формируемой метке,

2) информация об адресе перехода.

Эти оба параметра известны после трансляции. Формат первых 2-х операций показан на Рис.28.

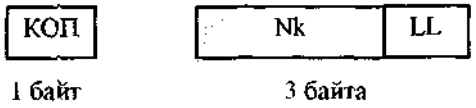


Рис.28. Вид операций ФМ и ФМБ.

Операция ФМ применяется для формирования меток, передающих управление в текущем программном сегменте. Поле БАЗА в формируемой метке заполняется информацией о текущем программном сегменте.

Операция ФМБ применяется для формирования меток, передающих управление в другой программный сегмент. Информация о базе нового программного сегмента должна находиться в верхушке стека. После выполнения операции эта информация вычеркивается из стека.

Операции ДФМ и ДФМБ являются динамическими вариантами соответственно операции ФМ и ФМБ. Информация о Nk должна находится в верхушке стека (Рис.29, 30, 31).

Рис.29. Вид операций ДФМ и ДФМБ.

Все эти операции, за исключением операций УФМ и УФМП, формируют такие метки, которые при выполнении перехода по ним производят изменение адресного пространства, приводя его к адресному пространству момента формирования метки, скорректированному с номером LL процедуры, в которую передается управление. Поэтому при формировании метки поле АДРЕС метки заполняется из поля МАНАЧ дескриптора, находящегося в базовом регистре с номером LL (Рис.32).

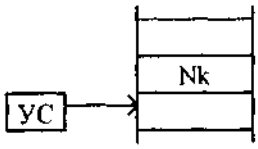


Рис.30. Состояние стека перед выполнением ДФМ.

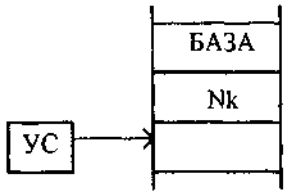


Рис.31. Состояние стека перед выполнением операции ДФМБ.

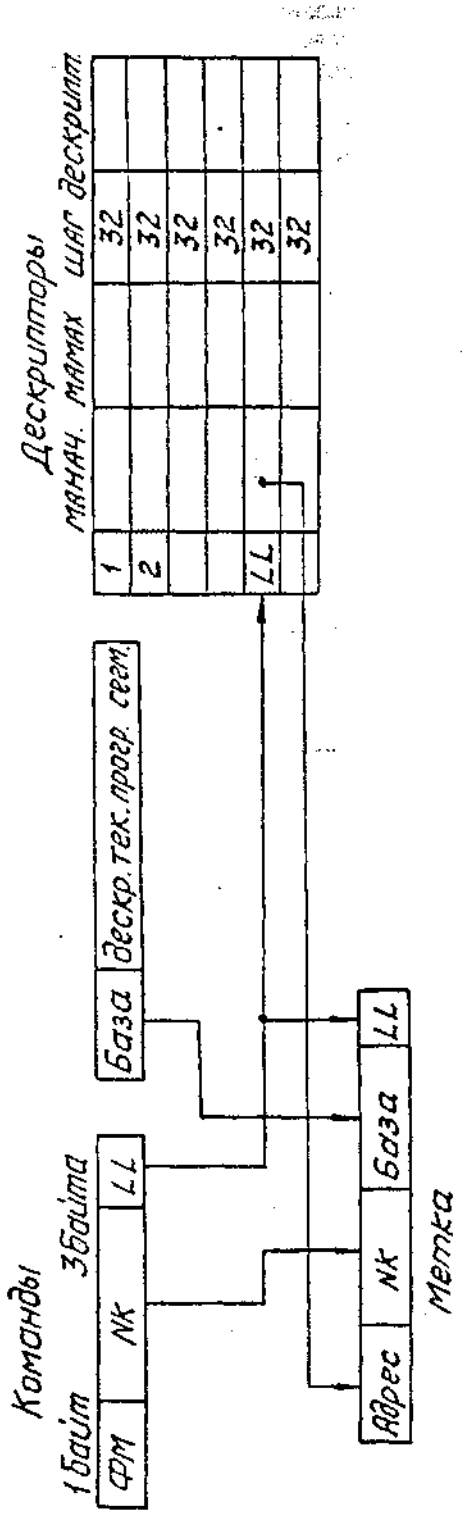


Рис.32. Выполнение операций УФМ и УФМП.

Операции УФМ и УФМП (Рис.33) применяется для формирования меток, передающих управление в любой программный сегмент со сменой адресного пространства, включая смену стекового сегмента.

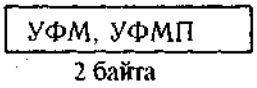


Рис.33. Вид операций УФМ, УФМП.

Перед выполнением операции в верхушке стека должны находиться целое, которое будет записано в поле **АДРЕС** метки, и заготовка (Рис.34).

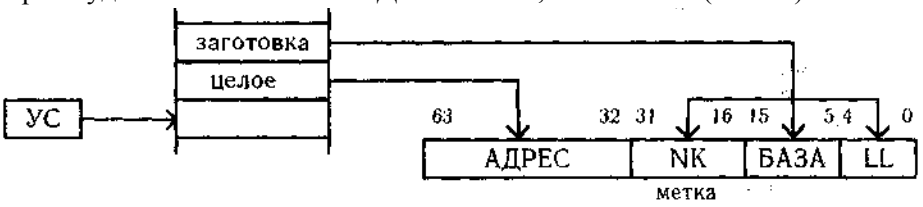


Рис.34. Выполнение операций УФМ и УФМП.

Заготовка представляет собой полный битовый набор, в котором в соответствующих местах записаны Nk, БАЗА и LL.

В дальнейшем команды ДФМ, ДФМП, ДФМБ исключены.

1.10.4. Операции перехода по метке

Так как вся информация о месте, на которое необходимо перейти, содержится в метке, а сама метка находится в стеке, эти операции имеют только динамический вариант и осуществляются следующими однобайтовыми операциями:

БПД - безусловный переход динамический,

УОД - условный переход по нулю динамический,

У1Д - условный переход по единице динамический.

В случае условных переходов в верхушке стека кроме метки должна содержаться величина (эталон); определяющая условие перехода. Эта величина может быть либо бит, либо неполный правый битовый набор, состоящий из одного элемента (Рис.35).

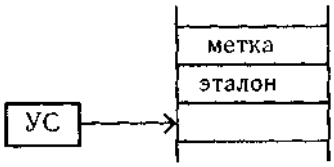


Рис.35. Состояние стека перед выполнением операций условных переходов.

Команда УОД вырабатывает переход по метке, если эталон равен нулю.

Команда У1Д вырабатывает переход по метке, если эталон равен единице

В случае невыполнения перехода операции УОД и У1Д вычеркивают саму метку и эталон из стека.

В случае выполнения условия перехода последующий алгоритм выполнения операций УОД и У1Д совпадает с алгоритмом выполнения операции БПД, который состоит в следующем:

1. Проверяется необходимость изменения адресного пространства. Если переход локальный, внутри адресного пространства выполняемой процедуры поле АДРЕС метки совпадает с МАНАЧ дескриптора базового регистра с номе-

ром LL происходит дальнейшее выполнение этой операции (случай №1). Если переход глобальный, происходит переход (прерывание) на процедуру операционной системы, которая продолжает выполнение этой операции (случай №2).

2. Случай №1. Происходит изменение счетчика Nk на номер Nk метки. Указатель стека устанавливается следующим образом:

а) По полю МАНАЧ дескриптора номер LL находится первое слово в стеке, относящееся к выполняемой процедуре (управляющее слово "маркер стека" (МКС)). В этом слове имеется поле РАЗМЕР, которое указывает поле, отводимое для данной процедуры.

б) Математический адрес самого маркера стека складывается с полем РАЗМЕР, и результат устанавливается в качестве УС.

Другими словами, УС устанавливается на начало процедуры с загруженными параметрами. Таким образом все величины, находящиеся в стеке ниже его нового значения, вычеркиваются. На этом операция заканчивается.

3. Случай №2. При изменении адресного пространства из стека неминуемо должны быть вычеркнуты данные одной или нескольких незавершенных процедур. Среди этих данных могут быть дескрипторы, описывающие сегменты, вызванные в оперативную память. Если вместе с вычеркиванием из памяти дескрипторов не будут убраны принадлежащие им массивы, оперативная память будет "засоряться" этими массивами до момента полного завершения всей задачи. Кроме того, сама текущая процедура может задавать определенную реакцию операционной системы на ее прерывания и так далее. Обычно при выходе из процедуры все ее последствия ликвидируют операции выхода из процедуры.

В данном случае необходимо просмотреть последствия не одной, а всех вычеркиваемых из стека процедур. В этом и заключается работа процедур операционной системы, вызываемых прерыванием.

В конце работы процедуры операционной системы должна стоять операция безусловный переход динамический (БПД). Параметром этой операции явля-ется та же самая метка, по которой произошел уход на операционную систему. Выполнение этой операции состоит в следующем:

а) По полю АДРЕС метки находится управляющее слово - маркер стека той процедуры, к которой необходимо осуществить переход.

б) Поле адреса.метки устанавливается в поле МАНАЧ формируемого дескриптора, который устанавливается в базовый регистр с номер LL метки. В поле ИНДМАКС устанавливается информация поля РАЗМЕР найденного управляющего слова, после чего производится корректировка базовых регистров по тому же алгоритму, что и при корректировке базовых регистров в момент выхода из процедуры (см. раздел "Процедуры").

в) По полю БАЗА метки находится дескриптор нового программного сегмента и устанавливается в базовый регистр текущего сегмента, счетчик Nk изменяется в соответствии с содержимым поля Nk метки.

г) Указатель стека устанавливается по математическому адресу нового маркера стека, измененному на величину поля РАЗМЕР самого маркера стека, в результате чего все данные процедур, находящиеся ниже указателя стека, вычеркиваются из стека.

На этом выполнение операции БПД заканчивается, а одновременно с этим заканчивается и выполнение операций БПД, УОД и У1Д в случае № 2.

Операция БПД имеет и самостоятельное значение, алгоритм ее при этом не меняется. Используя ее, пользователь сознательно идет на возможные последействия вычеркивания процедур. Воздействие этой операции в случае № 1 то же, что и операции БПД, однако алгоритм выполнения более сложный.

Совершенно иначе выполняются эти команды в том случае, когда вместо метки в верхушке стека находится целое или правый битовый набор (автоматически преобразуется в целое).

Алгоритм выполнения операций в этом случае тождественен алгоритму выполнения операций, описанных в следующем разделе настоящей главы. Случай перехода по целому вместо метки является динамическим вариантом статических операций БПН, УОН и У1Н.

д) Операции перехода внутри текущей процедуры. Адрес перехода в описываемых командах указывается непосредственно в командной последовательности, ввиду чего операции имеют следующие наименования:

БПН - безусловный переход непосредственный,

УОН - условный переход по нулю непосредственный,

У1Н - условный переход по единице непосредственный.

Эти операции могут передавать управление только внутри текущего программного сегмента и текущей процедуры. Информация об адресе перехода содержится в самой команде (Рис.36). Переход производится путем засылки в счетчик команд целого числа, содержащегося в команде.

Рис.36. Вид операций непосредственных переходов.

В случае условных переходов в верхушке стека должна содержаться величина, определяющая условие перехода. Возможные типы величин и условия, определяющие переход, такие же, как и в операциях динамических условных переходов УОД и У1Д. После выполнения этих операций в любом случае величина, определяющая переход, вычеркивается из верхушки стека, после чего указатель стека УС остается без изменения.

В дальнейшем операция БПДН исключена.

2. Обработка информации

Основной единицей обработки информации в ВК "Эльбрус" является процедура. В различных языках способы организации процедуры несколько различны. Процедура в ВК "Эльбрус" обладает следующими основными свойствами:

1. Телом процедуры является программа практически неограниченной длины.

2. Процедура имеет имя, по которому может быть вызвана "дистанционно"

3. Процедура имеет доступ к собственным данным (локальные, данные) и данным других процедур (глобальные данные). Доступ к глобальным данным регламентирован ее лексикографическим уровнем, задаваемым статически текстом программы, то есть каждая процедура имеет свое собственное адресное пространство.

4. Процедура может быть вызвана рекурсивно.

5: Фактические параметры процедуры определяются запускающей ее процедурой.

6. Процедура может поставлять значение и использоваться для получения "побочного" эффекта.

7. Исполнение процедуры определенным образом зависит от типа и форма-та поставляемых фактических параметров.

Наиболее полно этим свойствам удовлетворяют процедуры языков "Алгол 60" и "Алгол 68". Аналогичные понятия: этих и других языков можно рассматри-вать частным случаем вышеописанного понятия процедуры. Так "блок"

"Алгола 60" и "Алгола 68" можно рассматривать как процедуру, которая выполняется там, где она описана (процедура без имени, не обладающая свойст-вом дистанционного вызова).

Понятие "подпрограмма" и "функция" в языке "ФОРТРАН" может рассматриваться как процедура, не обладающая свойством рекурсивного запуска и описанная на одном лексикографическом уровне.

Операторная функция в языке "ФОРТРАН" может рассматриваться как процедура, не обладающая свойством рекурсивности, лексикографически описанная внутри подпрограммы либо функции.

Стандартные операторы операций: +, -, *, :, < и так далее, также являются процедурами, статически описанными на самом низком уровне (имена этих процедур доступны всей системе, включая процедуры операционной системы), тело которых вместе с их локальными данными реализованы аппаратно. Благодаря этим свойствам запуск таких процедур значительно упрощен: нет необходимости переключать счетчик команд, нет необходимости менять адресные пространства. Однако механизм постановки фактических параметров остается без изменения Таким образом, структура реализации всех операций, принятая в ВК "Эльбрус", тождественна с реализацией процедур.

Необходимо отметить, что алгоритм выполнения аппаратных процедур (операций), работающих с величинами типа значений, как правило, начинается с выполнения микрокоманды эквивалентной операции "взять значение одношагово", в то время как программы процедур чаще всего начинаются с операции просто "взять значение". Таким образом в том случае, когда в аппаратные процедуры (операции) в качестве параметра поступает величина типа косвенное слово или метка, происходит прерывание, в аналогичном же случае программные процедуры инициируют - обращение по косвенному слову или запуск процедуры по метке.

В дальнейшем изложении величины, подаваемые в качестве параметров аппаратным процедурам (операциям), будем именовать операндами.

2.1. Процедуры

2.1.1. Реализация обращений к процедурам в языках ФОРТРАН, КОБОЛ, ЛИСП

Для простоты рассмотрим сначала способ обработки процедур в языках, не имеющих блочной структуры (например, ФОРТРАН, КОБОЛ, ЛИСП, APL). В процедурах таких языков доступны лишь локальные параметры и фактические параметры, поставляемые через механизм передачи параметров (реализация COMMON в языке Фортран в настоящем разделе не рассматривается и будет рассмотрена в последнем разделе этой главы).

Распределение памяти для каждой вызываемой процедуры производится динамически. При входе в процедуру ей отводится в стеке необходимая область свободной памяти для формальных и локальных параметров, и УС передвигается на новую границу. Отводимая процедуре область памяти описывается дескриптором, помещенным в один из базовых регистров, например, с номером *N*. Обращение к величине в этой области осуществляется с помощью адресной пары *(N, И)*. В рассматриваемом упрощенном случае достаточно одного базового регистра, поэтому N является просто адресом одного из базовых регистров. В дальнейшем при реализации блочных структур N будет указывать уровень процедуры. Математический адрес величины вычисляется путем индексации дескриптора в базовом регистре с номером *N* индексом *И*. Область свободной памяти в стеке ниже новой границы используется для хране-

ния промежуточных результатов. При выходе из, процедуры Отведенная ей область памяти возвращается обратно в область свободной памяти путем восстановления прежнего УС. Таким образом, каждой процедуре отводится ровно столько памяти, сколько потребует динамика ее выполнения, и только тогда, когда это необходимо. Если из рассматриваемой процедуры *P* производится обращение к другой процедуре *Q,* то для процедуры *Q* отводится область свободной памяти точно таким же образом, как и для процедуры *P.* При этом переменные, описанные в процедуре *P,* и ее промежуточные результаты остаются в стеке и могут быть вновь использованы, когда произойдет возврат из процедуры *Q*.

При выходе из процедуры *Q* необходимо восстановить содержимое базового регистра с номером N и передать управление на оператор в процедуре Я, следующий за оператором вызова процедуры *Q*. Для этого при входе в любую процедуру в начале области свободной памяти отводится 2 слова для связующей информации, состоящей из специальных управляющих слов:

1) МКС - маркер стека (Рис.12),
2) УСВ - управляющее слово возврата (Рис.13).

МКС в некотором отношении аналогично дескриптору, описывающему отводимую область, так как оно содержит размер отводимой процедуре области для формальных и локальных параметров (в поле РАЗМЕР), а адрес МКС задает начало отводимой области (само МКС является первым словом в отводимой области).

УСВ похоже на метку, по которой осуществляется возврат из процедуры *Q* в процедуру Р, так как оно содержит информацию об адресе программного слога, на котором было прервано выполнение процедуры *P* (поля Nk и БАЗА), и информацию, необходимую для восстановления содержимого базового регистра для процедуры *P* и ее режима (поля Д АДРЕСА, ТРИГГЕРЫ, РЕЖ).

Для организации обращения к процедурам введен еш,е один регистр, называемый указателем маркера стека (УМС). Необходимость этого регистра вызвана тем, что загрузка формальных параметров производится из области данных запускающей процедуры (базовый регистр указывает на МКС запускающей процедуры и адресные пары оттранслированной программы (№ 1) адресуют относительно МА этого МКС). В то же время в момент входа в новую процедуру необходимо знать МА новой связующей информации, где находится метка процедуры, на выполнение которой необходимо перейти на этот переходный Период и необходим указатель маркера стека УМС.

При вызове процедуры *Q* принята следующая последовательность операций:

1. Загружается в стек метка процедуры *Q,* то есть информация о том, в каком месте математической памяти находится вход в процедуру *Q*.

2. Выполняется команда МС - маркировать стек. При этом:

а) отводится свободная ячейка в верхушке стека, и в нее помещается заготовка УСВ для вызываемой процедуры *Q;*

б) в поле Δ АДРЕСА заготовки УСВ записывается разность между адресом этого УСВ и содержимым УМС (формируется так Называемая динамическая цепочка). В УМС записывается адрес МЕТКИ.

Состояние стека до и после выполнения операции МС показано на Рис.37.

3. Следуют команды, которые последовательно загружают в стек информацию о фактических параметрах.

Состояние стека после этих операций показано на Рис.38, Предполагается, что процедура *Q* имеет 2 параметра *П1* и *П2.*

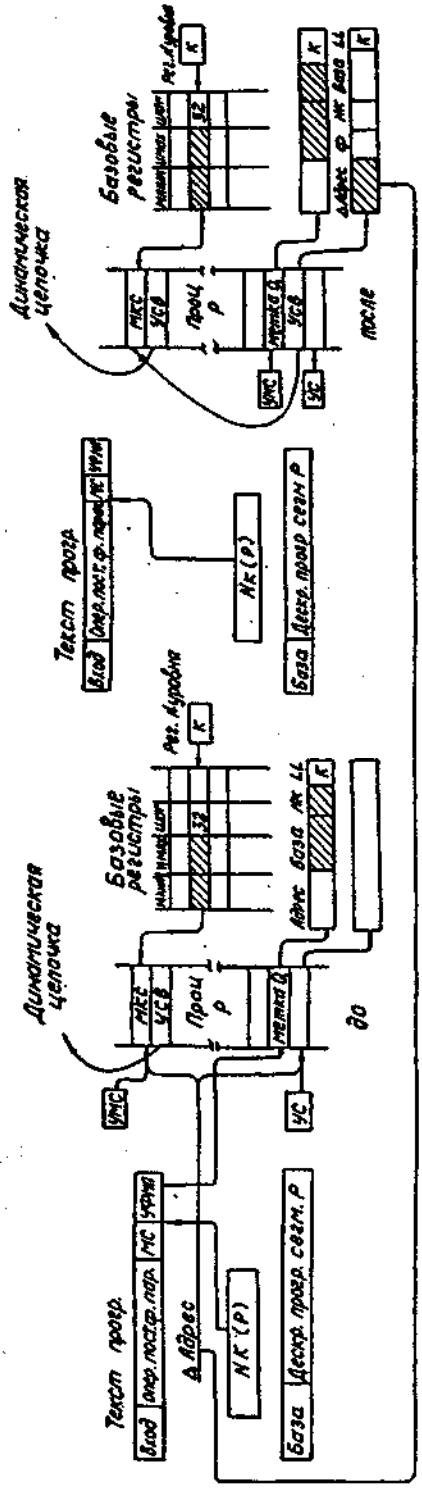


Рис.37. Состояние стека до и после выполнения операции МС.

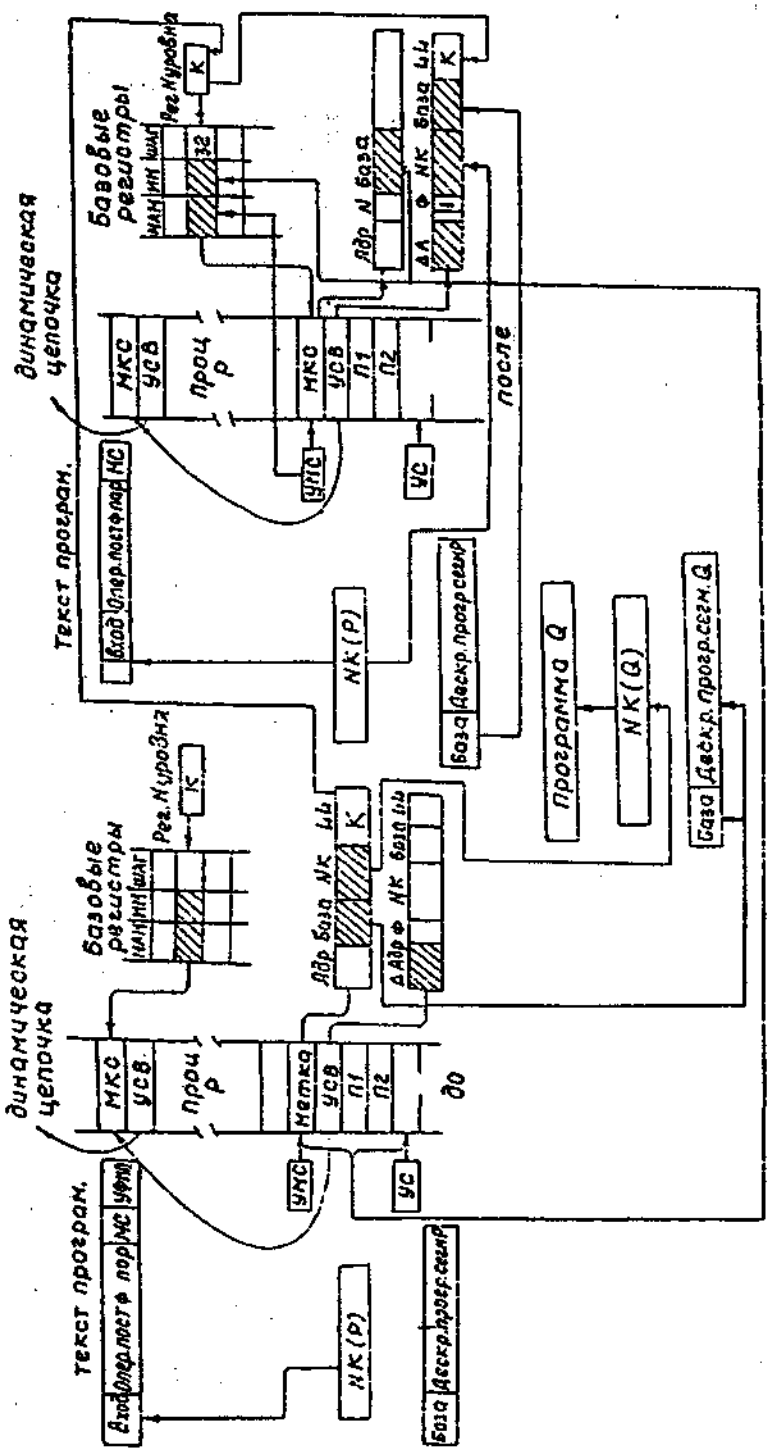


Рис.38. Состояние стека до и после выполнения операции ВХОД

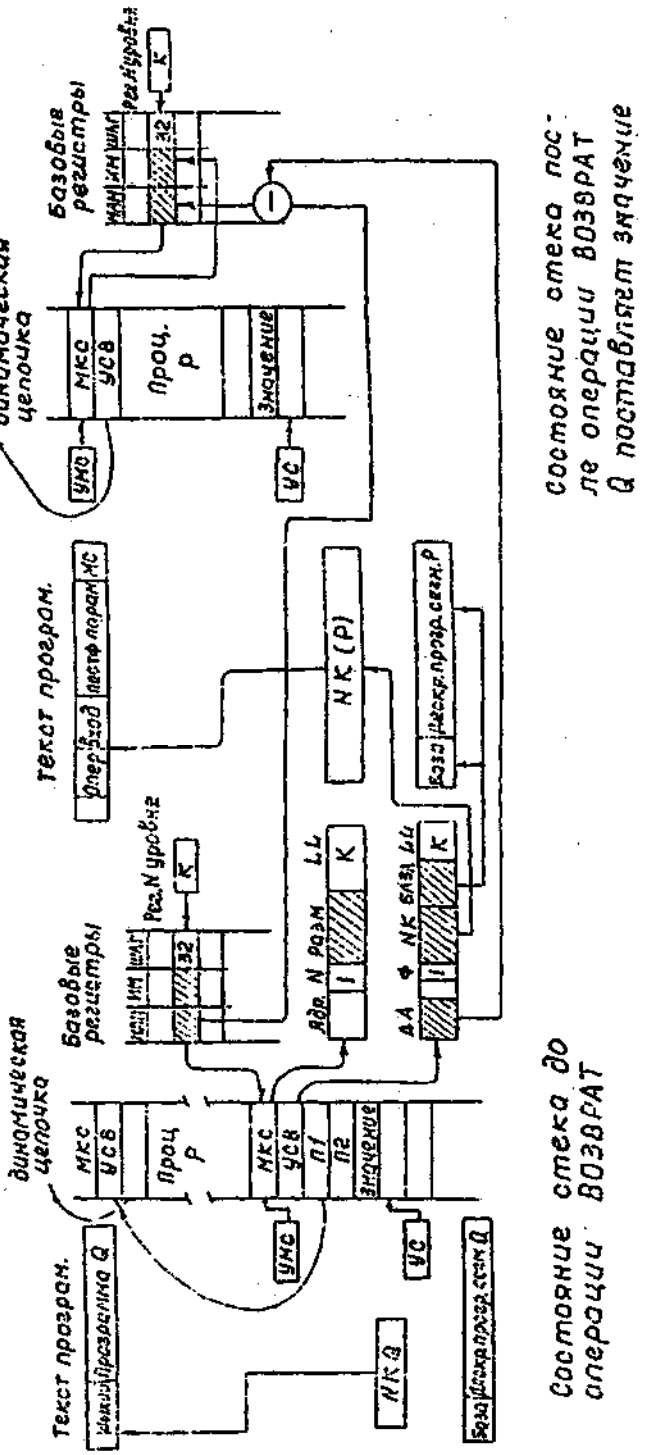


Рис.39. Состояние стека до и после выполнения операции ВОЗВРАТ.

4. Выполняется команда ВХОД. В результате выполнения этой команды окончательно формируется связующая информация и управление передается на вход процедуры (Рис.38).

Команда ВХОД выполняется следующим образом:

а) По адресу, указанному в УМС, находится метка процедуры *Q,* и на ее место записывается МКС процедуры *Q*.

б) Устанавливается признак Ф в слове УСВ, показывающий, должна ли вызываемая процедура *Q* поставлять значение. (Операция ВХОД имеет 2 варианта: ВХОД и ВХОДФ. Вариант ВХОДФ применяется при обращении к процедурам-функциям.)

в) Устанавливается размер области, занятой формальными параметрами, в поле РАЗМЕР слова МКС вызываемой процедуры *Q*.

г) В соответствии с содержимым вновь сформированного МКС в базовый регистр записывается дескриптор, описывающий область, отведенную для формальных параметров процедуры *Q*. В поле ИНДМАКС записывается та же величина, что и в поле РАЗМЕР слова МКС.

д) Управление передается в процедуру *Q*. При этом по содержимому поля БАЗА метки в регистр базы записывается дескриптор нового программного сегмента процедуры *Q*, а в счетчик команд записывается содержимое поля Nk - номер программного слога, с которого начинается выполнение процедуры *Q*.

Выход из процедуры *Q* осуществляется командой ВОЗВРАТ (Рис.39), которая обязательно должна встретиться в процедуре *Q*. При этом:

а) по содержимому регистра N уровня находим базовый регистр *K,* в котором записан МА МКС и УСВ процедуры *Q;*

б) путем анализа признака Ф в УСВ процедуры *Q* решается вопрос, оставлять ли значение в стеке;

в) по информации, хранящейся в поле А АДРЕСА УСВ процедуры *Q* (динамическая цепочка) поля МАНАЧ дескриптора *K,* находится МКС процедуры Р и формируется новый дескриптор в базовом регистре. Поле ШАГДЕСК этого регистра записывается из поля РАЗМЕР МКС процедуры *P*;

г) по информации, хранящейся в УСВ процедуры *P,* управление передается в процедуру *P*. При этом по содержимому поля БАЗА в регистр базы записывается дескриптор программного сегмента процедуры Р, а в счетчик команд записывается содержимое поля Nk - номер программного слова, с которого должно быть возобновлено выполнение процедуры *P*.

В данном разделе описаны частные действия операций МС, ВХОД и ВОЗВРАТ, более общее описание этих операций дано в разделе 2.1.3.

2.1.2. Реализация обращений к процедурам в языках с блочной структурой

В языках с блочной структурой (например, АЛГОЛ60, ПЛ/1, СИМУЛА, АЛГОЛ68) процедуре, описанной на некотором лексикографическом уровне *n,* доступны, кроме ее собственных формальных и локальных параметров, параметры, доступные в блоках и процедурах, в которые вложена рассматриваемая процедура. Поскольку блок является частным случаем процедуры, у которой описание и вызов находятся в одном и том же месте программы, в дальнейшем для краткости будем рассматривать только процедуры.

Отметим, что непосредственный вызов процедуры *Pn,* описанной на лексикографичеком уровне *n,* возможен только в том случае, если перед этим была вызвана, но не закончена процедура уровня P_{n-1}, в которую лексикографически вложена процедура *Pn*. Отсюда следует, что для вызова процедуры *Pn* уровня *n* необходимо, чтобы перед этим были вызваны, но не завершены вложенные каждая в последующую процедуры P_{n-1}, P_{n-2},... P_1 , P_0 соответственно уровней *n-1, n-2,*....., /, *0* (Рис.40).

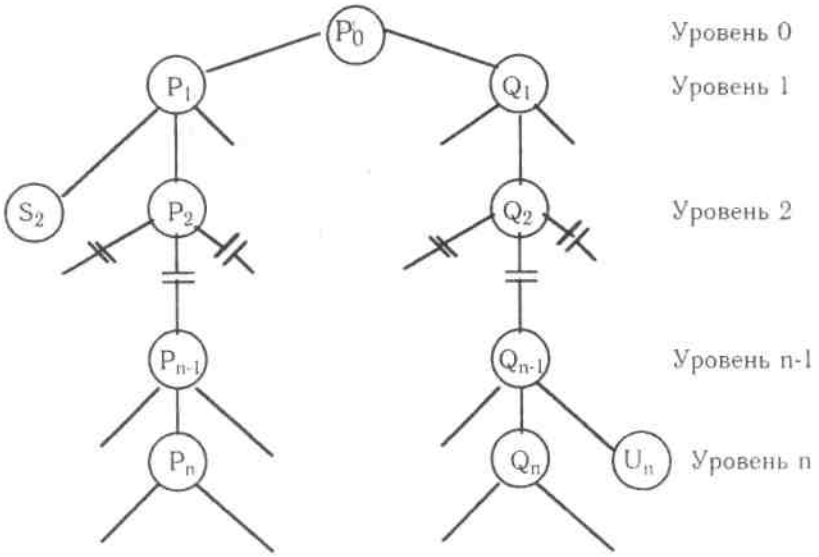


Рис. 40. Структура программы.

Таким образом, при вызове процедуры *Pn* в стеке уже должно существовать по крайней мере n областей памяти для этих вызванных, но незаконченных процедур, и переменные, описанные в этих областях, должны быть доступны рассматриваемой процедуре *Pn* на уровне *n.*

Адресация к переменным, доступным процедуре *Pm,* осуществляется при помощи адресных пар (*m, i*) В машине обеспечивается автоматическая загрузка в базовый регистр под номером *m (0 ≤ m ≤ n ≤ 31)* дескриптора, описывающего область памяти в стеке, выделенную для незавершенной процедуры *Pm,* переменные которой доступны процедуре *Pn.*

Рассмотрим, как осуществляется автоматическая загрузка базовых регистров в различных случаях. Отметим, что после трансляции известны лексикографические уровни всех процедур. Пусть процедура P_{n-1} вызывает процедуру *Pn.* В этом случае в *n-ый* базовый регистр достаточно загрузить дескриптор, описывающий область, отводимую для процедуры *Pn.* Если процедура P_{n-1} вызывает процедуру на меньшем лексикографическом уровне, например, процедуру S_2 на уровне 2 (Рис.40), то нулевой и первый базовые дескрипторы должны остаться без изменения, а во второй должен записываться дескриптор области в верхушке стека, отводимой для процедуры S_2 . Более сложная загрузка базовых регистров требуется при возврате из процедуры S_2 в процедуру P_{n-1} ,так как при этом необходимо восстановить содержимое базовых регистров со второго по *n-1.*

Чтобы обеспечить такой возврат, области памяти, доступные из процедуры P_{n-1} , объединены в список следующим образом. В поле АДРЕС слова МКС для процедуры P_{n-1} в момент входа в процедуру P_{n-1} (во время выполнения операции ВХОД) записан адрес МКС процедуры P_{n-2} . Аналогичным образом в МКС процедуры P_{n-2} записан адрес МКС процедуры P_{n-3} , из которой произошел вызов P_{n-2},, и так далее. В поле NIL слова МКС процедуры наинизшего уровня записывается признак конца списка. Таким образом все области памяти, доступные процедуре P_{n-1}, объединены в список (статическая цепочка).

Восстановление базовых регистров при возврате из процедуры S_2 в процедуру P_{n-1} осуществляется во время операции ВОЗВРАТ следующим образом.

По содержимому УМС система определяет адрес МКС выполнявшейся процедуры. По динамической цепочке в связующей информации система находит адрес МКС процедуры, в которую осуществляется возврат. (В примере это адрес МКС процедуры P_{n-1}) В слове МКС процедуры P_{n-1} в поле *LL* записан уровень процедуры, которая относится к этому МКС. (Эта запись производится при вызове процедуры во время выполнения операции ВХОД, смотри выше). В базовый регистр, определяемый полем *LL (n-1* - базовый регистр), записывается дескриптор, который формируется из найденного МКС. Этот дескриптор описывает область памяти, отведенную для процедуры P_{n-1} в момент обращения P_{n-1} к процедуре S_2. В поле АДРЕС слова МКС хранится начало статической цепочки, то есть адрес МКС процедуры P_{n-2}.. По этому адресу находится МКС для P_{n-2} и сформированный по этому МКС дескриптор записывается в базовый регистр с номером *n-2*. Аналогичным образом формируются дескрипторы в базовых регистрах с номерами *n-3, n-4,...., 1, 0.* Таким образом ,вызов процедуры и выход из нее для языков с блочной структурой отличается от работы с процедурами, описанной в предыдущем разделе настоящей главы, только корректировкой базовых регистров.

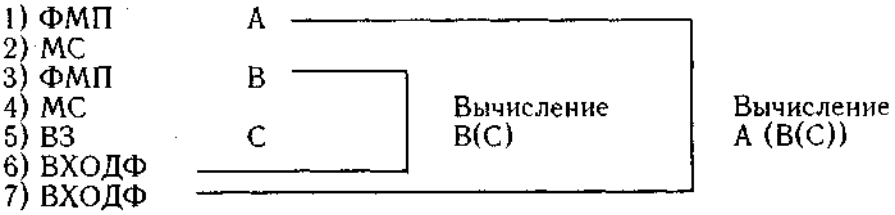
Как уже указывалось, при обращении к процедурам с параметрами после операции МС идут операции заполнения ячеек, отведенных под формальные параметры (формальных ячеек). Рассмотрим сначала случай, когда формальные параметры вызываются по значению. В этом случае в формальные ячейки в стеке должны быть загружены значения. Если фактические параметры являются простыми переменными, то загружаются в стек значения этих переменных и осуществляется вход в процедуру.

Если фактический параметр является выражением, то в программном потоке имеются операции для вычисления этого выражения, и результат остается в верхушке стека на месте соответствующего параметра. Например, обращение к процедуре *A (a, Ь+c)* уровня *n,* будет транслироваться следующим образом:

1)ФМП *n,* Nk
2) МС
3) ВЗ *n-2,* 3 % загрузка значения *a*
4) ВЗ *n-2,* 4 *%* загрузка значения *b*
5) ВЗ *n-1,* 3 % загрузка значения *c*
6) + % загрузка значения *b+c* вместо *b*
7) ВХОДФ

Предполагается, что значения переменных *a, b* и *c* хранятся в стеке соответственно по адресам *(n-2, 3), (n-2, 4), (n-1, 3).*

Формальные ячейки заполняются более сложным образом, если в качестве фактического параметра стоит обращение к процедуре-функции. Например, обращение к процедуре *A (B (C))* траслируется так:



Для определенности предположим, что вызов происходит из процедуры *P* уровня *n-1,* а процедуры *A* и *B* описаны в этой процедуре на уровне *n.*

Состояние стека до вызова *A (B(C))* показано на Рис.41а. Пусть вход в процедуру P_{n-1} завершен. Тогда для процедуры P_{n-1} уже сформирована связующая информация.

При вызове процедуры *A (B(C))* перед выполнением команды ВХОДФ происходит формирование меток *A* и *B* операцией ФМП.

При формировании меток в поле АДРЕС записывается информация, позволяющая определить статическую цепочку запускаемой процедуры, то есть адрес МКС на один уровень ниже запускаемой. В данном случае в поле АДРЕС обеих меток записывается адрес МКС процедуры P_{n-1}. При выполнении операций МС в коде Δ АДРЕС УСВ установлены Δ АДРЕСА, по которым замыкается динамическая цепочка. УМС всегда переключается на последнее в стеке МКС незавершенной процедуры. Поле БАЗА в метках *A* и *B* устанавливается из текущего базового регистра, а Nk соответственно из командного потока (Рис.41а и б).

Состояние стека после выполнения команды ВХОДФ показано на Рис.41 в. Метка, на которую указывал УМС, превращается в МКС, поле АДРЕС которого указывает на МКС *n-1* уровня. В базовом регистре *n* формируется дескриптор, описывающий поле процедуры P_n. Затем корректируются остальные базовые регистры. В данном случае по полю АДРЕС МКС процедуры *B* находится МКС процедуры P_{n-1} и формируется дескриптор, описывающий локальные данные этой процедуры.

Этот дескриптор должен быть записан в базовый регистр *n-1*. Однако в данном случае информация, записанная в нем, совпадает с той, которую необходимо записать, поэтому корректировка базовых регистров прекращается. Если бы в регистре была другая информация, то после записи в него дескриптора стека процедуры *B* по информации поля АДРЕС МКС процедуры *A* был бы найден МКС процедуры *n-2* уровня. Затем был бы сформирован дескриптор на локальные данные этой процедуры и записан в базовый регистр *n-2,* и так далее до совпадения информации базового регистра с вновь сформированным дескриптором.

После корректировки базовых регистров происходит изменение счетчика Nk, а если необходимо, и дескриптора текущего сегмента программы. Старая база дескриптора программы и старый Nk записываются в соответствующее поле УСВ процедуры *B*. В поле *LL* этого УСВ записывается информация из регистра уровня (в данном случае *n-1)*. В регистр уровня записывается уровень выполняемой процедуры. После этого в соответствующий разряд записывается 1, указывающая на то, что формирование связующей информации для данной процедуры завершено.

Операция ВОЗВРАТ в процедуре *B* поставляет в верхушку стека значение *B(C),* и управление передается в процедуру вызова *A(B(C))*. Для этого по динамической цепочке, используя поле Δ АДРЕС УСВ процедуры *B,* доходим до первого завершенного МКС (в нашем случае это МКС процедуры P_{n-1}). По этому МКС формируется дескриптор, который записывается в базовый регистр, номер которого указан в поле *LL* УСВ процедуры *B*. Этот же номер устанавливается в регистре номера уровня. Используя информацию поля АДРЕС МКС процедуры P_{n-1} , производится корректировка базовых регистров. Поле БАЗА и Nk УСВ процедуры *B* используются для заполнения базового регистра программы текущей процедуры и счетчика Nk, после чего МКС и УСВ этой процедуры вычеркиваются из стека, поставляя значение процедуры *B*. УМС переводится на последнюю метку стека связующей информации, используя информацию поля Δ АДРЕС УСВ (Рис. 41 г).

Затем операция ВХОД передает управление в процедуру *A*. При этом формируется МКС процедуры *A,* это МКС подсоединяется к статической цепочке, и корректируются по статической цепочке базовые регистры (Рис. 41 д).

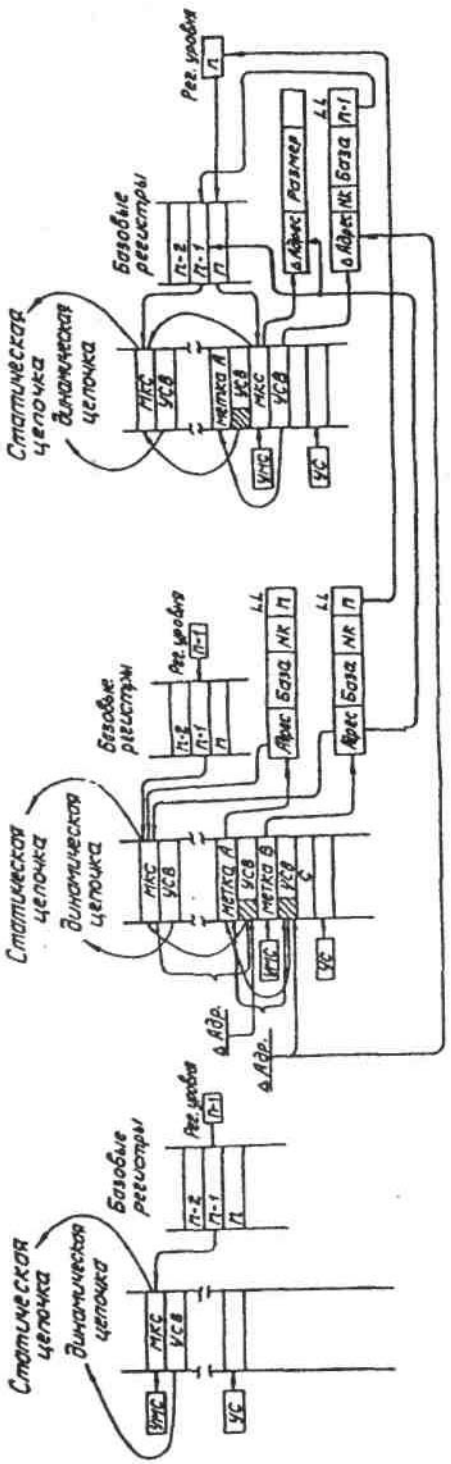


Рис.41. Изменение состояний стека при вызове A(B(C).
(Процедуры A и B вызывают свои формальные параметры по значению)

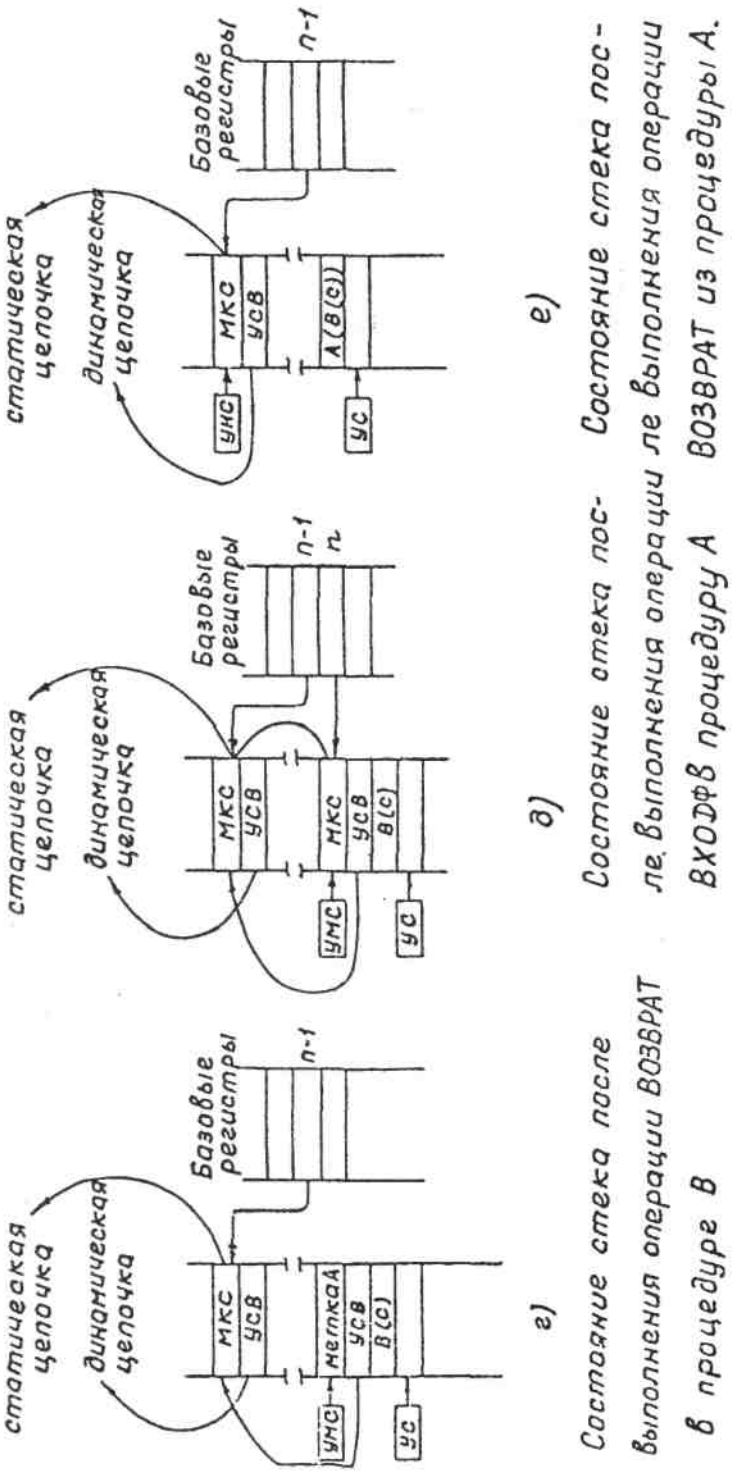


Рис.41. Продолжение

Операция ВОЗВРАТ в процедуре *A* поставляет в верхушку стека значение *A(B(C)),* и управление по динамической цепочке снова передается в процедуру P_{n-1} (Рис.41 е).

Полный перечень функций, выполняемых операцией ВХОД, представлен в разделе 2.1.3.

Рассмотрим теперь случай, когда при вызове процедуры *A* передача фактических параметров производится по имени. Если фактические параметры являются простыми переменными, то формальные ячейки заполняются косвенными словами на эти переменные. Если фактические параметры являются выражениями, то эти выражения оформляются как подпрограммы - функции без параметров того же уровня, тела которых находятся в месте вызова. В соответствующие формальные ячейки засылаются метки этих подпрограмм-функций. Например, обращение *A(a, b+ c),* где *A(x, y)* - процедура с двумя формальными параметрами, вызываемыми по имени, транслируется следующим образом:

1. ФМ *A* - формировать метку на процедуре *A*.
2. МС - маркировать стек.
3. ЗА *a* - загрузить адресную пару.
4. ВАЗА - преобразовать адресную пару в косвенное слово.
5. ФМП - на подпрограмму-функцию ПФ.
6. ВХОДФ

Выражение *b+ c* оформляется в такую подпрограмму-функцию:

ПФ: ВЗ *b*
ВЗ *c*
+
ВОЗВРАТ

Во время выполнения процедуры *A(x, y)* при обращении к формальному параметру *x* считывается косвенное слово, содержащее адрес значения *a,* автоматически происходит дальнейшее обращение по этому косвенному слову, и в верхушку стека считывается значение *a.*

При обращении к формальному параметру *y* считывается метка на процедуру без параметров, вычисляющую значение *b+ c.* При этом:

- автоматически в верхушку стека засылается метка ПФ на подпрограмму-функцию;
- автоматически выполняется команда маркировки стека МС, автоматически выполняется операция ВХОДФ, формируется связующая информация, и управление передается по метке.

Операция ВОЗВРАТ в подпрограмме-функции поставляет значение в процедуру *A* и передает в нее управление. Таким образом, обращение к формальному параметру *y* поставляет значение *Ь+ с.*

Если фактическим параметром процедуры *A (y)* является процедура с параметрами, например *B(x),* то обращение *A(B(x))* транслируется следующим образом:

1. ФМП на процедуру *A*
2. МС
3. ФМ на процедуру *B**
4. ВХОДФ

*B** является процедурой без параметров, которая осуществляет загрузку параметра *x* процедуры *B* и передачу управления на процедуру *B.* Тело процедуры *B** записывается рядом с вызовом процедуры *A* и имеет следующий вид:

B:* 1) ФМП на процедуру *B*
2) МС
3) ЗА *x* загрузить адрес переменной *x.*
4) ВАЗА преобразовать адресную пару в косвенное слово
5) ВОДФ
6) ВОЗВРАТ

Тело процедуры *B* может находится в совершенно другом месте программы.

Во время выполнения процедуры *A* обращение к формальному параметру процедуры *A* вызывает, как и в предыдущем случае, автоматическое выполнение приведенных операций. Управление передается в процедуру *B*.* В формальную ячейку загружается адресная пара фактического параметра *x.* Операция ВАЗА преобразует адресную пару в косвенное слово, содержащее МА параметра *x.* При обращении из процедуры *B* к своему формальному параметру *z* из формальной ячейки выбирается косвенное слово, которое указывает на значение *x.* Таким образом происходит временный возврат в место вызова *A(B(x)),* то есть хотя управление находится в процедуре *B, x* берется из того же окружения, которое имело место в момент вызова *A(B(x)).*

Операция ВОЗВРАТ в процедуре *B* поставляет результат *B(x)* в процедуру *B*.* Операция ВОЗВРАТ в процедуре *B** поставляет результат *B(x)* в процедуру *A,* а операция ВОЗВРАТ в процедуре *A* поставляет результат вызова *A(B(x)).* Изменение состояний стека показано на Рис. 42 а,б,в,г,д,е.

Механизм автоматического запуска процедур внутри операций типа "Вызвать значение" позволяет легко реализовать обращение к процедурам-функциям без параметров. Действительно, пусть, например, внутри некоторой процедуры встретилось выражение *a+ h,* где *h* - процедура-функция без параметров. Это выражение транслируется обычным образом:

ВЗ *a*
ВЗ *h*
+

В ячейке, соответствующей *h,* хранится метка на процедуру-функцию без параметров. При обращении к этой ячейке происходит автоматический запуск процедуры, и операция ВОЗВРАТ в процедуре *h* поставляет результат в верхушку стека. Таким образом конечный итог такой же, как и при вызове значения по адресу.

Перед выполнением операции МС в верхушке стека может находиться не обычная метка, а адрес, ведущий через цепочку из косвенных слов и сквозных меток к обычной метке. В этом случае операция МС заменяет адрес в верхушке стека на эту обычную метку, а затем уже маркирует стек. Это свойство операции МС используется следующим образом.

Например, пусть имеется процедура *A(x, τ*) и в процедуре известно, что формальному параметру *τ* соответствует в качестве фактического параметра имя другой процедуры. Тогда обращение *A(a, Q)* транслируется так:

ФМП *A*
МС
ВЗ *a*
ФМП *Q*
ВХОДФ

В момент обращения *A(a, Q)* в формальную ячейку, соответствующую *τ*, записывается обычная метка на процедуру *Q.*

Пусть в теле процедуры А встретилось обращение *τ(Ь).* Это обращение транслируется следующим образом:

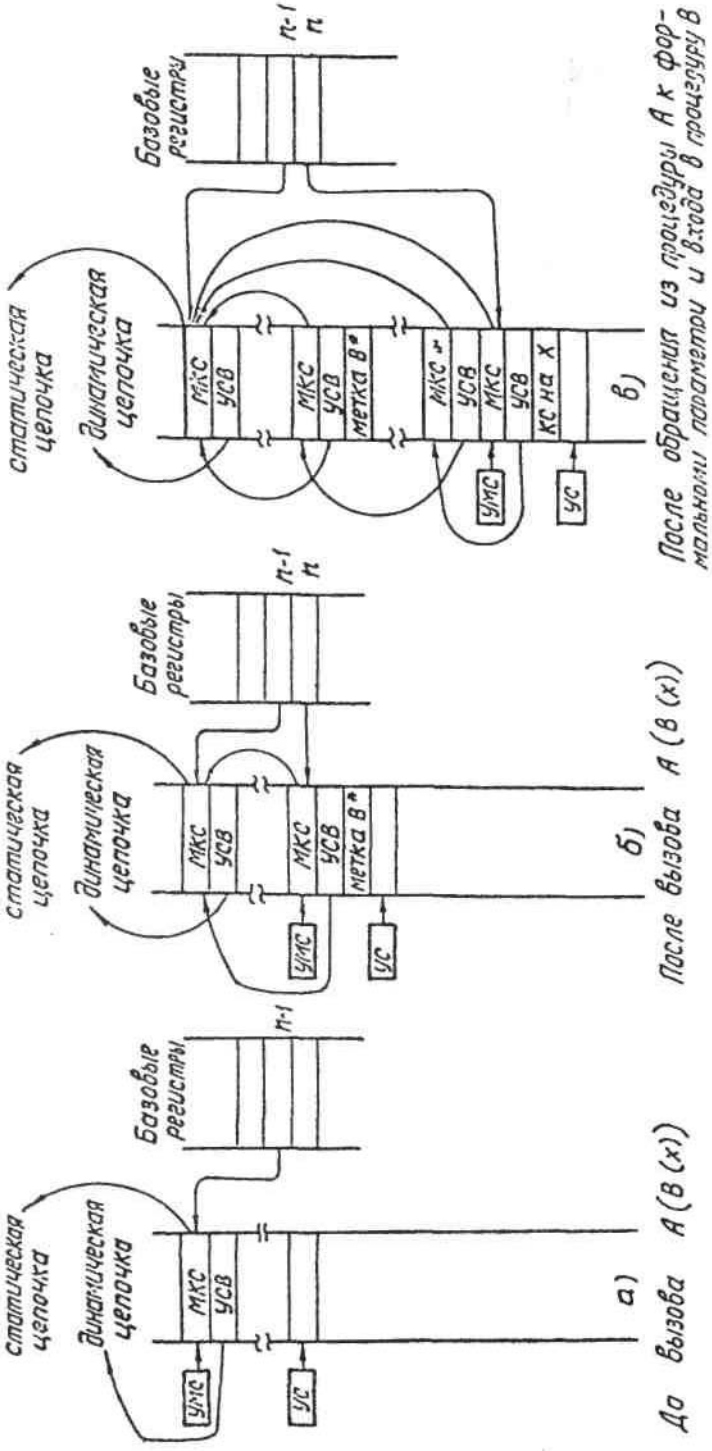
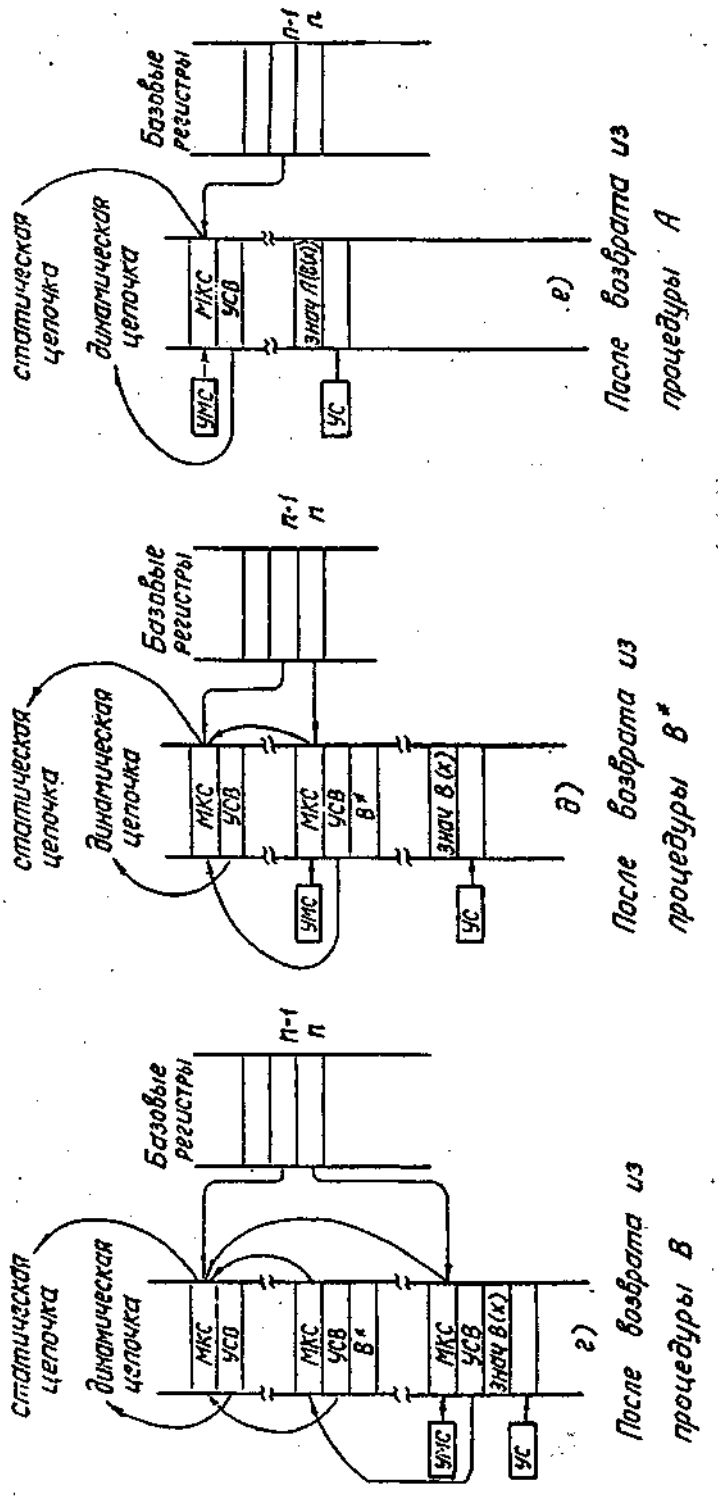


Рис.42. Состояние стека при вызове процедуры A(B(x))
(Формальные параметры вызываются по имени)



Изменение состояний стека при вызове процедуры A(B(x)) из процедуры Pn-1.
(Формальные параметры вызываются по имени)

Рис.42. Продолжение

ЗА*(n, 2)*
МС
ВЗ*b*
ВХОДФ

где: *(n, 2)* - это адрес формальной ячейки, соответствующий формальному параметру *τ* .

Операция МС обращается к верхушке стека. Там записана адресная пара *{n, 2).* Считывается значение по этой адресной паре. В верхушку стека будет считана обычная метка на процедуру *Q.* Затем маркируется стек, и операция МС на этом заканчивается. После этой операции загружается значение *Ь,* и управление передается в процедуру *Q.* Операция ВОЗВРАТ в процедуре *Q* будет поставлять значение вызова *Q (b)* в процедуру А.

Рассмотрим еще один пример. Пусть имеется следующий вызов:

if a. then b else c fi (x,y),

где: *a* имеет вид bool a *b* и *c* - процедуры с двумя параметрами.

В зависимости от значения *a* вызываются процедуры либо *Ь,* либо *c* двумя параметрами *x* и *y.*

Этот вызов может быть оттранслирован так:

ФМП*d*
Преобразовать сформированную метку в сквозную метку
МС
ВЗ*x*
ВЗ*y*
ВХОДФ

По метке *d* записана процедура, поставляющая обычную метку либо на процедуру *Ь,* либо на процедуру *c* в зависимости от значения *a.*

d: ВЗ*a*
УОН условный переход по нулю непосредственный
ФМП *b*
ВОЗВРАТ
ФМП c
ВОЗВРАТ

После того как сформирована сквозная метка *d,* операция МС передает управление по этой метке в процедуру, которая поставляет обычную метку. Дальнейшее выполнение вызова производится стандартным образом.

Кроме операции МС введена операция СМС - специальная маркировка стека. Выполнение этой операции показано на Рис.43.

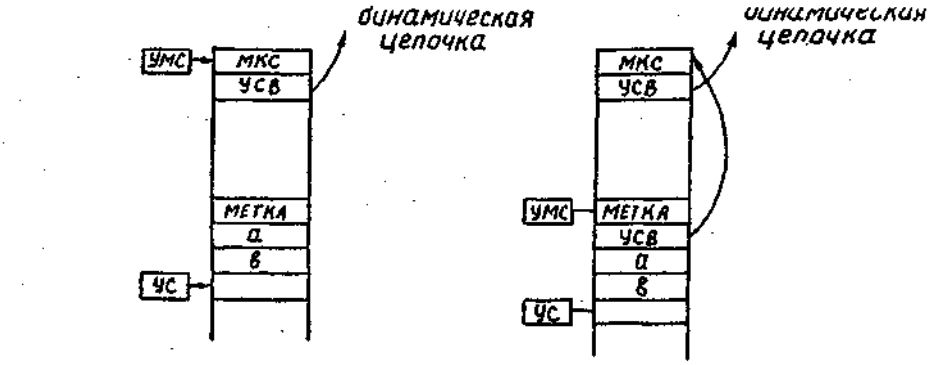


Рис.43. Выполнение операции СМС - специальная маркировка стека.

Обычно эта операция применяется в однопроходных трансляторах при вызовах процедур, записанных в инфиксной форме. Например, выражение *(a+Ь) ↑ (c+d)* представляет собой вызов процедуры возведения в степень с двумя параметрами. Вызов этой процедуры может быть оттранслирован следующим образом:

1. ВЗ *a*
2. ВЗ *Ь*
3. +
4. ВЗ *c*
5. ВЗ *d*
6. +
7. ФМП на процедуру возведения в степень
8. ПОВЕРНУТЬ ВВЕРХ
9. СМС
10. ВХОДФ

После операции № 8 уже вычислены два параметра вызываемой процедуры, и загружена метка процедуры, но необходимая для вызова процедуры операция маркировки стека еще не выполнена. Операция СМС как раз и исправляет нарушенную последовательность. После выполнения этой операции метка, УСВ и параметры оказываются на своих местах.

Если при выполнении операции СМС вместо обычной метки имеется адресная пара, косвенное слово или сквозная метка, то автоматически происходит поиск обычной метки по этой адресной информации, так же как и при выполнении операции МС.

Легко убедиться, что рассмотренный механизм распределения памяти при вызове процедур позволяет осуществлять рекурсивно обращение с произвольным уровнем рекурсии.

Поскольку при выполнении программы все переменные находятся в стеке и сама программа при этом совершенно не меняется, рассмотренный механизм распределения памяти автоматически обеспечивает повторную входимость всех программ, то есть возможность использования одной программы несколькими одновременно работающими задачами, каждая со своими данными.

Аналогичным образом обрабатываются и процедуры прерывания. Прерывание реализуется как аппаратный запуск процедуры с использованием текущего стека. Аппаратура обеспечивает автоматическую загрузку в стек необходимых параметров, организацию возврата и переход по соответствующей метке.

2.1.3. Операции МС, ВХОД, ВОЗВРАТ для организации вызова процедур

а) Операция "Маркировать стек" - МС.

Если в верхушке стека находится метка, то выполняются следующие действия.

1. В верхушку стека загружается заготовка УСВ, и в поле Δ АДРЕСА записывается разность между адресом этой заготовки и адресом МКС запускающей процедуры.
2. В УМС записывается адрес метки, расположенной выше заготовки УСВ.

Состояние стека до и после выполнения операции МС показано на Рис.37.

Если в верхушке стека находится адресная информация, ведущая через цепочку косвенных слов и сквозных меток к обычной метке, то операция МС сначала заменяет адресную информацию на обычную метку, а затем выполняет описанные выше действия. Если адресная информация не приводит к обычной метке - прерывание.

б) Операция ВХОД.

Пусть из вызывающей процедуры *Pn* уровня *n* происходит обращение к вызываемой процедуре *Qm* уровня *m.* Тогда:

1. По уровню, указанному в метке *m,* адрес которой содержится в УМС, устанавливается номер уровня в регистре Уровня.

2. По содержимому УМС и УС устанавливается поле РАЗМЕР в МКС вызываемой процедуры *Qm.*

3. Из метки вызываемой процедуры *Qm* выделяется поле АДРЕС, по которому находится МКС процедуры уровня *m-1,* в которую вложена вызываемая процедура *Qm.* В поле АДРЕС МКС процедуры *Qm* записывется адрес найденного МКС (формируется статическая цепочка).

4. По статической цепочке происходит коррекция базовых регистров, начиная с номера *m.* В базовый регистр номера записывается адрес МКС запускаемой процедуры, который берется из регистра УМС. По полю АДРЕС этого МКС находится МКС *m-1* уровня и записывается в базовый регистр номера *m-1.* Таким образом, производится заполнение всех базовых регистров от *m* до 0.

5. Устанавливается признак Ф в УСВ вызываемой процедуры *Qm,* показывающий, должна ли эта процедура поставлять значения.

6. Признак завершенности входа в процедуру 3 в УСВ процедуры *Qm* устанавливается в 1.

7. Управление передается в процедуру *Qm.* При этом в регистр базы записывается дескриптор программного сегмента, определяемый полем БАЗА метки *Qm,* а в счетчик команд записывается целое из поля *Nk.*

в) Операция ВОЗВРАТ.

Возврат из процедуры *Qm* в процедуру *Pn* происходит следующим образом:

1. По содержимому регистра уровня находится дескриптор стека процедуры *Q,* а следовательно, МКС и УСВ процедуры *Qm.*

2. Путем анализа признака Ф в УСВ процедуры *Qm* решается вопрос, оставлять ли значение в стеке.

3. По информации, хранящейся в поле Δ АДРЕСА УСВ процедуры *Qm* (динамическая цепочка), находится МКС процедуры *Pn.*

4. Найденный адрес МКС процедуры *Pn* записывается в базовый регистр по номеру, указанному в поле *LL* УСВ процедуры *Qm.* Этот же номер записывается в регистр номера уровня.

5. По статической цепочке, начало которой хранится в поле АДРЕС МКС процедуры *Pn,* производится коррекция базовых регистров, начиная с номера *n-1.*

6. Устанавливается УС на первую ячейку свободной памяти, имея в виду, что размер занятой памяти определяется содержимым поля РАЗМЕР МКС процедуры *Pn.*

7. По информации, хранящейся в УСВ процедуры *Qm,* управление передается в процедуру *Pn.* При этом по содержимому поля БАЗА в текущий регистр базы записывается дескриптор программного сегмента процедуры *Pn,* в счетчик команд записывается содержимое поля *Nk* - номер программного слога, с которого должно быть возобновлено выполнение процедуры *Pn.*

В дальнейшем с целью ускорения ВХОДа и ВОЗВРАТа в/из процедур, не изменяющих контекст (например, стандартные функции или процедуры, не использующие контекст вызывающей процедуры), введена однобайтовая команда ОМС - открытая Маркировка стека, которая сводит к минимуму работу со связующей информацией.

Процедура возврата анализирует поле 3 в МКС и в зависимости от его значения оставляет в вершине стека 0, I или 2 слова. Это поле заполняется при выполнении операции ВХОД. Введено 3 разновидности операции: ВХОД0, ВХОД1, ВХОД2, которые заносят в поле 3 в МКС соответствующее значение.

2.1.4. Средства обработки COMMON-блоков в ФОРТРАНЕ

В ФОРТРАНе передача параметров между процедурами может происходить не только через обычный механизм связи фактических и формальных параметров, но и через общие группы переменных, известных в той и другой процедуре.

Например, если в процедуре SUBROUTINE P объявлены COMMON-блоки A, B, C оператором

COMMON│ A│ *x,y,z*│ B│ *u,v*│ C| *d,e,f,g*

и процедура *P* вызывает процедуру *Q,* в которой объявлены COMMON-блоки F, C, A, E оператором

COMMON│ F| *l ,m,n* │ C│ *h,i,j,k*│ A│ *u,Y ,W*│ E│ *x,y,*

то значения переменных, расположенных в COMMON-блоках A и C, передаются в процедуру *Q*. (При этом эти значения переименовываются: значение переменной *x* становится значением переменной *u* и так далее.)

Процедура *Q* может изменить значения переменных в COMMON-блоках, и измененные значения в COMMON-блоках A и C под первоначальными именами будут известны процедуре *P*. Поэтому переменные в COMMON-блоках нет смысла помещать в стек при вызове процедуры вместе с формальными и локальными параметрами, так как при выходе из процедуры память, отводимая для этих параметров, возвращается в область свободной памяти.

COMMON-блоки помещаются в память, отводимую для массивов, и обращение к COMMON-блокам осуществляется через описывающие их дескрипторы. Для уменьшения времени обращения к переменным в COMMON-блоках в системе команд предусмотрены средства, позволяющие помещать дескрипторы COMMON-блоков в базовые регистры. В этом случае обращение к переменным в COMMON-блоках может осуществляться адресной парой, так же как и обращение к переменным в стеке. При обращении к процедуре P, имеющей COMMON-блоки A, B, C, в базовые регистры последовательно загружаются дескрипторы, описывающие эти COMMON-блоки. Таким образом, если дескриптор, описывающий блок A, загружается в базовый регистр с номером *m,* то процедура *P* будет работать на *m+3* уровне.

При вызове процедуры *Q* производится загрузка аналогичным образом дескрипторов, описывающих COMMON-блоки F, C, A, E. При этом процедура *Q* будет работать на уровне *m+4,* то есть уровень процедуры определяется количеством имеющихся в ней COMMON-блоков. Выполняется это следующим образом. Перед входом в процедуру формируется пачка дескрипторов, описывающих все COMMON-блоки вызываемой процедуры (Рис. 44).

Эта пачка оканчивается косвенным словом, содержащим адрес МКС той процедуры, в которую вложена вызываемая процедура Q. (На Рис. 45 для ФОРТРАН косвенные слова всех пачек указывают на МКС процедуры уровня *m-1)*. При помощи операции УФМ формируется метка, в поле АДРЕС которой записан адрес первого дескриптора из сформированной пачки. Затем выполняется операция МС и операция загрузки формальных ячеек.

При коррекции статической цепочки во время выполнения операции ВХОД автоматически выполняется следующий алгоритм:

1) если звено в статической цепочке указывает на МКС, то происходит переход по этому МКС, и из его содержимого формируется дескриптор в очередном базовом регистре (обычный случай);

2) если поле АДРЕС МКС указывает на дескриптор, то происходит загрузка этого дескриптора в очередной базовый регистр и анализ следующей за дескриптором ячейки.

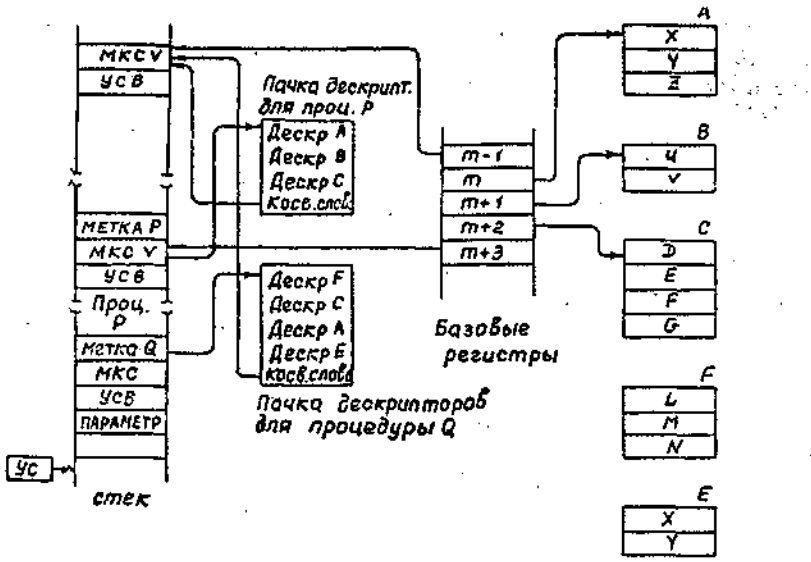


Рис.44. Состояние стека до входа в процедуру *Q*.

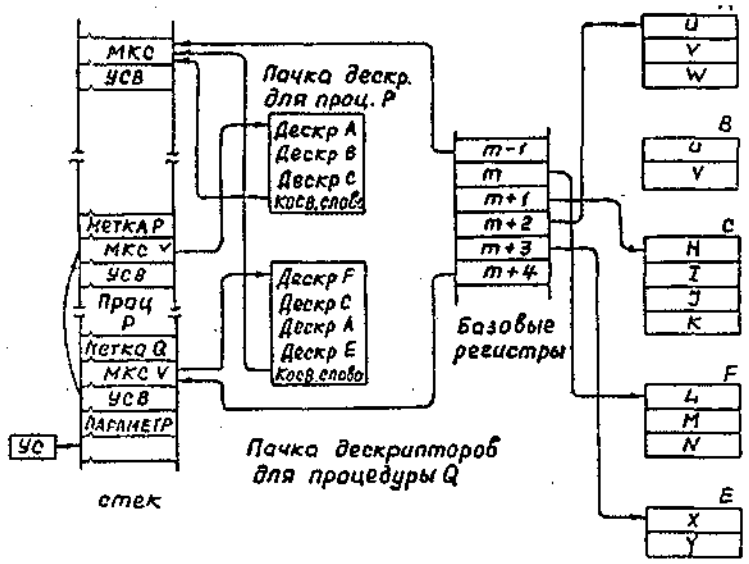


Рис.45. Состояние стека после входа в процедуру *Q*.

В этой ячейке должны содержаться либо дескриптор, либо косвенное слово. Если там содержится дескриптор, то пункт 2 повторяется сначала. Если содержится косвенное слово, то происходит переход по этому косвенному слову, и описанный алгоритм повторяется.

После окончания операции ВХОД все дескрипторы из пачки будут записаны в базовые регистры.

При выходе из процедуры *Q* по динамической цепочке находится МКС процедуры *P,* и снова восстанавливаются базовые регистры в те состояния, которые они имели до вызова процедуры *Q*.

В дальнейшем несколько изменена структура информации, подготавливаемой для загрузки дескрипторов COMMON-блоков в базовые регистры. Поле адреса в МКС указывает не непосредственно на пачку дескрипторов COMMON-блоков, а на дескриптор, описывающий эту пачку.

2.2. Арифметические операции, операции отношения и логические операции

2.2.1. Арифметические операции

Арифметические операции производятся над одной или двумя величинами, содержащимися в верхушке стека. Как правило, операнды являются целыми или вещественными, причем, если операнды различного типа (целое и вещественное), то перед операцией производится автоматическое преобразование целого в вещественное. Если операнды имеют различные форматы, то перед операцией производится автоматическое "выравнивание длин" по наибольшей длине. Таким образом, собственно операция производится над двумя целыми или двумя вещественными одного формата - этот формат, как правило, и присваивается результату, за исключением случая переполнения (Табл. 2.1). Исходные числа убираются из стека (с соответствующим смещением указателя), а затем результат помещается в верхушку стека.

Одноместные арифметические операции берут операнд из верхушки стека и заменяют его на результат.

Кроме целых и вещественных в качестве операндов арифметических операций допускаются правые битовые наборы и индексные слова. Битовый набор перед началом операции автоматически преобразуется в целое без знака формата 64 независимо от его полноты. В случае индексного слова перед началом операции производится автоматическое выделение поля текущего значения индекса ИНДЕК и преобразование его в целое.

Таблица 2.1. Преобразование типов и форматов при арифметических операциях.

	Ц32	В32	Ц64	В64	В128
Ц32	Ц32 Ц64*	В32 В128**	Ц64 В64**	В64 В128**	В128 ***
В32	В32 В128**	В32 В128**	В64 В128**	В64 В128**	В128 ***
Ц64	Ц64 В64**	В64 В128**	Ц64 В64**	В64 В128**	В128 ***
В64	В64 В128**	В64 В128**	В64 В128**	В64 В128**	В128 ***
В128	В128 ***	В128 ***	В128 ***	В128 ***	В128 ***

* Происходит преобразование длины, вырабатывается сигнал прерывания - преобразование длины.
** Происходит преобразование типа, вырабатывается сигнал прерывания - преобразование типа. *** Результат неверный, вырабатывается сигнал прерывания - переполнение. Происходит преобразование в формат 128. Вырабатывается сигнал прерывания - преобразование в формат 128.

Появление в качестве операндов элементов остальных типов приводит к прерыванию. Бит, байт и цифра в арифметических операциях не допускаются. Все арифметические операции можно разбить на следующие 4 группы:

а) Обычные операции.

Поскольку арифметические операции могут выполняться над значениями, представленными разными форматами и типами, форматы и типы результатов могут быть самыми разнообразными. При выборе формата и типа результата принимались во внимание следующие соображения:

1. Целые рассматриваются как частный случай вещественных. Поэтому имеет смысл сохранить максимум информации о результате. Иными словами, если в операциях над целыми числами получается целый результат, то желательно его сохранить целым.

2. Формат результата система пытается сохранить таким, каким был больший формат у операндов, участвующих в операции (то есть, если форматы участвующих в операции операндов равны Ф1 и Ф2, то формат результата Ф по возможности равен Ф = max{Ф1,Ф2}). Отметим, что из-за аппаратурных соображений было принято нецелесообразным преобразование формата результата к меньшему, когда это оказывается возможным.

3. Если формат результата не укладывается в формат Ф = max (Ф1,Ф2), то происходит переход к следующему большему формату с соответствующим прерыванием П1.

4. Если полученное целое не укладывается в формат 64, то происходит преобразование целого в вещественное формата 64 с соответствующим округлением и выдачей прерывания П2.

5. Если полученное вещественное не укладывается в формат 128, то выдается прерывание ПЗ.

Следует заметить, что описанная стратегия, при которой изменение формата может происходить только в сторону увеличения, приводит к некоторому увеличению среднего формата промежуточных результатов. Однако при записи этих результатов в память имеются средства для сокращения форматов (см. команды записи).

Руководствуясь приведенными выше соображениями, получим для операций "+", "-", "*"таблицу преобразования типов и форматов (Табл. 2.1).

Для операции ":" справедлива эта же таблица с единственным исключением: при делении целых чисел M:N результат будет целым лишь при M{mod N}= 0.

Операция ХУД (умножить с удвоенной точностью) отличается от обычной операции тем, что для вещественных операндов получается результат вдвое большего формата, чем при обычном умножении "*".

Если оба операнда целочисленные, то удвоение длины не происходит, если результат укладывается в формате Ф = max (Ф1,Ф2).

Если один из операндов имеет формат 128, то формат результата также будет 128.

б) Операции целочисленной арифметики.

Эти операции отличаются от обычных тем, что работают только с целочисленными операндами и правыми битовыми наборами (автоматически преобразуются в целые) и выдают целочисленный результат.

Если один из операндов не удовлетворяет этим условиям, то - прерывание.

Если результат не умещается в формат 64 - прерывание.

в) Операции с контролем значимости.

В результате сложения и вычитания двух вещественных чисел может получиться результат, у которого все разряды мантиссы равны нулю. Вообще говоря,

это имеет место, если результат *Р* по модулю меньше $16^{П-N}$, где П - порядок результат, N- количество шестнадцатиричных цифр в мантиссе. После обычных операций "+" и "-" этот результат превращается в нормализованный нуль, но при этом теряется информация о точности, с которой этот результат равен нулю.

Для контроля точности представления вещественного нуля вводятся различные способы представления вещественного нуля, а именно, возможны различные коды порядка при нулевой мантиссе. Число вида $16^{П}$. О рассматривается в машине не как "точный нуль", а как число, меньшее по модулю чем $16^{П-N}$ - ненормализованный нуль.

Для работы с ненормализованными нулями введены операции "+" и "-" с контролем значимости +3 и -3 соответственно. Если в результате этих операций в мантиссе получились все нули, то результат не нормализуется и сохраняется порядок, который получился в результате операции. В дальнейшем этот результат используется только этими операциями, а также операциями отношения (раздел 2.2.2). Если этим операциям поставлен в качестве операнда ненормализованный нуль, который представлен с меньшей точностью, чем второй операнд, то, поскольку сложение или вычитание этих операндов становится бессмысленным, выдается прерывание.

В обычных операциях ("+" и "-") ненормализованный нуль воспринимается как нормализованный. Операции "*" и ":" изменяют порядок ненормализованного нуля. Переполнение порядков не фиксируется. (В аппаратуру введена искусственная защита от переполнения из-за ненормализованных нулей).

г) Одноместные операции.

Введены две одноместные операции:

ИЗН - изменение знака,

ВЗН - взять знак.

Операция ИЗН изменяет знак у операнда в верхушке стека. Тип и формат операнда сохраняются. Нормализованный нуль преобразуется в нормализованный нуль.

Операция ВЗН помещает в верхушку стека 32-разрядное число, равное соответственно 1, 0, - 1 в зависимости от того, является ли операнд положительным, нулем или отрицательным.

2.2.2. Операции отношения

Введены следующие операции отношения:

= - равно, ≠ - не равно,

< - меньше,

≤ - меньше либо равно,

> - больше,

≥ - больше либо равно.

Операции проверяют отношение между двумя операндами в верхушке стека. Результат операций двоичный (1 - при выполнении отношения, иначе - 0). Формат операции - 1 байт.

Операции допускаются:

1) между целыми, вещественными и правыми битовыми наборами (как в арифметических операциях),

2) между одноименными одинаково ориентированными наборами (при этом меньший набор дополняется нулями до большего),

3) между набором и величиной такого же типа (при этом величина расширяется до соответствующего набора),

4) между однотипными величинами.

Сравнение двух вещественных в случае равенства может сопровождаться сигналом потери значимости. Сравнение целого нуля с вещественным нулем также может сопровождаться сигналом потери значимости.

Введены 2 операции логического отношения:

Л=- логически равно,

Л≠ - логически не равно.

Операндами могут быть любые элементы памяти.

Первая операция производит проверку на полное совпадение (включая тип и формат). Результат операций - двоичный.

Вторая операция эквивалентна последовательности операций Л=, ⌉.

.

В дальнейшем вместо правых и левых наборов оставлен только один тип НАБОР, который рассматривается как целое без знака.

2.2.3. Логические операции

Введены следующие логические операции:

1) ⌉-отрицание,
2) v- логическое сложение,
3) ^- логическое умножение,
4) ≡- эквивалентность.

Все эти операции являются однобайтовыми.

Первая операция является одноместной. Она может выполняться только над битовыми наборами и двоичными элементами. Результатом операции является величина того же типа, но все единичные элементы заменены нулевыми и наоборот.

Все двухместные логические операции могут выполняться только над одноименными битовыми наборами и битовыми величинами. В случае неполных наборов более короткий дополняется нулями. Результатом операции является величина того же типа, что и тип более длинного операнда.

2.3. Операции над наборами

Поскольку непосредственная адресация к элементам набора невозможна, для обработки элементов набора введены операции над наборами. Следует подчеркнуть, что операции над наборами применяются не только к наборам, но и к операндам любого типа (к дескрипторам и косвенным словам эти операции применяются только для изменения содержимого поля ЗАЩИТА в сторону усиления защиты. Это ограничение введено для того, чтобы пользователь не имел средств нарушить защиту памяти). При этом любой операнд, отличный от набора, трактуется как правый битовый набор. Операнды формата, меньшего 64 разрядов, трактуются как неполные правые битовые наборы. Исключение составляют операнды типа вещественное формата 32, которые трактуются как левые битовые наборы, и операнды типа цифра и байт, которые трактуются соответственное как неполные правые десятичные и байтовые наборы, состоящие из одного элемента. Таким образом, операции над наборами выполняются над операндами, преобразованными к формату 64.

Операции над наборами применяются к операнду, расположенному в верхушке стека. Операнды формата 128 в этих операциях не допускаются. Операции над наборами позволяют выделять, преобразовывать, вставлять подполя в операнде, расположенном в верхушке стека. В большинстве этих операций используется описание поднаборов, которое состоит из:

- номера позиции в операнде,
- количества элементов в операнде.

Описание поднабора может задаваться непосредственно в команде - статический вариант. В этом случае номер позиции и количество элементов задаются целыми без знака. На каждое целое отводится по одному байту. В конце названия операции добавляется буква Н.

Описание поднабора может также задаваться в верхушке стеки - динамический вариант. В конце названия операции добавляется буква Д. В этом случае поднабор может описываться:

1) одним операндом, представляемым интервалом, в котором задается и номер позиции, и количество элементов,

2) двумя операндами в верхушке стека, каждый из которых может быть:

- целым без знака,
- правым битовым набором (автоматически преобразуется в целое),
- индексным словом (берется текущее значение индекса).

Операнды загружаются в стек в том же порядке, в каком они следуют в командном потоке. Таким образом, в верхушке стека должно быть количество элементов, а еще выше - номер позиции (Рис.46).

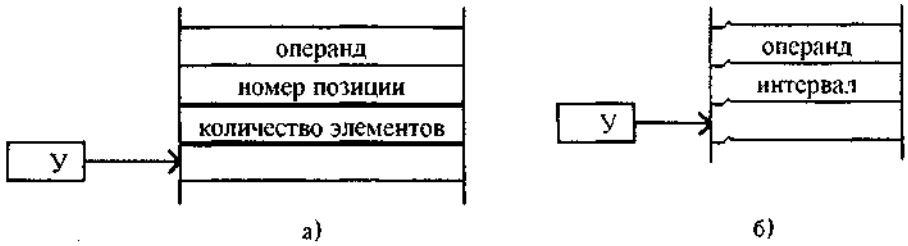


Рис.46. Состояние верхушки стека перед выполнением операций ;над наборами (динамический вариант)

а Поднабор определяется двумя операндами.

б Поднабор определяется одним операндом - интервалом.

Динамические операции являются однобайтовыми. Во всех случаях, когда в описании поднабора встречаются не имеющиеся в операнде номере элементов, - прерывание. В системах команд имеется набор операций, позволяющих взять элемент набора, взять поднабор, вставить поднабор, сосчитать число единиц набора и так далее, обеспечивающих, в первую очередь, обработку текстовой информации.

2.4. Обработка массивов

Как уже указывалось, область памяти, отводимая под массив, описывается дескриптором. Массивы могут размещаться как в стеке, так и вне его. Обращение к элементам массива производится при помощи операций индексации. Для организации циклической обработки элементов массива введена трехбайтовая операция КЦ - конец цикла (Рис.47).

Рис.47. Вид операции КЦ - конец цикла.

Имеется 4 варианта этой операции, так как:

1) в верхушке стека может находиться либо индексное слово, либо адресная информация, указывающая непосредственно на индексное слово;

2) при выходе из цикла информация об индексном слове может оставаться или вычеркиваться из верхушки стека.

Мнемоника этих вариантов следующая:

КЦИС - в верхушке стека находится индексное слово, и после выхода из цикла оно сохраняется.

КЦИВ - в верхушке стека находится индексное слово, и после выхода из цикла *оно* вычеркивается.

КЦАС - в верхушке стека находится адрес индексного слова. Индексное слово берется по этому адресу одношагово. После выхода из цикла этот адрес сохраняется.

КЦАБ - в верхушке стека находится адрес индексного слова без лишней косвенности, и после выхода из цикла этот адрес вычеркивается.

Выполняются эти операции следующим образом. Производится модификация поля текущее значения ИНДТЕК индексного слова шагом Δ и сравнение с предельным значением ИНДМАКС. В случае, если (ИНДМАКС - ИНДТЕК) Δ < 0, происходит окончание цикла и переход к следующей команде. В противном случае производится передача целого Числа, указанного в команде КЦ, на счетчик команд и модифицированное индексное слово записывается на прежнее место.

Например, пусть требуется поместить в массив, описываемый дескриптором ДЗ, первые *m* компонентов суммы нечетных компонентов двух *n*-мерных векторов a_i и b_t *(i= 1,2,..... , n; 2m-1≤ n),* описываемых соответственно дескрипторами Д1 и Д2 (Рис.48).

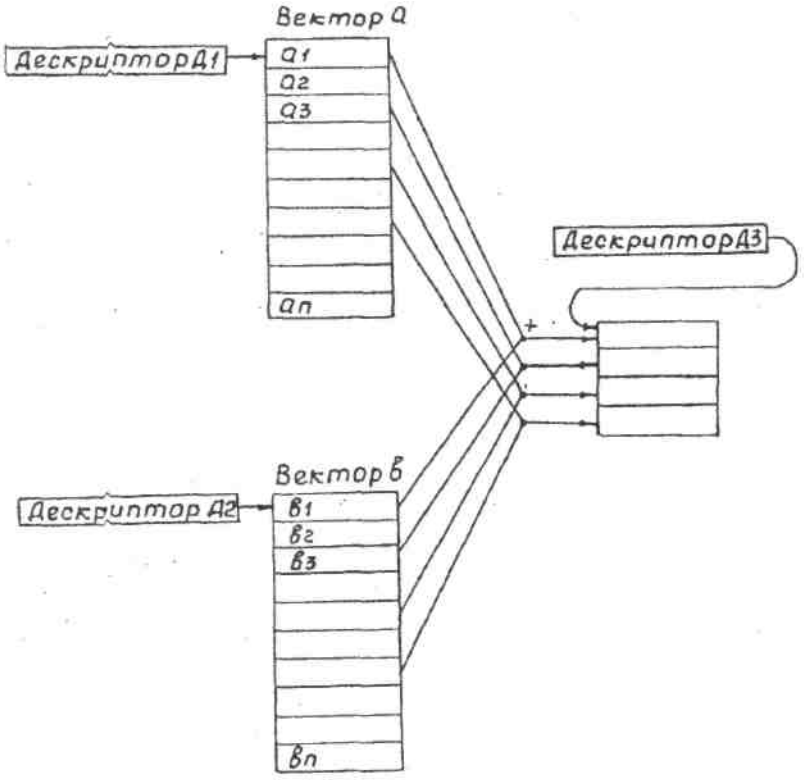


Рис.48. Вид массивов при сложении нечетных компонентов векторов.

Это можно выполнить следующим образом (для наглядности неоптимальным). Пусть сформировано три индексных слова ИС1, ИС2 и ИС3, в которых поля Δ, ИНДМАКС, ИНДТЕК первоначально соответственно равны 2, *n,* 0; 2, *n,* 0; и 1, *m, 0.* Программа, выполняющая эту задачу, имеет следующий вид:

```
НАЧ:    ВЗ      Д1
        ЗА      ИС1
        ИНДЗ+
        ВЗ      Д2
        ЗА      ИС2
        ИНДЗ +
        +
        ВЗ      Д3
        ЗА      ИС3
        ИНД
        ЗП      без сохранения
        ЗА      ИС3
        КЦ      НАЧ
```

Здесь в поле НАЧ последней команды записан номер байта первой команды, помеченной НАЧ.

Групповые операции над массивами

В групповые операции над массивами включены только операции пересылки, сравнения и редактирования. Эти операции можно разбить на 5 групп:

1) Операции пересылки:
   ПС - пересылка слов.
2) Операции пересылки с проверкой отношения:
   ПСПО - пересылка с проверкой отношения.

Имеется 7 модификаций этой операции: 1 операция безусловной пересылки, и 6 операций различаются проверяемым отношением (=, ≠, <, ≤, >, ≥).

3) Операции пересылки по шкале:
   ПСШП - пересылка по шкале прямая,
   ПСШД - пересылка по шкале дополнительная.
4) Операция сравнения массивов:
   СРМ - сравнение массивов.

Имеется 6 модификаций этой операции в зависимости от применяемого отношения (=, ≠, <, ≤, >, ≥).

5) Операции пересылки с переводом и редактированием:
   ПСП - пересылка с переводом,
   ТР - табличное редактирование,
   ОР - одиночное редактирование,
   ОРОС - одиночное редактирование с одним указателем.

В операциях обрабатываются один или несколько массивов. С каждым массивом связывается указатель массива. Указатель массива состоит из двух операндов: индекс (целое) и дескриптор. (В стеке они помещаются в таком порядке: снизу целое, выше дескриптор). Кроме того, в верхушке стека указывается количество элементов массива, участвующих в операции (Рис.49).

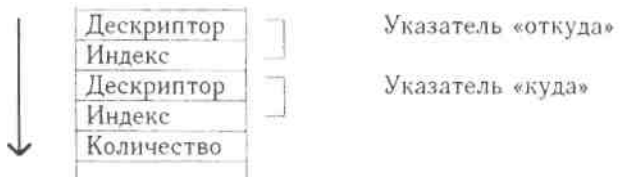


Рис. 49. Состояние верхушки стека перед выполнением операций над массивами.

1Если указанное в операции количество элементов меньше количества элементов, которое можно переслать из массива-источника, то операция выполняется над указанным количеством элементов.

2. Если указанное в операции количество элементов больше количества элементов, которое можно переслать из массива-источника, то операция выполняется над всеми указанными элементами массива-источника.

3. Если указанное количество элементов в массиве-назначения меньше участвующего в операции количества элементов по пунктам 1 и 2, то после выполнения операции над всеми элементами массива-назначения - прерывание.

Если вместо дескриптора массива-источника поставлен операнд - не дескриптор, то значение этой величины рассылается по массиву-записи.

Массив считывания может быть заменен набором, в этом случае в стеке помещается сам набор без указателя массива, и этот набор рассматривается как массив элементов набора. Перемещение указателя массива при этом происходит по элементам набора.

Если вместо дескриптора массива записи поставлен операнд - не дескриптор, то элементы массива считывания никуда не переписываются, в операциях пересылки это приводит к считыванию "в воздух".

Независимо от того, используется ли указатель массива, набор либо другой операнд, количество операндов в верхушке стека перед выполнением операции должно быть постоянно. Если указатель массива не используется, то содержимое второго слова, отведенного под указатель, во внимание не принимается.

Каждая операция имеет два варианта: с сохранением и без сохранения. В первом случае в стеке сохраняются все операнды, бывшие там перед операцией (указатели массивов, счетчик и другие) в своем конечном состоянии (например, указатель перемещен на *n* элементов, из счетчика вычтено *n* и тому подобное). Во втором случае все операнды, находившиеся в стеке перед операцией, вычеркиваются из стека путем соответствующего перемещения указателя стека. Мнемоника этих вариантов такая же, как в операциях записи и конца цикла. Буква С на конце означает вариант с сохранением, буква Б -без сохранения.

Рассмотрим особенности операций каждой группы.

Операция ПС - пересылки слов, применяется для быстрого перемещения разноформатной информации из одной области памяти в другую. При пересылке МКС выдается прерывание, так как перемещение МКС из одной области в другую может привести к нарушению защиты памяти.

Оба дескриптора, участвующие в операции, могут индексироваться любым из описанных способов, но получающиеся МА должны иметь нули в 6 младших разрядах, то есть указывать на начало слова.

В указателе "откуда" вместо дескриптора может стоять некоторое слово. В этом случае это слово рассылается в массив "куда".

В следующих 3-х группах операций над массивами участвует глобальная переменная - бит отношения. Если операция имеет "естественный конец", то есть оканчивается по причинам, описанным выше, то бит отношения принимает нулевое значение. Если причина окончания операции - невыполнение отношения, то бит отношения устанавливается в единицу. В системе команд имеется операция, загружающая этот бит отношения в стек.

В операциях ПСПО - пересылка с проверкой отношения в верхушку стека помещается эталон, с которым сравниваются пересылаемые из массива-источника элементы массива (Рис.50).

Рис. 50. Состояние верхушки стека перед выполнение операций пересылки с проверкой отношения.

Если выполняется операция безусловной пересылки, то эталон в верхушку стека не помещается.

В операциях пересылки по шкале в верхушку стека помещается дескриптор битового массива - шкалы. Массив-источник и массив назначения должны быть байтовыми массивами или байтовыми наборами. Выполняются операции следующим образом. Дескриптор шкалы индексируется индексом, равным И, где *И* - целое, равное коду очередного байта, взятого из массива источника Получившийся МА используется для считывания бита из шкалы. Условием окончания пересылки является нулевое значение бита шкалы в случае прямой пересылки и единичное - в случае дополнительной пересылки. Эта операция обычно используется для пересылки куска из текста, хранящегося в байтовом массиве-источнике до тех пор, пока не встретятся некоторые наперед заданные символы (например, пробелы, знаки препинания, ограничители и тому подобное), определяемые шкалой.

В операциях сравнения массивов не происходит никакой пересылки элементов массива. В этом отношении оба массива, участвующие в операций, совершенно равноправны. "Естественный конец" операции будет при этом либо при исчерпании количества элементов, либо при исчерпании одного из массивов.

Операции выполняются следующим образом. Происходит поэлементная проверка заданного отношения. Операция заканчивается либо при первом невыполнении отношения, либо "естественным концом". Бит отношения устанавливается в единицу, если операция заканчивается из-за невыполнения отношения. Вместо указателей массивов могут быть заданы в любой комбинации наборы и одиночные операнды.

Операция ПСП - пересылка с переводом, используется для пересылки байтовых массивов с перекодировкой (Рис.51). В верхушке стека помещается указатель байтового массива "Таблица".

Выполняется операция следующим образом. Дескриптор таблицы индексируется так же, как и в операциях пересылки на шкале. Полученный из таблицы байт отсылается в массив назначения. Обычно эта операция используется для перекодировки текстов из различных систем кодирования в единое внутреннее представление.

Операция ТР - табличное редактирование, используется для редактирования текстов по заданной таблице микроопераций.

Дескриптор на эту таблицу помещается в верхушку стека. В этой операции не задается количество участвующих в операции элементов массива.

Операция ОР - одиночное редактирование, отличается от операции табличного редактирования лишь тем, что выполняется только одна микрооперация, которая

следует в командном потоке за операцией ОР. Вместо указателя таблицы в верхушке стека находится целое, используемое при работе микрооперации.

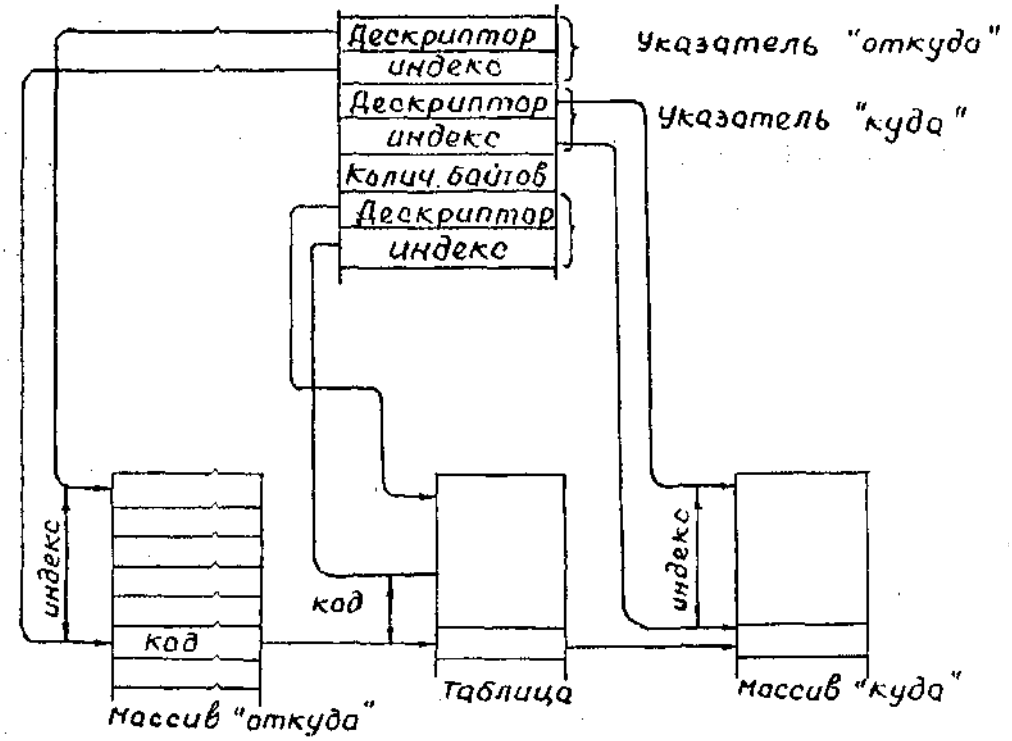


Рис.51. Выполнение операции пересылки с переводом.

Операция ОРО - одиночное редактирование с одним указателем, отличается от предыдущей лишь тем, что она использует в стеке только один указатель (в качестве указателя "куда" и "откуда"). Эта операция имеет только вариант с сохранением.

В дальнейшем несколько изменены условия окончания операций. С помощью операнда КОЛИЧЕСТВО можно задать любой из двух режимов управления:

а) управление от данных (КОЛИЧЕСТВО <0). Операция заканчивается (если не предписано иначе) исчерпанием массива-источника или массива-назначения.

б) управление от счетчика (КОЛИЧЕСТВО > 0). Операция заканчивается (если не предписано иначе) исчерпанием, количества, массива-источника или Массива-назначения.

Если конец операции произошел исчерпанием массива-источника, устанавливается в 1 триггер ИИ; если исчерпанием массива-назначения, устанавливается в 1 триггер ПЕРЕПОЛНЕНИЕ. Если операция заканчивается исчерпанием КОЛИЧЕСТВО, то триггера ИИ и ПЕРЕПОЛНЕНИЕ устанавливаются в 0.

Принципы работы операционной системы МВК «Эльбрус»

В.С.Бурцев, В.П.Торчигин

Введение

Операционная система (ОС) МВК "Эльбрус" выполняет все функции ОС современных многопроцессорных систем, в которых в процессе выполнения задач распределяется оперативная память и другие физические ресурсы системы, такие как процессоры, внешняя память, устройства ввода-вывода информации и связывающие их каналы.

Особенностью организации ОС МВК "Эльбрус" является процедурная обработка информации, которая эффективно реализуется за счет описанной выше системы команд и аппаратной поддержки средств синхронизации наиболее часто встречающихся конструкций вычислительных процессов.

Рассмотрим наиболее принципиальные функции ОС, существенно отличающих ОС МВК "Эльбрус" от других аналогичных зарубежных систем. К ним относятся:

- средства синхронизации параллельных процессов;
- средства синхронизации центральных процессоров;
- средства организации обменов с внешними устройствами;
- организация системы файлов МВК "Эльбрус";
- принципы распределения ресурсов.

1. Средства для синхронизации параллельных процессов

Синхронизация параллельных процессов осуществляется с помощью известной техники семафоров [1,10]. Семафор - это тип данных в системе команд МВК, который имеет свой тег [9,10]. Непривилегированный пользователь не может никак изменить содержимое семафора. Он может лишь подавать семафор по имени в качестве параметра в процедуры синхронизации (семафоры не могут копироваться и каждый семафор существует лишь в одном экземпляре, однако на каждый семафор может быть установлено несколько ссылок). ОС МВК поддерживает 2 типа синхронизации: событийную и ресурсную.

При событийной синхронизации используются процедуры ЖДАТЬ(имя смф), ОТКРЫТЬ (имя смф) и ПРОПУСТИТЬ(имя смф) [2]. Приставка имя в

параметре процедур свидетельствует о том, что ,в процедуру передается ссылка на семафор, а не сам семафор, то есть параметр передается по имени, а не по значению. Если параметр в процедуре ЖДАТЬ ссылается на открытый семафор, то процедура ЖДАТЬ заканчивается и выполняются операторы, следующие за процедурой ЖДАТЬ. Если же семафор закрыт, то стек, на котором выполняется процедура ЖДАТЬ, снимается с ЦП и ставится в очередь к закрытому семафору. Заметим, что процедура ЖДАТЬ может выполняться над рассматриваемым семафором на различных стеках, относящихся к одной задаче. В результате, к этому семафору может возникнуть очередь стеков, ждущих определенного события, после которого произойдет открытие семафора. Такая ситуация показана на Рис.1, где после выполнения процедуры ЖДАТЬ на стеке 3 в очередь к семафору добавляется этот стек.

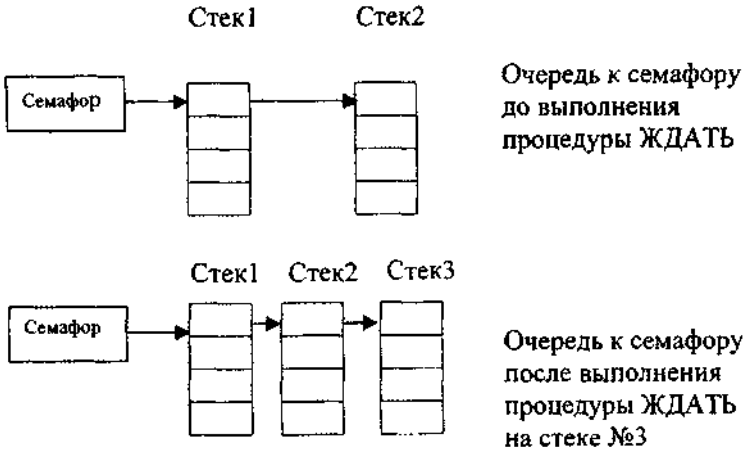


Рис.1. Модификация очереди к семафору после выполнения операции ЖДАТЬ.

Когда на некотором активном стеке происходит событие, после которого семафор может быть открыт, этот стек вызывает процедуру ОТКРЫТЬ. Эта процедура переводит семафор в открытое состояние. При этом вся очередь стеков к семафору переносится в очередь готовых работ (Рис.2). Место в этой очереди для каждого стека определяется в соответствии с его приоритетом. В дальнейшем, когда переведенные в очередь готовых работ стеки станут активными, на ЦП будут выполняться операторы, следующие за процедурой ЖДАТЬ. В том случае, если при наступлении ожидаемого события необходимо активизировать стеки, ожидающие этого события, но после этого оставить семафор закрытым, выполняется процедура ПРОПУСТИТЬ.

При ресурсной синхронизации используются процедуры ЗАКРЫТЬ(имя смф), ОТКРЫТЬ (имя смф). Если семафор открыт, то процедура ЗАКРЫТЬ закрывает этот семафор и на этом ее действия заканчиваются. Если же семафор уже закрыт, то процедура ЗАКРЫТЬ, также как и процедура ЖДАТЬ, снимает стек, на котором она выполнялась с ЦП, и ставит его в очередь к закрытому семафору. Отличие ее от процедуры ЖДАТЬ состоит в том, что при открытии семафора, после того как стек из очереди готовых работ попадет на ЦП, выполняется не оператор, следующий за процедурой ЗАКРЫТЬ, а процедура ЗАКРЫТЬ, то есть делается очередная попытка закрыть семафор. Эта попытка снова может быть неудачной, если после открытия семафора его успел закрыть другой стек, который раньше рассматриваемого выполнил процедуру ЗАКРЫТЬ над открытым семафором.

Рис.2. Модификация очередей стеков процедурой ОТКРЫТЬ.

На Рис.3 показана ситуация, при которой после открытия семафора стек 3, имеющий более высокий приоритет и поэтому раньше стеков 1 и 2 получающий ЦП, успел закрыть семафор и получил доступ к общим данным. При этом стеки 1 и 2 снова наткнулись на закрытый семафор при повторной попытке его закрыть и снова оказались в очереди к закрытому семафору. Таким образом для любого стека имеется гарантия, что после успешного выполнения процедуры ЗАКРЫТЬ семафор оказывается всегда закрытым. Такая ситуация необходима для работы с общими данными. Действительно, работа в общими данными происходит в критической секции [1], ограниченной синхронизирующими скобками, в качестве которых используются процедуры ЗАКРЫТЬ и ОТКРЫТЬ. Внутри критической секции общие данные, которые охраняются данным семафором, монополизированы процессом, выполняющим критическую секцию.

Нетрудно убедиться, что описанная техника семафоров предоставляет пользователю инструмент для синхронизации параллельных процессов. Однако, на пользователя возлагается ответственность за правильностью использования этой техники. Например, если какой-то процесс открывает семафор, который им не закрывался, то процесс, который закрывал семафор, теряет монополию на работу с данными, охраняемыми этим семафором. Кроме того, эта техника не гарантирует от возникновения тупиков.

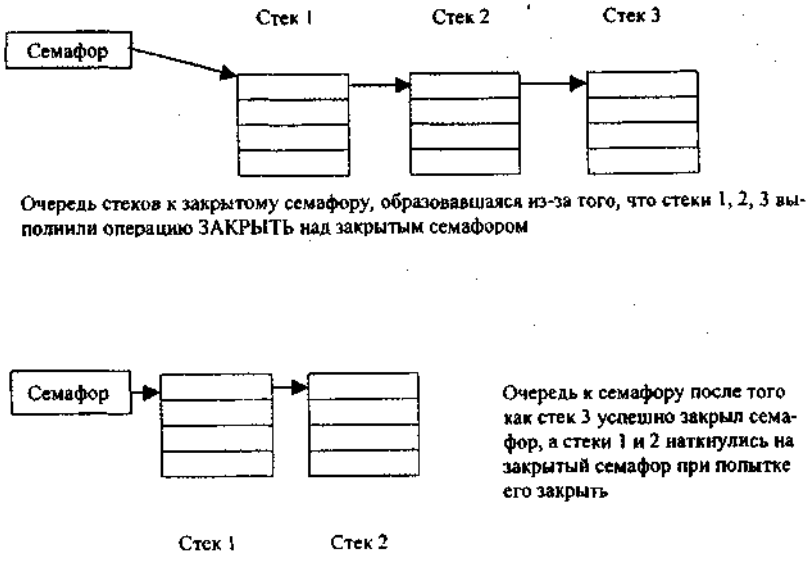


Рис.3. Состояния очереди стеков к семафору после выполнения процедуры ОТКРЫТЬ.

Действительно, если процесс 1 закрыл семафор 1 и пытается закрыть семафор 2, который был уже закрыт процессом 2, то процесс 1 окажется в очереди к семафору 2. Если после этого процесс 2 пытается закрыть семафор 1, то он окажется в очереди к семафору 1. В результате оба стека, которые должны открывать семафоры 1 и 2, оказываются неактивными и поэтому эти семафоры никогда не будут открыты, а стеки активизированы. Ответственность за недопущение подобных ситуаций полностью возложена на пользователя. ОС может лишь обнаружить сам факт наличия тупика, контролируя время, в течение которого стек находится в неактивном состоянии. .

Кроме описанных выше процедур синхронизации, доступных непривилегированному пользователю, для привилегированных пользователей имеются дополнительно две операции ЖУЖ(имя смф) и ОТКСЕМ(имя смф). По своему назначению эти операции аналогичны процедурам ЗАКРЫТЬ, ОТКРЫТЬ, но выполняются гораздо быстрее. Однако это налагает ограничения на выполняемые внутри критической секции действия. Должна быть гарантия, что эти действия относительно кратковременны. На операторы в критической секции налагается условие, что при ее выполнении не возникнут прерывания (следовательно, нельзя выполнять обмен с внешними носителями, обращаться по математическому адресу и так далее). Более того, сама критическая секция выполняется с закрытыми прерываниями. При невыполнении этих условий может нарушиться работа всей ОС. Поскольку непривилегированный пользователь не должен иметь возможность нарушить работу ОС ни при каких условиях, то ему указанные возможности не предоставляются. Операции ЖУЖ и ОТКСЕМ широко используются в процедурах самой ОС для обращения к общим данным, расположенным в системных таблицах, списках и тому подобное, и могут рассматриваться как открывающая и закрывающая скобки критической секции, соответственно.

Если в операцию ЖУЖ подана ссылка на открытый семафор, то он закрывается. При этом анализ состояния семафора и его закрытие представляют собой неделимое действие. Между ними не могут вклиниться действия по

анализу состояния и закрытию семафора, выполняемые на другом ЦП. Это приводит к тому, что каждая операция ЖУЖ работает с семафором, находящемся в одном из двух состояний: открытом или закрытом. Семафоры в промежуточном состоянии, когда они еще открыты, но принято решение об их закрытии, отсутствуют. Неделимость анализа и закрытия семафора реализована на аппаратном уровне.

Если в операцию ЖУЖ подана ссылка на закрытый семафор, то проверка состояния семафора повторяется периодически в цикле, то есть семафор опрашивается с достаточно высокой частотой. Таким образом, в отличие от процедур ЖДАТЬ и ЗАКРЫТЬ, которые при закрытом семафоре деактивизируют стек, операция ЖУЖ выполняется на ЦП до тех пор, пока какой-либо другой активный процесс с другого ЦП не откроет семафор. Разумеется, с целью экономии времени ЦП такое «жужжание» должно быть кратковременным. Обычно это время равно времени считывания, анализа, модификации и записи некоторых общих данных и меньше времени выполнения нескольких десятков машинных команд. Операция ОТКСЕМ открывает закрытый семафор, давая тем самым возможность процессу, который в это время выполнял операцию ЖУЖ над этим семафором, его закрыть.

Легко понять, что в операциях ЖУЖ, ОТКСЕМ и в операциях ЗАКРЫТЬ, ОТКРЫТЬ в одно и то же время не может использоваться один и тот же семафор. Указанные выше средства синхронизации параллельных процессов активно используются самой ОС, процедуры которой могут одновременно работать на многих стеках. Действительно, каждый процесс пользователя может вызывать процедуры ОС, которые обычно выполняются на стеке вызывающего их процесса. Таким образом, на многих ЦП могут одновременно выполняться одни и те же или разные процедуры ОС, работающие с одними и теми же данными. В ОС имеется несколько системных семафоров, охраняющих различные системные данные [2]. Один из семафоров охраняет документацию о существующих в системе стеках, второй - документацию о структуре свободной оперативной памяти, третий - документацию о свободной памяти на барабанах и так далее. В документации каждого открытого файла (заголовка файла) имеется семафор, охраняющий работу с заголовком файла [3]. Таким образом, количество имеющихся в системе семафоров не постоянно и зависит от конкретной ситуации.

Ниже в качестве примера приводится фрагмент программы на АВТОКОДЕ, содержащей критическую секцию, который может быть в каждой процедуре, работающей с общими данными, охраняемыми семафором смф.

```
До обработка  закончена
Цикл
        ЖУЖ(имя смф);
        {анализ и/или модификация общих данных}
        если возникла_необходимость_в_длительных_действиях
          то        убрать  сделанные  модификации
                    ОТКЗАКРЫТЬ(имя смф, имя смф1);
                    выполнить длительные действия
                    ОТКРЫТЬ( имя смф 1)
          иначе
                    ОТКСЕМ(имя смф);
                    обработка  закончена! % выход из цикла
          все
    повторить
```

Здесь в большинстве типичных случаев, когда не требуется длительных действий, анализ и модификация общих данных ограничены синхронизирую-

щими скобками ЖУЖ(имя смф), ОТКСЕМ(имя смф). Однако в некоторых случаях, когда возникает необходимость в длительных действиях, которые не могут выполняться внутри указанных синхронизирующих скобок (например, требуется считать фрагмент общих данных с внешнего устройства), происходит передача охраны общих данных от семафора смф к семафору смф1. Для этих целей с состав процедур синхронизации введена дополнительная процедура ОТКЗАКРЫТЬ(имя смф, имя смф1), которая открывает семафор смф и закрывает семафор смф1.

После выполнения длительных действий (например, после считывания с внешнего носителя фрагмента общих данных в ОЗУ) повторяется очередной виток цикла в попытке провести анализ и модификацию общих данных. Если при этом никаких длительных действий не требуется, то происходит открытие семафора смф и выход из цикла.

2. Средства для синхронизации работы центральных процессоров

Причины, приводящие к необходимости организовывать взаимодействие процессоров в МВК, можно разбить на 2 класса [4]. К первому классу относятся взаимодействия, связанные с мультипроцессорностью системы (например, автоматическая реконфигурация системы в случае сбоев в каком-либо ЦП, синхронизация времени на разных ЦП и тому подобное). Такого рода взаимодействия процессоров обеспечивают надежность функционирования и живучесть системы и возникают достаточно редко, не оказывая заметного влияния на эффективность работы МВК.

Ко второму классу относятся взаимодействия, связанные с наличием у каждого ЦП собственного сверхоперативного запоминающего устройства (СОЗУ) и, следовательно, с соответствием данных в оперативной памяти (ОЗУ) и сверхоперативной памяти (СОЗУ) процессора. Проблема такого соответствия возникла еще в однопроцессорных системах, когда обмен стал выполняться асинхронно с работой ЦП, и усложнилась в мультипроцессорных системах. Частота таких взаимодействий зависит от архитектуры ЦП и оказывает влияние на производительность всей системы в целом.

Введение СОЗУ в архитектуру МВК позволило значительно сократить число обращений в сравнительно медленную оперативную память и тем самым существенно увеличить производительность ЦП. При этом при обращении к общим данным возможна ситуация, когда в СОЗУ процессора будут находиться старые данные, в ОЗУ - обновленные (другим ЦП).

В МВК эта проблема решается следующим образом. Считается, что работа с общими данными может производиться только в критической секции, то есть между операциями ЗАКРЫТЬ и ОТКРЫТЬ или операциями ЖУЖ и ОТКСЕМ. Все данные, записываемые в СОЗУ процессора между этими операциями, помечаются как «общие» и, кроме этого, запоминается время их занесения в СОЗУ. При открытии семафора в него записывается время открытия. При каждом закрытии семафора анализируется записанное в него время и время записи общих данных в СОЗУ. Те общие данные, которые попали в СОЗУ раньше открытия этого семафора, теперь не достоверны, так как могли быть изменены другим ЦП. Те данные, которые попали в СОЗУ после открытия семафора, достоверны, так как они не имеют отношения к данному семафору. Таким образом, на основании сравнения времени открытия семафора и времени занесения общих данных в СОЗУ ЦП однозначно решает проблему достоверности данных. Реализация этого алгоритма позволяет значительно снизить степень взаимодействия между ЦП, однако на пути ликвидации взаимных прерываний, связанных с наличием СОЗУ у ЦП, пред-

стоит решить проблему общих данных, которые заносятся в СОЗУ аппаратно без закрытия каких-либо семафоров. Сюда относятся дескрипторы программных сегментов и таблицы соответствия математических и физических адресов.

С учетом приведенных соображений, введены следующие команды взаимодействия процессоров: ПРЕРВАТЬЦП, ЖДАТЬЦП и ОТВЕТЦП.

Параметром команды ПРЕРВАТЫДП служит маска прерываемых ЦП. В результате исполнения команды процессоры, указанные в маске, получают сигнал прерывания.

Выполнение команды ЖДАТЬЦП заключается в ожидании исполнения команды ОТВЕТЦП от всех процессоров, указанных в команде (параметре команды)

Команда ОТВЕТЦП служит сигналом процессорам продолжать работу.

Подробные примеры организации взаимодействия между ЦП приведены в [4].

3. Организация обменов с внешними устройствами

В высокопроизводительных мультипроцессорных системах с большим количеством внешних устройств, к каждому из которых в целях обеспечения живучести системы существует несколько путей доступа, применение обычных методов сопряжения ЦП с каналами, при которых каждый канал доступен только одному ЦП и может выдавать прерывания только для этого ЦП, приводит к значительным трудностям. Действительно, на ОС возлагается дополнительная работа по планированию операций ввода-вывода при поступлении от какого-либо из ЦП запроса на обмен с ВУ, непосредственно не подсоединенному к каналу, связанному с этим ЦП.

Для преодоления этих трудностей в МВК «Эльбрус» введены специальные процессоры ввода-вывода (ПВВ), которым доступны все имеющиеся в конфигурации МВК внешние устройства (ВУ) и которые сами, в зависимости от текущей обстановки, выбирают канал, через который организуется доступ к требуемому ВУ. Эти ПВВ «экранируют» также ЦП от прерываний ввода-вывода со стороны внешних устройств, освобождая ЦП от рутинной работы по организации ввода-вывода [6,10].

Процессор ввода-вывода является непрограммируемым процессором, работающим по фиксированному алгоритму. Ему, также как и любому ЦП, доступна вся оперативная память. Поэтому ПВВ может производить обмен между ВУ и любым участком оперативной памяти.

Для того, чтобы ПВВ выполнил обмен с некоторым ВУ, ЦП должен оформить заявку на обмен, так называемый блок ввода-вывода (БВВ). Заявка на обмен является единственным документом, который содержит полную информацию о параметрах обмена (код операции, идентификацию ВУ, описатели массива в оперативной памяти, с которым должен быть произведен обмен, направление обмена, режим обмена и тому подобное).

Заявка на обмен представляет собой массив в любом Месте оперативной памяти. В этом массиве на заранее оговоренных местах записаны все параметры требуемого обмена. Поскольку заявки могут формироваться процедурами ОС в любом количестве в произвольные моменты времени, а ПВВ не могут мгновенно выполнить все заявки, взаимодействие между ЦП и ПВВ осуществляется следующим образом.

Во время инициализации операционной системой МВК в оперативной памяти создается так называемая карта работ, содержащая информацию о всех имеющихся в конфигурации внешних устройствах, способах их подключения к ПВВ, а также информацию о всех существующих в данный момент выполненных и невыполненных заявках на обмен. Карта работ доступна как ЦП, так и

ПВВ, в регистры которых при инициализации записываются адреса всех ее элементов.

Карта работ состоит из следующих элементов.

1. Таблица устройств (ТУС) - содержит информацию о всех устройствах, имеющихся в конфигурации МВК, их типах, к каким каналам они подсоединены. Всем ВУ, входящим в конфигурацию МВК, присвоены уникальные номера в диапазоне 0...1023, каналам - номера 0...63, процессорам ввода-вывода - номера 0...3.

Для каждого ВУ в таблице ТУС отведено одно слово, причем номер ВУ в МВК совпадает с номером соответствующего слова в ТУС. В этом слове указывается тип ВУ, путь (или пути) к нему, его состояние в текущий момент времени и другая необходимая для ПВВ информация. ТУС создается ОС во время инициализации МВК. Тогда же адрес этой таблицы записываемся в соответствующий регистр ПВВ. Во время работы МВК информация из ТУС используется и модифицируется как аппаратно, так и программно процедурами ОС.

2. Таблица очередей (ТОЧ) - дает возможность ПВВ иметь доступ ко всем заявкам на обмен, имеющимся в данный момент времени. Каждая заявка представлена так называемым блоком ввода-вывода (БВВ), содержащим всю информацию о параметрах обмена. БВВ представляет собой массив размером около 10 слов, который может быть расположен в любом месте ОЗУ. Все заявки на обмен, относящиеся к одному и тому *же* ВУ, объединены в список таким образом, что в первой заявке записан адрес второй, во второй - адрес третьей и так далее. В ТОЧ для каждого ВУ отводится 2 смежных слова. В первом слове записан адрес первой заявки в очереди, во втором - последней. Если очередь пуста, то соответствующие адреса содержат нули. Начало очереди к К-му ВУ находится в 2К-ом слове таблицы.

3. Дескриптор выполненных работ (ДВР) представляет собой одно слово, которое содержит адреса первого и последнего БВВ в списке выполненных заявок. Этот список создается ПВВ следующим образом. В очередную выполненную заявку ПВВ записывает информацию о результатах выполненного обмена, исключает ее из соответствующего списка заявок к устройству и подсоединяет к концу списка выполненных заявок. В этом списке находятся заявки, относящиеся ко всем устройствам. Следует подчеркнуть, что реорганизация списков осуществляется ПВВ путем модификации адресов в заявках без фактического перемещения заявок. Разбор результатов обменов и исключение заявок из списка выполненных заявок производится ОС в те моменты времени, когда по тем или иным причинам выполнение программы на каком-либо ЦП не может продолжаться и все равно требуется вызов системной процедуры, чтобы переключить ЦП на другой процесс.

4. Базовая команда (БАК) - представляет собой несколько смежных слов в ОЗУ, куда СУПФ записывает команды, которые должны быть выполнены ПВВ после получения соответствующего сигнала от ОС Существует около 20 команд ПВВ. Основной командой взаимодействия ПВВ с ОС и ЦП является команда ПУСКОБЪЕКТА, параметром в которой является номер ВУ. По этой команде ПВВ начинает поочередно выполнять заявки, относящиеся к данному ВУ, выбирая их из головы соответствующего списка в ТОЧ и помещая после выполнения (удачного или с ошибкой) в конец списка выполненных заявок. Работа ПВВ с заданным ВУ прекращается только при исчерпании списка заявок к этому ВУ. После этого ПВВ автоматически начинает обрабатывать очередь заявок к ВУ, если таковая имеется, заявки в которой не могли выполняться из-за того, что требуемый канал был занят. Остальные команды ПВВ служат для специальных целей, таких как загрузка регистров ПВВ, проверка ПВВ и тому подобное.

Элементы 1 и 2 карты работ являются общими для всех ПВВ, а элементы 3 и 4 - у каждого ПВВ свои.

Таким образом, во время работы МВК процессоры ввода-вывода перерабатывают списки заявок на обмены из ТОЧ, выполняя обмены с различными ВУ и стремясь при этом полностью загрузить подсоединенное к, нему оборудование. ПВВ стремятся сделать эти списки пустыми. С другой стороны, все ЦП пополняют списки заявок в ТОЧ и временами анализируют результаты выполненных обменов, просматривая списки выполненных заявок и убирая заявки из них. Такое взаимодействие через очереди заявок между ПВВ и ЦП позволяет свести к минимуму синхронизацию между ними.

Описанный жесткий и достаточно простой алгоритм работы ПВВ по таблицам ТУС и ТОЧ позволяет, с одной стороны, реализовать его аппаратно, а, с другой стороны, освобождает ЦП от рутинной работы, связанной с поиском в данный момент времени пути к ВУ и так далее. Кроме того, окончание обмена с каким-либо ВУ необязательно приводит к прерыванию ЦП, поскольку окончания многих обменов могут быть обработаны ЦП в одном сеансе обработки в наиболее подходящее для них время.

Таким образом, для выполнения обмена ОС должна лишь заполнить заявку на обмен в любой области оперативной памяти и вставить ее в список заявок к заданному ВУ. Если при этом список заявок был пуст, ОС должна загрузить в БАК команду ПУСКОБЪЕКТА с заданным номером ВУ и послать сигнал ПВВ (прервать ПВВ). Через некоторое время после завершения обмена процедура ОС при анализе очереди выполненных заявок встретит эту заявку и может приступить к анализу записанной ПВВ в заявку информации о результатах обмена.

Для повышения живучести системы и обеспечения возможности доступа к информации на всех накопителях на магнитных дисках (НМД) при выходе из строя некоторых устройств управления дисками (УД), каналов и ПВВ, УД подключаются к НМД через электронные коммутаторы, которые могут коммутировать любой из подсоединенных к ним УД (до 4-х устройств) с любым из подсоединенных к ним НМД (до 16 устройств). При выходе из строя какого либо из УД сохраняется доступ ко всем НМД через другие УД. Для обеспечения доступа ко всем 16 НМД при выходе из строя канала или ПВВ достаточно, чтобы один из подсоединенных к электронным коммутаторам УД был связан с другим каналом или ПВВ.

При реализации компонент ОС, отвечающих за обмен с внешними устройствами, принимались во внимание следующие 2 обстоятельства.

Во-первых, обмены производятся специализированными процессорами ввода-вывода, которые работают асинхронно и независимо от ЦП, выполняя приготовленные ими заявки на обмены. Во-вторых, длительность обмена в лучшем случае при отсутствии очереди к соответствующему ВУ измеряется десятками миллисекунд. За это время ЦП может выполнить около миллиона операций. Поэтому в том случае, если стек, от имени которого заказан обмен, должен ожидать окончания обмена, имеет смысл этот стек снять с ЦП и предоставить ЦП другому процессу из очереди готовых процессов. Снятый с ЦП стек становится в очередь к семафору, который открывается по окончании обмена. Как отмечалось выше, при этом стек переводится в очередь готовых стеков и, когда подойдет его очередь, будет активизирован на любом освободившемся в это время ЦП. Очевидно, что в многопроцессорной системе к одному и тому же ВУ могут обращаться стеки, активизированные на разных ЦП. Поэтому к ВУ может образоваться очередь на обмены.

Во многих применениях оказывается полезным иметь так называемые асинхронные обмены, при которых обмен с ВУ производится одновременно с продолжением работы на стеке, заказавшем обмен. Если выполняемой на стеке про-

грамме понадобятся результаты обмена, то она выполняет операцию ЖДАТЬ над семафором, связанным с этим обменом.

Если обмен завершен, произведен анализ его результатов и открыт семафор в БВВ, то процедура ЖДАТЬ заканчивается и выполняются следующие за ней операторы, в которых могут уже использоваться результаты обмена. Если обмен еще не закончен и семафор закрыт, то процедура ЖДАТЬ деактивизирует стек и ставит его в очередь к семафору, связанному с обменом.

Асинхронные обмены могут быть реализованы с помощью имеющегося в ОС и описанного выше механизма параллельных процессов. Однако, порождение параллельного процесса с собственным стеком для каждого обмена ведет к большим накладным расходам. Гораздо проще сформировать заявку на обмен в виде небольшого массива, в котором указаны все параметры обмена (номер ВУ, адрес на ВУ, адрес и размер массива в ОП, который участвует в обмене в ВУ, направление обмена, режим проверки правильности обмена и тому подобное). Эта же заявка содержит семафор, который будет открыт при успешном завершении обмена. Структура этой заявки известна аппаратуре процессора ввода-вывода (ПВВ). Ему же известны голова и хвост очереди из заявок к каждому ВУ, а также местоположение областей (линков) в заявке, используемых для организации очереди к ВУ в виде списка.

При такой организации обменов алгоритм работы ПВВ чрезвычайно прост. Имея информацию о заявках на обмен, о текущем состоянии ВУ, о текущих путях доступа к ВУ, ПВВ старается максимально загрузить все ВУ. По завершении обмена каждое ВУ выдает информацию о результатах обмена, в том числе о встретившихся при обмене ошибках (исключительных случаях). Эту информацию ПВ без всякого анализа записывает в соответствующую заявку и переставляет заявку из головы очереди к ВУ в хвост очереди выполненных работ. Если к этому ВУ имеются в очереди еще заявки, то запускается следующая заявка.

Очередь выполненных работ одна на все ВУ. Пока о завершении обмена никто не знает и семафор в заявке закрыт, необходимо сначала проанализировать результаты обмена. Эта работа выполняется соответствующей процедурой ОС АНАЛИЗ. Время, стек и ЦП для работы этой процедуры выбираются следующим образом. Пока на всех ЦП стеки активны, не имеет смысла их прерывать для запуска процедуры АНАЛИЗ. Однако, если по каким-либо причинам некоторый стек требуется снять с ЦП, то должна вызваться процедура ОС СМЕНИТЬСТЕК для смены стеков на ЦП. В это время ОС имеет в своем распоряжении стек (правда, не свой) и ЦП. Перед снятием стека на нем запускается процедура АНАЛИЗ, которая проверяет результаты обмена.

Если обмен по некоторой заявке прошел успешно, то процедура АНАЛИЗ вызывает процедуру ОТКРЫТЬ для семафора в заявке и удаляет заявку из очереди выполненных работ. Если при обмене были ошибки, то предпринимаются различные действия в зависимости от типа ВУ и характера ошибки. В одних случаях при сбоях обмен повторяется с подсчетом количества повторов, в других случаях перед повтором предпринимаются действия по возврату ВУ в исходное состояние (например, при чтении с магнитной ленты), либо действия, связанные с запросом к оператору на помощь.

Если системные реакции на ошибки обмена оказались безуспешными, в стек, заказавший обмен, выдается ситуация, связанная с ошибкой обмена. Пользователь имеет возможность указать собственную реакцию на эту ситуацию. Если такая реакция отсутствует, то выполняется системная реакция, приводящая обычно к прекращению задачи с выдачей соответствующего сообщения. В любом случае процедура АНАЛИЗ анализирует по очереди и уда-

ляет заявки из очереди выполненных работ. Подчеркнем, что системная процедура АНАЛИЗ выполняется на случайном стеке пользователя.

Во многих применениях результаты обмена требуются сразу после того, как обмен заказан, то есть стек не может быть активным пока не закончится обмен. В этом случае заявка на обмен может быть сформирована непосредственно в стеке. После этого стек может быть деактивизирован и поставлен в очередь к семафору, находящемуся в заявке, которая расположена в этом стеке. При успешном выполнении обмена семафор в заявке открывается и стек попадает в очередь готовых работ. При активизации стека на ЦП заканчивается процедура, заказавшая обмен. При этом БВВ вычеркивается из стека и на стеке выполняются операторы, следующие за этой процедурой.

4. Организация системы файлов МВК «Эльбрус»

С целью предоставить пользователю наиболее удобный уровень работы с информацией на внешних носителях в ОС реализовано 3 иерархически упорядоченных уровня файлов [5]:

- простые файлы;
- буферизованные файлы;
- структурированные файлы.

Для решения традиционных задач, возникающих при реализации режима мультипрограммирования, связанных с необходимостью скрыть от пользователей реальное размещение информации на внешних устройствах и обеспечить независимость программ от конкретной конфигурации периферийного оборудования, выделен естественный минимальный уровень простых файлов. Этот уровень характеризуется следующими двумя характерными чертами.

Простые файлы демонстративно игнорируют специфику информации, которая в них записана. В частности, в простом файле отсутствует понятие логической записи. Простые файлы различаются по типам. Каждому типу внешнего устройства соответствует свой тип простого файла. В МВК «Эльбрус» имеются следующие 10 типов простых файлов.

1. Файлы на магнитных дисках (МД-файлы)
2. Файлы на магнитных барабанах (МБ-файлы)
3. Файлы на магнитных лентах (МЛ-файлы)
4. Файлы на алфавитно-цифровых печатающих устройствах (АЦПУ-файлы)
5. Вводные файлы на перфокартах (ЧПК-файлы)
6. Выводные файлы на перфокартах (ЗПК-файлы)
7. Вводные файлы на перфолентах (ЧПЛ-файлы)
8. Выводные файлы на перфолентах (ЗПЛ-файлы)
9. Файлы на алфавитно-цифровых дисплеях (АЦД-файлы)
10. Файлы на пишущих машинках (ПМ-файлы)

Каждый тип файла имеет специфику в наборе операций, которые над ним можно выполнять, в системе адресации к информации в файле, в наборе ошибок (исключительных случаев), которые могут возникнуть при работе с файлом.

Введение уровня простых файлов позволяет:

- снабдить файл некоторыми полезными макросвойствами, которые отсутствуют у соответствующего ВУ, в частности, обеспечить стандартные реакции на ошибки обмена;
- унифицировать для пользователя операции обмена с различного типа ВУ, что не только облегчает изучение, но и позволяет одной и той же программе обрабатывать файлы различных типов;
- обеспечить необходимые пользователю реакции на ошибки обмена, предоставив ему тем самым возможность изменять свойства ВУ.

Простые файлы используются в различных языках программирования. Поэтому у написанных на различных языках программ появляется возможность обмениваться информацией через простые файлы. Большинство средств для работы с простыми файлами встроены в базовый язык программирования АВТОКОД. Поскольку сама ОС написана на этом языке, то имеется возможность использовать простые файлы при обменах с внешними устройствами, то есть свою же собственную продукцию.

Следует отметить следующее немаловажное обстоятельство. Введение простых файлов, которыми пользуются все без исключения компоненты программного обеспечения МВК «Эльбрус», позволяет перенести на этот уровень практически всю сервисную службу, освободив тем самым пользователей простых файлов от необходимости создания собственных сервисных программ. Сервисная служба простых файлов обеспечивает решение общих задач, возникающих при эксплуатации МВК, а именно:

- предоставление памяти на внешних носителях практически неограниченного объема;
- развитые средства для администрации МВК по организации хранения данных, санкционированию доступа к данным, контролю за деятельностью пользователей;
- каталогизацию простых файлов;
- контекстную защиту простых файлов от несанкционированного доступа;
- копирование и сличение простых файлов;
- «сборку мусора», то есть обнаружение файлов, которые никем не могут быть использованы, и их уничтожение;
- автоматическое выталкивание редко используемых файлов на менее дефицитные носители;
- получение различных сбросов и дампов;
- восстановление документации простых файлов при сбоях аппаратуры и повреждении носителей;
- очистку некоторого диска или барабана от информации путем перезаписи этой информации на другие носители.

Функции сервисной службы могут постоянно расширяться и дополняться. При этом дополнения автоматически становятся доступными всем другим компонентам системы.

На основе простых файлов реализованы буферизованные файлы. При открытии буферизованного файла с ним связывается совокупность системных массивов в оперативной памяти, называемых буферами. Пользователь имеет возможность вести обработку информации непосредственно на буферах. Понятие логической записи на этом уровне не вводится. Содержимое буфера рассматривается как копия в оперативной памяти некоторого фрагмента файла (блока).

При использовании буферизованных файлов оптимизируется последовательная обработка файлов. При последовательном чтении организуется под-качка впрок очередных порций информации из файла в буфера, а при последовательной записи - освобождение впрок буферов с тем, чтобы пользователь имел возможность заполнить их своей информацией.

При обработке файлов в произвольном порядке совокупность связанных с файлом буферов рассматривается как прикрепленное к файлу ассоциативное запоминающее устройство, более быстрое, чем то, на котором хранится записанная в файл информация. При этом в буферах оказывается наиболее часто используемая информация, что приводит к сокращению количества обменов с внешними устройствами при работе с файлом. Такой подход к буферизации является продолжением общего подхода, принятого при разработке не только

программного обеспечения, но и аппаратуры МВК (ассоциативное запоминающее устройство чисел, ассоциативное запоминающее устройство страниц, буфер команд и тому подобное). В тех случаях, когда производится заполнение файла новой информацией и заведомо известно, что имеющаяся в файле старая информация не нужна и может быть забита, вместо операции записи используется операция НОВ, которая является оптимизацией для этого случая операции ЗАП. При выполнении операции ЗАП считывание из файла не производится, а пользователю сразу выдается дескриптор на свободный буфер для его заполнения. Большинство средств для работы с буферизованными файлами также встроено в АВТОКОД.

Для удовлетворения требований различных СУБД на следующем уровне иерархии введены структурированные файлы. Как следует из самого названия, системные процедуры, обрабатывающие файлы на этом уровне, «понимают» структуру записанной в файл информации. Эта структура может определяться на специальном языке описания данных (ЯОД). На уровне структурированных файлов появляются понятие логической записи и средства по обработке записей. Имеется возможность выбирать записи с заданным значением некоторого поля (ключа). В файлах, расположенных на устройствах с произвольным доступом, появляется возможность добавлять, удалять, обновлять записи, выбирать запись по ссылке, хранящейся в другой записи, устанавливать ссылки между записями и так далее. Распределение свободной памяти внутри структурированного файла также осуществляется системными процедурами. Следует подчеркнуть, что пользователь не может обращаться к данным в структурированном файле иначе, чем через системные процедуры, связанные с этим файлом. На уровне структурированных файлов реализуются общие для различных систем управления базами данных (СУБД) средства по организации коллективной работы многих пользователей с общими данными, средства по протоколированию изменений в данных и откатам назад, средства по восстановлению данных при сбоях аппаратуры, поломке носителей, ошибках пользователей.

4.1. Система простых файлов

Процедуры системы простых файлов (СПФ) непосредственно связаны с аппаратурой ввода-вывода. Однако это совершенно не означает, что процедуры СПФ должны использовать только аппаратные средства. В это время уже функционируют системы управления памятью и процессами. В частности, сами процедуры СПФ работают на стеке, соответствующем некоторому процессу, от имени которого заказывается обмен. В это время доступны также все средства синхронизации, представляемые ОС.

Файл, с которым должен быть произведен обмен, определяется заголовком (описателем) файла, ссылка на который передается в процедуру обмена в качестве параметра. Подобно тому как дескриптор массива содержит полную информацию о параметрах массива (местоположение, размер, формат элементов и тому подобное), заголовок файла содержит полную информацию о файле. Однако, в отличие от дескриптора массива, информация о файле значительно превосходит по объему одно машинное слово. Поэтому файл идентифицируется машинным словом, содержащим адрес массива, в котором содержится вся информация о файле, и признак, что это слово указывает на заголовок файла. Это слово имеет специальный тег адресной информации. В непривилегированном режиме пользователь не имеет возможности сформировать или изменить это слово, точно также, как он не может сформировать дескриптор массива: Он может лишь получить такое слово, обратившись к процедурам СПФ, осуществляющим открытие или создание простого файла, а затем

подавать это слово в качестве параметра в процедуры СПФ. Это слово называется словом файла, а массив, на который оно ссылается и который содержит полную информацию о файле, называется заголовком файла.

СПФ различает синхронные и асинхронные непосредственные обмены. При выполнении асинхронного обмена, например, чтения с ВУ, СПФ формирует БВВ и вставляет его в очередь к соответствующему ВУ. На этом работа процедуры СПФ заканчивается и продолжается работа заказавшего обмен процесса асинхронно с выполнением обмена. После завершения обмена ПВВ переставляет БВВ в очередь выполненных заявок. В некоторый момент времени на некотором случайном стеке в то время, когда некоторый процесс не может продолжаться и требуется вызов системной процедуры, чтобы снять его с ЦП, заодно запускается системная процедура АНАЛИЗ, обеспечивающая анализ результатов обмена по заявкам, имеющимся в очереди выполненных заявок. В том случае, если обмен выполнился без ошибок, процедура АНАЛИЗ открывает имеющийся в БВВ семафор, который был закрыт при формировании БВВ.

Если в какой-то момент времени заказавший асинхронный обмен процесс должен воспользоваться результатами обмена, например, начать анализ считанной информации, то перед этим должен быть выполнен примитив ЖДАТЬ, параметром которого является ссылка на семафор в БВВ. Если во время выполнения этого примитива семафор уже открыт, то процесс продолжает выполнять команды, следующие за этим примитивом (например, начинает анализ считанной информации).

Если же семафор закрыт, то процесс примитивом ЖДАТЬ снимается с ЦП и ставится в очередь ждущих процессов к этому закрытому семафору. Когда обмен будет закончен и СПФ убедится, что обмен прошел нормально, она открывает семафор в БВВ при помощи примитива ОТКРЫТЬ. При этом рассматриваемый процесс становится готовым к дальнейшему выполнению и переводится этим примитивом в очередь готовых процессов. После того, как подойдет очередь этого процесса и освободится некоторый ЦП, процесс будет продолжен (например, начнет анализироваться считанная информация).

При выполнении синхронного обмена примитив ЖДАТЬ выполняется СПФ сразу после того, как сформированный БВВ будет вставлен в очередь к соответствующему ВУ. Поскольку к этому моменту времени обмен заведомо не успевает выполниться, заказавший обмен процесс будет снят с ЦП.

Таким образом, можно выделить следующие этапы выполнения непосредственного синхронного обмена.

1. Формирование заявки на обмен (БВВ) и включение его в очередь заявок к соответствующему ВУ.
2. Пребывание БВВ в этой очереди.
3. Фактическое выполнение обмена с помощью ПВВ, который переставляет БВВ в очередь выполненных заявок.
4. Пребывание БВВ в этой очереди.
5. Анализ результатов обмена.
6. Пребывание заказавшего обмен процесса в очереди готовых процессов в ожидании, когда освободится некоторый ЦП.

Этап 1 выполняется на стеке того процесса, который заказывает обмен.

Этап 3 выполняется некоторым ПВВ.

Этап 5 выполняется на некотором случайном стеке, на котором СПФ производит анализ результатов обменов по заявкам, имеющимся в очереди выполненных заявок.

В общем случае при выполнении непосредственного синхронного обмена необходимо пройти через 3 очереди на этапах 2,4,6.

Следует заметить, что при формировании синхронного непосредственного обмена память для БВВ может быть взята в верхушке стека, так как стек до завершения обмена для дальнейшего выполнения программы все равно не нужен, и при этом не требуется обращаться к процедурам ОС для выделения памяти.

При формировании асинхронного непосредственного обмена, поскольку одновременно с выполнением обмена будет идти дальнейшее выполнение программы и для этого нужен стек, место для БВВ должно выделяться вне стека. Чтобы при формировании каждого асинхронного обмена не обращаться к процедурам выделения и освобождения памяти, в СПФ имеется возможность открыть файл для асинхронного обмена. В этом случае СПФ связывает с заголовком файла область, которая используется для формирования БВВ, а пользователю выдается в качестве результата слово файла на эту область (Рис.4). Разумеется, от имени такого слова файла может быть сформирована заявка на асинхронный обмен только в том случае, если соответствующий БВВ в это время не участвует в другом обмене.

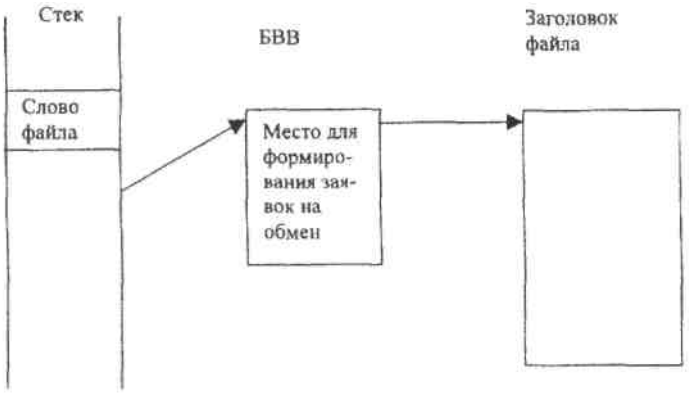


Рис.4. Представление документации файла при асинхронных обменах.

Заявки на синхронные и асинхронные обмены формируются процедурой СПФ, в параметрах которой указываются:

- код операции (запись, чтение и другие);
- слово файла, определяющее файл, с которым должен быть произведен обмен;
- описатель массива пользователя (если выполняются операции записи или чтения);
- адрес на файле (для файлов с произвольным доступом).

Заполнение БВВ производится на основе информации из параметров процедуры и заголовка файла. При этом следует принимать во внимание следующие обстоятельства.

1. ПВВ совершенно не понимает математических адресов и в БВВ необходимо записывать только физические адреса. Поэтому, если в качестве параметра в процедуру подан дескриптор массива пользователя, содержащий адрес, у которого 8 старших разрядов отличны от 0, то этот массив загружен в математическую память и, следовательно, может перемещаться в ОП, а может и вообще там отсутствовать (непривилегированный пользователь только такие массивы и может иметь). На время обмена такой массив необходимо зарезидентировать, то есть сделать неперемещаемым в ОП. Резидентация выполняется путем вызова процедуры ОС, которая отмечает в документации массива, что он резидентный, находится в обмене. Эта же процедура выдает в качестве результата его физический адрес.

2. Поскольку при формировании БВВ для асинхронного обмена память для БВВ берется из стека, необходимо обеспечить, чтобы БВВ не попал на границу двух смежных страниц, так как в этом случае по физической памяти БВВ будет не смежным, а ПВВ не может обрабатывать такие БВВ. В этих случаях обычно заказывается память удвоенного размера и в ней выбирается для БВВ смежный по физической памяти массив.

Для того, чтобы освободить пользователя от необходимости все время принимать во внимание ограниченный объем информации, умещающейся на одном накопителе на магнитных дисках (НМД), в СПФ введено понятие контейнера. В контейнер на магнитных дисках (МД-контейнер) может входить произвольное количество НМД. МД-контейнер выступает как поставщик ресурсов под МД-файлы и является местом их хранения. Файл на магнитных дисках представляет собой пронумерованную совокупность сегментов размером по 32 64-разрядных слова. Для облегчения выделения внешней памяти под файл, он разбивается на листы. Размер листа и максимальное количество листов в файле могут быть указаны при создании файла. Листы в файле последовательно пронумерованы, начиная с 0. Каждому листу файла по мере необходимости в МД-контейнере отводится непрерывная область на каком-либо НМД. У одного и того же файла листы могут располагаться на разных НМД. Но каждый лист всегда располагается на одном. Любой МД-файл всегда полностью располагается в каком-либо одном МД-контейнере.

Заголовок МД-файла представляет собой массив, в котором записаны адреса всех связанных с файлом листов. Адрес листа определяется двумя целыми числами: номером НМД в МД-контейнере и номером сегмента, с которого начинается лист на этом НМД. Заголовок МД-файла содержит также служебную информацию, определяющую текущее состояние файла и значения его атрибутов.

Допускается обмен между МД-файлом и оперативной памятью порциями произвольного размера. Адрес порции на файле задается номером сегмента относительно начала файла. При записи на файл по адресу, соответствующему еще неинициализированному листу, СПФ запрашивает под лист свободную память в контейнере и прикрепляет полученную область к файлу.

С целью упростить организацию непосредственных обменов, на размер массива на файле накладываются ограничения, при которых каждый обмен с файлом производится всегда только в рамках одного листа. Вообще, каждая операция обмена с любым простым файлом соответствует одной операции обмена с соответствующим ВУ (чтение одной перфокарты, печать одной строки на АЦПУ и тому подобное).

Все имеющиеся в системе магнитные барабаны объединены в один резидентный МБ-контейнер. Свойства и организация МБ-файлов совпадают с МД-файлами.

По аналогии с контейнерами на магнитных дисках в СПФ введено понятие контейнера на магнитных лентах, который представляет собой совокупность одной или нескольких лент, объединенных общим именем и пронумерованных в пределах этой совокупности. Допускается расположение нескольких файлов на одной ленте, одного файла на нескольких лентах и произвольное расположение многих файлов на многих лентах, причем никаких ограничений на размещение файлов на лентах не накладывается. Это позволяет пользователю при желании не принимать во внимание ограниченный объем одной ленты и не заботиться о необходимых переходах с одной ленты на другую.

В МВК принято, что магнитные ленты являются основными носителями информации для обмена информацией с другими вычислительными системами. Поэтому общая структура информации на магнитных лентах полностью отвечает требованиям соответствующего отраслевого стандарта. В терминологии этого

стандарта МЛ-контейнер может быть охарактеризован как «многотомная многофайловая совокупность».

4.2. Система буферизованных файлов

При буферизованных обменах файл рассматривается как совокупность блоков информации. Максимальный размер блока определяется атрибутом файла МАКСДЛИНБЛОКА. Все блоки файла последовательно пронумерованы, начиная с 0.

В отличие от непосредственных обменов, при которых обмен происходит между ВУ и массивом, который заказал сам пользователь, при буферизованных обменах обмены производятся между ВУ и системными массивами, называемыми буферами. Все буфера у одного и того же файла имеют одинаковый размер. Пользователю выдаются дескрипторы этих буферов, так что он может обрабатывать информацию в них, как в обычных массивах.

Таким образом, при буферизованных обменах с файлом пользователь избавлен от необходимости заказывать собственные массивы для обработки информации в файле и решать вопрос, какие из этих массивов и когда можно переиспользовать.

Кроме того, система буферизованных файлов (СБФ) пытается предугадать, какие блоки информации понадобятся в будущем и организует их подкачку в неиспользуемые буфера на фоне выполнения работающей с файлом задачи. В некоторых случаях это может существенно ускорить обработку файла. Буферизованные обмены полностью реализуются на основе непосредственных и в этом смысле они являются надстройкой над последними. Особое внимание при проектировании буферизованных файлов обращалось на эффективную реализацию последовательной обработки файлов, которая и в настоящее время остается наиболее распространенной.

При организации буферизованных обменов с МБ - файлами и МД - файлами, обеспечивающими произвольный доступ к хранящейся в них информации, необходимо иметь в виду, что различные системы управления базами данных (СУБД) будут реализованы, главным образом, на основе этих обменов. Поэтому выдвигаемые СУБД специфические требования по сохранению целостности, непротиворечивости информации, возможности одновременного доступа к одним и тем же данным со стороны различных процессов, по обеспечению надежности хранения и восстановления информации, по обеспечению возможностей управлять алгоритмом переиспользования буферов должны приниматься во внимание при организации буферизованных обменов.

Это нашло отражение в том, что каждый файл МБ и МД, независимо от количества открытий и количества работающих с ним задач, имеет один общий связанный с заголовком файла комплект буферов. Число буферов у файла не фиксировано, а определяется из тех же соображений, что и общий объем массивов в ОП, относящихся к некоторой задаче, то есть изменяется в зависимости от интенсивности работы с файлом и наличия свободной памяти в системе.

Для хранения специфичной для каждого приложения информации о буфере (семафоров для обеспечения синхронизации при обращении к данным в буфере, различных признаков и счетчиков, характеризующих состояние буфера) пользователю предоставляется возможность связывать с каждым буфером дополнительный массив, размер которого может указываться при открытии файла для буферизованного обмена в качестве значения атрибута ОБЛАСТЬБУФ, а также средства для чтения и записи информации в такие массивы.

При буферизованных обменах с файлом пользователю предоставляются следующие средства.

1. Чтение блока с заданным в операции номером в один из буферов, связанных с заголовком файла. В качестве результата операция чтения выдает дескриптор буфера, в который было произведено чтение, с признаком запрета записи. По этому дескриптору пользователь может считывать информацию из любой части буфера, как из своего собственного массива. Буфер по этому дескриптору доступен до тех пор, пока пользователем каким-либо способом не будет указано о разрыве связи между дескриптором и буфером.

2. Запись блока с заданным номером. В этом случае СБФ производит чтение блока с указанным номером в один из связанных с заголовком файла буферов и в качестве результата операции выдает дескриптор этого буфера без признака запрета записи. При помощи этого дескриптора пользователь может заполнять буфер произвольной информацией. Фактическая запись буфера на файл производится в неизвестный для пользователя момент времени, но уже после того, как он каким-либо способом указал о разрыве связи между дескриптором и буфером.

3. В целях оптимизации работы с файлом при записи информации в тех случаях, когда пользователю заранее известно, что записываемый на файл буфер будет им полностью заполнен новой информацией и поэтому нет необходимости считывать из соответствующего блока файла информацию в буфер, введена операция НОВ, которая находит среди связанных с заголовком файла свободный буфер, приписывает ему заданный в операции НОВ номер блока и выдает дескриптор этого буфера в качестве результата. Пользователь может использовать полученный дескриптор так же, как в операции записи.

4. СБФ предоставляет пользователю возможность в приведенных выше операциях задавать требуемый номер блока не в абсолютном виде, а относительно некоторого блока, определяемого дескриптором буфера, в котором находится этот блок.

5. СБФ вводит понятие текущего блока, как блока, который участвовал в последней операции над файлом. В операциях 1-3 пользователь может задать требуемый блок относительно текущего. Кроме того, СБФ имеет средства устанавливать текущим любой блок файла, а также сдвигать текущий блок к концу или к началу файла на любое количество блоков.

6. В СБФ имеются средства для явного указания на разрыв связи между некоторыми дескрипторами и описываемыми ими буферами (операция ОТКРЕП), для явного указания необходимости форсированной записи на файл перечисленных в параметрах буферов (операция СБРОС), для явного указания необходимости ликвидации перечисленных в параметрах буферов (операция ЛИКВИДБУФ). Имеется возможность явно указывать в операциях чтения и записи, что буфер целесообразно сразу переиспользовать после того, как он будет откреплен.

7. СБФ поддерживает так называемый однобуферный способ обмена, при котором при каждой очередной операции неявно выполняется операция ОТКРЕП для буфера, участвующего в предыдущей операции чтения или записи. Таким образом, при таком способе обмена пользователь может иметь не более одного действительного дескриптора буфера.

8. СБФ следит за стилем обработки информации в файле. В тех случаях, когда начинается последовательная обработка файла, СБФ организует подкачку впрок в свободные буфера очередных блоков файла, если выполняются операции чтения или записи, либо организует наличие свободных буферов, если выполняются операции НОВ.

9. Все перечисленные операции могут выполняться в синхронном и асинхронном режимах. Стиль использования операций в этих режимах такой же, как и

рассмотренный выше стиль использования синхронных и асинхронных непосредственных обменов.

Прежде чем пользователь может приступить к буферизованным обменам с файлом, СБФ должна связать с соответствующим заголовком некоторую документацию и совокупность буферов. Наиболее просто для СБФ и, естественно, для пользователя, если перед началом производится подготовка файла для буферизованных обменов, а после того, как необходимые обмены с файлом выполнены, выполняются обратные действия, которые возвращают файл в прежнее состояние. Иными словами, прежде чем производить обмены с файлом каким-либо способом, должен быть создан объект соответствующего типа, а после окончания обменов с файлом этим способом созданный объект следует открепить.

При открытии файла для буферизованного обмена СБФ создает в оперативной памяти и прикрепляет к заголовку открываемого файла системный массив, называемый позиционная переменная (ПОЗП). Он содержит ссылку на заголовок файла, признак, определяющий режим буферизованного обмена (многобуферный или однобуферный), информацию о текущем буфере, формате элементов, из которых, как считает пользователь, состоит буфер (значение атрибута ФОРМЭЛЕМ), начальный адрес области математической памяти, которая будет использоваться для загрузки в нее буферов, когда дескрипторы буферов выдаются в качестве результата операций ЧИТ, ЗАП, НОВ. Массив ПОЗП загружен в математическую память задачи, которая произвела открытие файла. Слово файла на этот массив выдается пользователю процедурой открытия в качестве результата, счетчик открытий в заголовке файла при этом увеличивается на 1. С одним и тем же заголовком файла могут быть связано несколько массивов ПОЗП.

4.3. Система структурированных файлов

Структурированные файлы предназначены, в первую очередь, для организации различных систем управления базами данных (СУБД). В настоящее время известно несколько тысяч различных СУБД, ориентированных на различные приложения и различные ЭВМ. Такое многообразие показывает, что требования пользователей СУБД настолько разносторонни и противоречивы, что попытка разработать некоторую универсальную СУБД, удовлетворяющую всем требованиям, является, по-видимому, малоперспективной. Тем не менее к настоящему времени выработаны некоторые основополагающие принципы, которые должны выполняться при реализации любой универсальной СУБД. Такими принципами являются:

1. В процессе эксплуатации базы данных должна иметься возможность изменять структуру хранения и стратегию доступа к данным.
2. Должна быть обеспечена так называемая независимость данных, то есть прикладные программы не должны изменяться при изменении структуры хранения и стратегии доступа к ним.
3. Пользователь может обращаться к данным только через процедуры доступа, которые связаны с этими данными и ведут их обработку. Процедуры доступа рассматриваются как составная часть СУБД.
4. Должна быть некоторая служба, представляемая администратором базы данных, которая бы отвечала за сохранность и достоверность данных, и на основе компромисса разрешала противоречивые требования различных пользователей. Для выполнения этих функций администратор базы данных должен иметь соответствующие средства.
5. По требованию отдельных пользователей должна быть обеспечена секретность хранения данных, то есть должна иметься возможность защищать данные от несанкционированного доступа.

6. Должны быть факультативные средства для синхронизации одновременного доступа к данным со стороны нескольких независимых задач.

7. Должны иметься средства, обеспечивающие сохранность данных при отказах оборудования и поломках носителей, На которых хранятся данные.

Анализ реализаций различных СУБД показывает, что многообразие используемых при этом методов и приемов существенно меньше, причем они поддаются классификации. Так, данные обычно хранятся в записях, представляющих собой некоторым образом структурированный набор элементов. Записи хранятся в файлах. Имеются средства для включения и исключения записей, обновления записей, поиска записей, удовлетворяющих некоторому условию и тому подобное.

Между записями могут устанавливаться ссылки, объединяющие записи в ансамбли, имеющие некоторую определенную семантику (вхождение набора в сетевой модели данных, список записей-деталей в иерархической, отношения в реляционной). В качестве средства для объединения записей в ансамбли широко используется группировка записей в физически смежные области (блок, файл). Различие в реализациях проявляетея лишь в способах группировки и упорядочения записей.

Подобно тому, как включение в состав СПФ разнообразных сервисных средств позволяет пользователям свести к минимуму собственную сервисную службу, в ССФ целесообразно включить средства, облегчающие пользователям эффективную и удобную эксплуатацию собственной СУБД. В первую очередь это относится к средствам, обеспечивающим одновременную работу независимых пользователей с информацией в структурированных файлах, и к средствам, обеспечивающим живучесть структурированных файлов, то есть восстановление содержащейся в них информации при сбоях ЦП, ОП, Пропадании информации на внешних носителях, а также при ошибочных действиях самих пользователей. При проектировании любой СУБД эти средства требуют к себе достаточно большого внимания и реализующие их программы занимают значительный объем в общем объеме программ, обеспечивающих функционирование СУБД.

Таким образом, введение уровня структурированных файлов предоставляет пользователям удобные средства для работе с структурированными данными, освобождая пользователей от необходимости реализации механизмов, обеспечивающих как фундаментальные, так и сервисные функции.

5. Принципы распределения ресурсов МВК

Задача распределения ресурсов является одной из основных, которые приходится решать ОС. При этом имеется в виду распределение времени и пространства между различными задачами. Распределение времени - распределение времени работы всех наличных ЦП для решения имеющихся в системе задач. При этом необходимо учитывать приоритет задачи, требуемые ею ресурсы, а также текущую обстановку в системе. Под пространством прежде всего имеются в виду ресурсы системы, используемые для хранения данных различных задач. К ним относятся вся имеющаяся в наличии оперативная память (ОП) системы, внешняя память на всех наличных барабанах и резидентных дисках, дисководы для съемных дисков, магнитофоны для магнитных лент и тому подобное.

5.1. Распределение ЦП

Если не задано иначе, ОС МВК стремится к повышению общей производительности МВК, то есть стремится загрузить ЦП работой по выполнению за-

дач в максимальной степени. При этом обычно в решении находится гораздо большее количество запущенных задач *b* (решение которых начато, но не закончено), чем количество *a* активных задач, которое может выполняться в каждый момент времени. Очевидно, что *a* равно количеству работающих в данный момент времени ЦП. В многопроцессорной системе весьма продуктивным с точки зрения повышения скорости решения конкретной задачи является понятие процесса - независимой активной деятельности в рамках одной задачи [2]. Каждая задача может запустить достаточно много параллельных процессов, которые могут выполняться одновременно с запустившей их задачей на других ЦП, повышая таким образом общую скорость решения задачи. Параллельные процессы могут быть практически независимы от задачи (например, фоновая печать результатов), либо их выполнение может быть засинхронизовано между собой и с запустившей их задачей. В любом случае параллельный процесс имеет сходство с задачей в том отношении, что он имеет собственный стек и может на равных претендовать на использование ресурсов, в том числе и ЦП. Однако все параллельные процессы в рамках одной задачи работают в общей математической памяти и поэтому могут иметь в ОП общие данные. В этом случае задача представлена деревом стеков, которое во время решения может видоизменяться, так как некоторые процессы могут заканчиваться и соответствующие стеки исчезать. В то же время могут инициироваться новые процессы с появлением соответствующих новых стеков.

Что касается различных задач, то каждая из них имеет собственную математическую память. Поэтому связь между задачами через общую ОП отсутствует. Это обеспечивает надежную защиту данных каждой задачи от других задач. Во многих случаях ОС не делает различия между стеками, относящимся к задачам, и стеками, относящимся к процессам. В дальнейшем, когда это различие несущественно, вместо терминов задача или процесс будем использовать термин стек, имея при этом в виду соответственно задачу или процесс, к которым относится этот стек.

Все стеки, готовые для выполнения на ЦП, но не активные в данный момент по причине отсутствия свободных ЦП, находятся в очереди готовых работ. Эта очередь представляет собой однонаправленный список, голова и хвост которого хранятся в системных ячейках. Стеки в очереди упорядочены по приоритетам. Приоритет приписывается задаче при вводе ее в систему, исходя из многих соображений. Основными из них являются тип задачи (пакетный или диалоговый), требуемые ресурсы по времени и памяти, важность задачи и тому подобное.

Когда на некотором ЦП стек перестает быть активным по некоторым причинам (исчерпание отведенного для него кванта времени, ожидание окончания обмена с внешним устройством (ВУ), ожидание некоторого еще не свершившегося события и так далее), вызывается процедура ОС, которая его деактивизирует, то есть снимает с ЦП и записывает в него из ЦП всю информацию, необходимую для последующей активизации стека (содержимое базовых регистров, содержимое счетчика команд и некоторых триггеров, определяющих характер выполнения некоторых операций). При последующей активизации стека эта информация переписывается обратно из стека в соответствующие регистры ЦП и после этого ЦП начинает выполнять команду, следующую за последней выполненной перед деактивизацией стека.

Судьба снятого с ЦП стека различна в зависимости от причин, по которым он снят. Если исчерпан квант отведенного для его работы времени, то стек готов для дальнейшего выполнения. Поэтому он вставляется в очередь готовых работ. Место в очереди определяется приоритетом стека. Если стек ждет со-

вершения некоторого события, то стек ставится, в очередь к семафору, который будет открыт при наступлении этого события.

Например, если стеку 1 требуется работать с общими данными, с которыми в это время начал работать другой стек 2, то стек 1 ставится в очередь к охраняющему эти данные семафору. При завершении стеком 1 работы с общими данными этот стек открывает охраняющий семафор. При открытии семафора предоставляется возможность всем стекам, в том числе и стекам, ждавшим открытия этого семафора, закрыть его и тем самым монополизировать работу с общими данными. Более подробно эти вопросы рассмотрены в разделе 1.

С учетом приведенных замечаний на Рис.5 показана очередь готовых к выполнению стеков (очередь готовых работ), очередь стеков ждущих открытия некоторого семафора и активные стеки, выполняющиеся на различных ЦП.

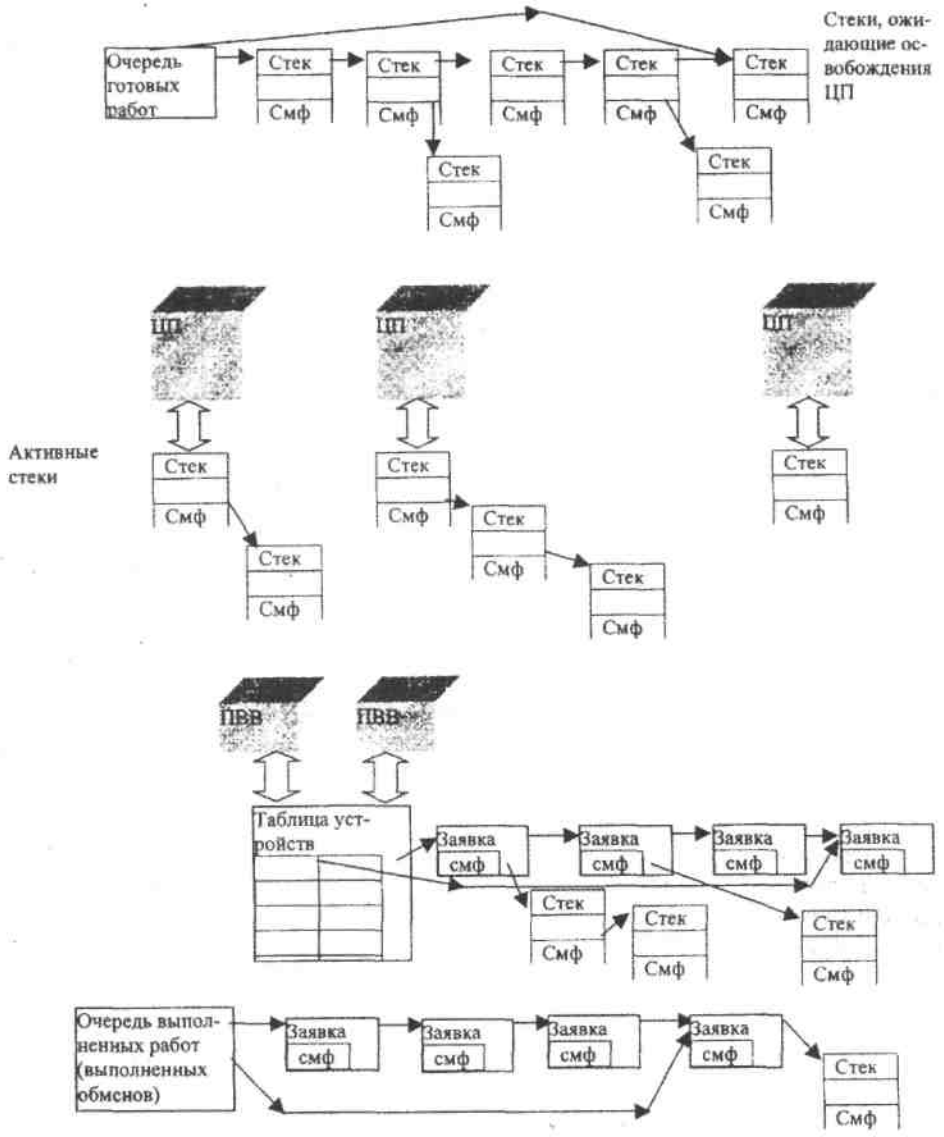


Рис.5. Структура информации о наличных стеках в оперативной памяти.

Для полноты картины на этом же рисунке внизу показаны стеки, ждущие выполнения заказанных ими обменов с внешними устройствами (ВУ), и стеки, ожидающие анализа результатов обменов. Работа с этими стеками описывается в разделе 3.

Максимальное количество стеков в системе определяется ее ресурсами. Необходимо иметь в виду, что каждый стек занимает определенный объем оперативной памяти. Этот объем определяется размерами не только самого стека, но и массивами вне стека, к которым он может обращаться. Известно, что любая задача для эффективного своего выполнения должна иметь в ОП определенное количество данных, называемое рабочей совокупностью (working set).

При обращении к отсутствующим в ОП данным происходит приостановка задачи и подкачка недостающих данных на свободное место в ОП. При этом рабочая совокупность задачи увеличивается. Если такого места нет, то производится откачка из ОП некоторых данных, относящихся к другой задаче. Легко видеть, что у другой задачи рабочая совокупность уменьшается, что может привести при работе этой задачи к откачке данных из другой задачи. В частности, такой задачей может быть первая задача. В результате возникает явление, получившее название трежинг, при котором резко возрастает количество обменов и падает производительность всей системы.

Регулируя количество стеков в системе, ОС следит за тем, чтобы в любое время в системе имелся некоторый запас свободной оперативной памяти. Если рабочая совокупность некоторой задачи расширяется и объем свободной памяти становится меньше минимального предела, ОС временно уменьшает количество присутствующих в системе стеков путем откачки наименее приоритетных стеков и связанных с ними данных на внешние носители. При завершении некоторой задачи или процесса соответствующий стек уничтожается и объем свободной памяти возрастает. При этом откачанные стеки снова могут быть подкачаны в ОП. Если таковые отсутствуют, то ОС вводит в решение новые задачи из очереди подготовленных к решению задач. Предполагается, что поток на вычисления таков, что эта очередь всегда имеется. Таким образом, регулируя количество находящихся в оперативной памяти стеков, ОС обеспечивает наличие в ОП необходимых рабочих совокупностей и таким образом обеспечивает максимальную скорость выполнения задач.

5.2. Распределение оперативной памяти

Выделение оперативной памяти производится по запросам со стороны активных стеков и осуществляется процедурой ДАТЬПАМЯТЬ [7]. В параметрах этой процедуры указывается размер требуемой области ОП и характер запроса (категоричный, по возможности, резидентная память и тому подобное). Алгоритм учета и выделения свободной памяти довольно естественен. Все свободные области памяти объединены в упорядоченный список свободных областей. Упорядочение производится по размеру области. В системе команд имеется специально для этих целей предназначенная команда ПОИСК ПО СПИСКУ, которая находит либо область, точно совпадающую по размеру с требуемым, либо минимально превосходящую по размеру требуемый размер. В первом случае найденная область исключается из списка, загружается в математическую память (если это требуется) и в качестве результата процедуры формируется дескриптор этой области. Во втором случае, если размер найденной области превосходит 15 слов, из найденной области забирается область требуемого размера, а остаток возвращается в область свободной памяти, где он будет включен в соответствующее место списка свободной памяти. Если размер найденной области превосходит требуемый менее,

чем на 15 слов, то остаток в область свободной памяти не отдается. Не выдается он и пользователю. Пользователю выдается дескриптор на область памяти размером, в точности совпадающим с запрашиваемым. Таким образом остаток в этом случае не может быть использован до тех пор, пока выделенная область не будет возвращена обратно в область свободной памяти.

В том случае, если памяти требуемого размера нет, происходит попытка консолидация участков свободной памяти в ОП путем перемещения занятых участков меньшего размера, разделяющих соседние свободные области, в имеющиеся свободные области ОП. Если это не приводит к цели, происходит откачка занятых участков на внешние носители. Для участков с кодами программ (кодовых сегментов) такая откачка не делается, так как на внешних носителях уже имеются такие коды (коды программы в МВК во время ее выполнения остаются неизменными). Нетрудно убедиться, что последовательность указанных шагов приводит в конечном счете к тому, что память требуемого размера будет выделена. Как уже отмечалось, в ОС МВК имеются механизмы, которые следят за тем, чтобы в системе всегда имелась свободная память определенного размера. Поэтому действия, указанные в последнем пункте, связанные с откачкой занятых участков на внешние носители, предпринимаются относительно редко.

Каждая задача имеет свой файл откачки. Каждая занятая область в ОП имеет служебную информацию, в которой указано к какой задаче относится эта область и, следовательно, известно в какой файл откачки необходимо ее откачивать.

Распределение свободной памяти в файле откачки производится по битовой шкале, в которой один разряд соответствует области размером в одну страницу. В отличие от одного общего файла откачки на всю систему, связывание с каждой задачей отдельного файла откачки позволяет резко сократить работу по освобождению памяти на внешних носителях при завершении задачи. Действительно, в последнем случае весь связанный с задачей файл откачки отдается в область свободной памяти на внешних носителях. При этом нет необходимости разбираться, какие из областей файла откачки заняты, а какие свободны.

На процедуры распределения ОП возложена также ответственность по поддержанию необходимой рабочей совокупности каждой задачи. Увеличение объема рабочей совокупности происходит по необходимости при обращении со стороны задачи к данным, содержащимся в массиве, отсутствующим в ОП. Для уменьшения рабочей совокупности в ОС имеется системный процесс, который периодически «отстригает» от каждой задачи определенный процент занимаемой им памяти. Если в процессе решения задачи ее рабочая совокупность расширяется, то запросы на расширение памяти преобладают над «стрижкой». Наоборот, если рабочая совокупность задачи в процессе ее решения становится меньше, то задача реже обращается за новыми, отсутствующими в ОП данными, а «стрижка» продолжается в том же темпе. В результате общий объем оперативной памяти, занимаемой задачей, уменьшается. Таким образом, в системе динамически поддерживается необходимый объем данных для каждой задачи.

Известно, что большинство вычислительных систем с математической памятью (МП) используют страничную организацию МП. При этом обмен данными между ОЗУ и внешними носителями происходит порциями одинакового размера, равными размеру выбранной страницы. При этом ОС и компиляторы стремятся поддерживать максимально возможное заполнения страниц данными. Это приводит к тому, что в одной странице могут оказаться данные, имеющие различную динамическую судьбу. Одни данные в некоторый момент

времени обрабатываются задачей и должны находиться в ОЗУ. В это же время другие находящиеся в этой же странице данные не требуются и их нахождение в ОЗУ не оправдано. Это приводит к дополнительным накладным расходам объема ОЗУ и времени на перекачку избыточных данных. Однако наличие страниц фиксированного размера существенно облегчает задачу учета и распределения оперативной памяти, так как любая страница может быть подкачана на место любой вытолкнутой из ОЗУ страницы.

Известен и другой, так называемый сегментный способ организации математической памяти. В этом случае обмен данными между ОЗУ и внешней памятью производится порциями произвольного размера, называемыми сегментами. Обычно сегмент содержит данные, имеющие общую динамическую судьбу (массив, запись, стек и тому подобное). В этом случае экономится место в ОЗУ и время на передачу данных. Однако при этом существенно усложняется задача учета и распределения свободной памяти в ОЗУ. Кроме того, как показывает статистика, средний размер сегмента относительно невелик (около 100 слов). Специфика аппаратных средств, осуществляющих обмены с дисками и барабанами, такова, что основное время тратится на установку головок (для дисков) и на ожидание поворота носителя до тех пор, пока данные окажутся под головками чтения/записи (для барабанов). Поэтому при обмене порциями данных малого размера резко сокращается средний темп обмена, характеризуемый количеством слов, передаваемых в единицу времени.

В МВК принята смешанная странично-сегментная организация МП |7|, которая объединяет преимущества страничной и сегментной организаций (и их недостатки тоже). Каждая задача имеет свою МП объемом $7*2^{29}$ слов или около 7 миллионов страниц по 512 слов в каждой. Область математической памяти, номера страниц которой имеют в старших разрядах двоичного представления номера страницы код 000, является общей для всех задач. Эта область используется для нужд операционной системы. Каждому очередному запрашиваемому некоторой задачей массиву (сегменту) выделяются очередные страницы МП таким образом, что их общий размер равен либо минимально превосходит размер загружаемого в МП массива.

В ОЗУ все области оперативной памяти, относящиеся к одной задаче (и, следовательно, загруженные в одну и ту же МП), объединены одним списком, голова и хвост которого находятся в документации задачи. Каждая область в ОЗУ содержит в служебной информации номер страницы, в которую она загружена. Поэтому при обращении по математическому адресу просматривается указанный список в поиске указанного в обращении номера странницы. Этот поиск производится одной командой ПОИСК ПО СПИСКУ, специально включенной в систему команд для организации более эффективной работы механизма математической памяти [9]. Если требуемая страница найдена, то к ее физическому адресу прибавляется смещение внутри страницы до требуемого слова. По полученному физическому адресу происходит считывание или запись требуемого слова.

Для ускорения последующих обращений в указанную страницу ее физический и математический адрес записываются в ассоциативную память страниц. Такая память имеется у каждого ЦП.

Кроме того, само слово, к которому происходит обращение, вместе со своим математическим адресом помещается в ассоциативную память чисел, которая также имеется у каждого ЦП. Такая память является аналогом современной КЭШ памяти.

Информация об откачанных или еще не инициализированных страницах помещается в таблицу откачанных страниц (ТОС). Эта таблица состоит из

«головы», находящейся в ОЗУ, и областей переполнения на барабане. Ее величина пропорциональна количеству существующих страниц, а не всей математической памяти.

Для убыстрения работы с ТОС вводится дополнительный буфер, который дублирует в общем случае информацию, расположенную на барабане. Во время работы системы информация об откачанных страницах сначала появляется в виде новых строчек в буфере. В большинстве случаев работа производится с буфером и никаких подкачек ТОС не требуется.

При обращении по математическому адресу к отсутствующей в ОЗУ информации происходит прерывание, которое обрабатывается процедурой ОТСУТСТВИЕВОЗУ. Эта процедура ищет соответствующую строчку в ТОС, где находится соответствие математического адреса файловому адресу, после чего через процедуру ДАТЬПАМЯТЬ заказывается физическая память, в которую считывается информация либо с файла откачки (если обращение относится к отсутствующим данным), либо из программного файла (если обращение относится к отсутствующему программному сегменту). Если же страница неинициализированная, то физическая область прописывается пустышками.

Стремление к тому, чтобы все необходимые действия по выделению оперативной памяти под массив выполнялись как можно позже, привело к использованию техники заявок. Компиляторы с языков программирования транслируют генераторы памяти в команды, оформляющие заявку на память. Заявки на память имеют тег, дающий прерывание при считывании. Обработка этого прерывания поставляет на место заявки готовый к использованию дескриптор.

Обычная последовательность обработки заявки заключается в том, что сна-чала вместо нее появляется дескриптор, описывающий некоторую область из математической памяти, а затем уже при работе с массивом отводится оперативная память с помощью срабатывания процедур управления памятью.

Если при решении чрезвычайно сложных задач возникает ситуация, при которой математическая память задачи оказывается полностью израсходованной, вызывается процедура, которая производит консолидацию всех наличных в это время у данной задачи занятых страниц МП. Математические адреса всех таких страниц уменьшаются таким образом, что нумерация оставшихся неиспользуемых страниц оказывается сплошной без разрывов. При этом производится просмотр всех относящихся к задаче стеков и соответствующим образом корректируется адресная информация, содержащая математические адреса. Это может быть сделано, так как каждое машинное слово в МВК имеет дополнительный описатель-тег, который позволяет отличить слова с адресной информацией (дескрипторы, метки, ссылки).

5.3. Распределение памяти на внешних носителях

Учет и распределение памяти на внешних носителях (барабанах, дисках) производится иначе, чем оперативной памяти [5]. Информация о том, какие области внешней памяти свободны и их размерах содержится не в этих областях, а в так называемых точных таблицах. Точные таблицы создаются при инициализации системы и считываются в ОП при работе с ними. Для каждого барабана или диска имеется одна или несколько точных таблиц, в которых содержится информация о всех наличных областях свободной памяти. Описание одной области занимает одно слово и содержит адрес и размер этой области. Все области в точной таблице упорядочены по адресам. Размеры всех точных таблиц одинаковы и равны 32 словам (минимальному кванту информации, записываемому на барабан или диск). Если информация о свободных областях не умещается в одной точной таб-

лице, то производится расщепление этой таблицы, каждая из которых заполнена наполовину. При этом корректируется грубая таблица, каждое слово в которой является описателем одной точной таблицы. Слово грубой таблицы содержит информацию о номере носителя, к которому она относится, об адресе первой свободной области в соответствующей точной таблице и о максимальном размере области в ней (Рис.6).

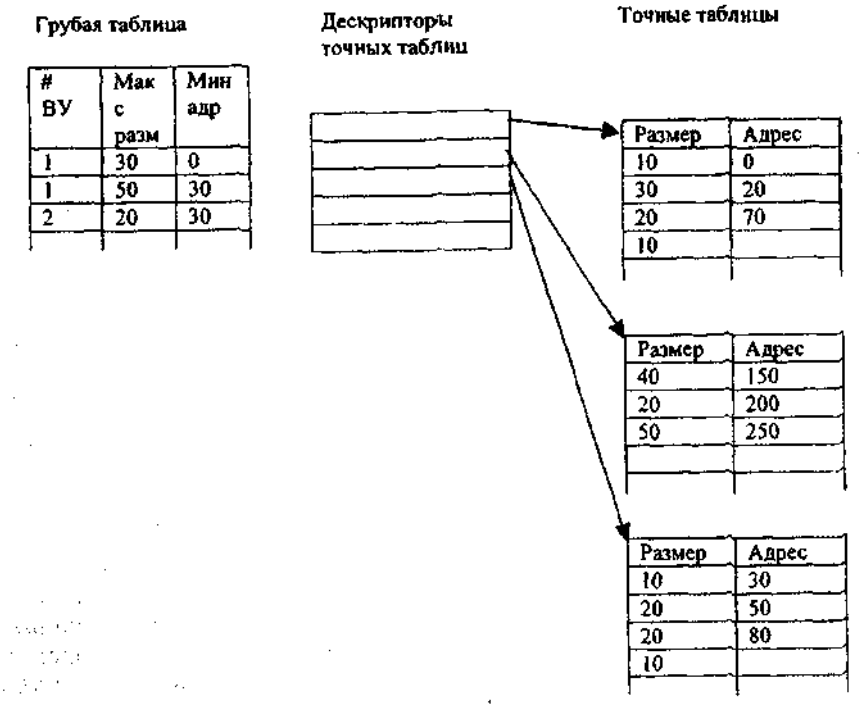


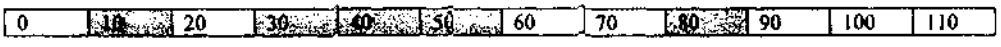
Рис.6. Структура документации о свободных областях на дисках и барабанах.

Выделение свободной памяти на заданном ВУ производится процедурой ДАЙКУСОК, где в качестве параметров указаны номер ВУ, на котором должна быть выделена свободная память, и размер требуемой области. Процедура просматривает грубые таблицы и находит такой описатель точной таблицы, в котором записан размер области, равный или минимально превосходящий требуемый. В этом случае имеется гарантия, что по соответствующей точной таблице будет выделена память требуемого размера. Если найдена грубая таблица с размером, точно совпадающим с требуемым, то в соответствующей точной таблице находится адрес этой области и информация об этой области исключается из точной таблицы. При этом в грубую таблицу записывается максимальный размер области в модифицированной точной таблице. При модификации точная таблица консолидируется, то есть все занятые Ячейки располагаются подряд в начале таблицы. Это облегчает поиск в точной таблице. В том случае, если из точной таблицы оказывается исключенной последняя ячейка, то есть таблица оказывается полностью свободной, такая таблица уничтожается. При этом корректируется грубая таблица. Из нее исключается соответствующая ячейка, описывающая точную таблицу. Оставшиеся ячейки в грубой таблице также консолидируются.

Алгоритм работы процедуры в том случае, когда в точной таблице нет области по размеру, совпадающим в точности с требуемым, является еще более простым. В этом случае в точной таблице находится область, минимально превосходящая по размеру требуемый, и эта область уменьшается на требуемый размер. При этом количество ячеек в точной таблице оказывается неизменным. Отметим, что при выделении области свободной памяти количество областей свободной памяти не увеличивается, а в некоторых случаях может уменьшаться.

Возврат занятых областей на внешних устройствах в область свободной памяти производится процедурой ВОЗЬМИКУСОК. В качестве параметров в процедуре указывается точное местоположение возвращаемой памяти: номер ВУ, адрес начала возвращаемой области на ВУ и ее размер. Сначала процедура по грубым таблицам находит ту точную таблицу, которая должна быть скорректирована. При коррекции возможны случаи, когда количество занятых ячеек в таблице уменьшается, увеличивается, остается неизменным. Алгоритм коррекции точных и грубых таблиц естественен, показан на Рис.7 и не требует дополнительных пояснений.

С целью увеличения объема доступной внешней памяти на съемных носителях введено понятие контейнера, который может объединять произвольное количество дисков. При этом увеличивается соответственно доступная для пользователей внешняя память. Имеются средства, позволяющие выделять память на заданном диске контейнера, в пределах одного цилиндра диска, равномерно распределять память, относящуюся к одному файлу по различным дискам и тому подобное.



Заштрихованы занятые области на барабане или диске

Адрес	Размер
0	10
20	10
60	20
90	30

Первоначальная точная таблица

Модификация точной таблицы:

Возвращена область (30, 10)

Адрес	Размер
0	10
20	20
60	20
90	30

Возвращена область (10, 10)

Адрес	Размер
0	30
60	20
90	30

Возвращена область (50, 10)

Адрес	Размер
0	10
20	10
50	30
90	30

Возвращена область (40, 10)

Адрес	Размер
0	10
20	10
40	10
60	20
90	30

Рис.7. Модификация областей свободной памяти на дисках и барабанах.

Все магнитные барабаны также объединены в один контейнер. При выделении памяти на барабанах ОС следит за тем, чтобы все барабаны были равномерно загружены. Это позволяет выровнять потоки обращений к различным барабанам и тем самым уменьшить среднее время обмена.

6. Реакция ОС на исключительные случаи, возникающие при решении задач

В высокопроизводительной системе, где в процессе решения одновременно находится достаточно большое количество задач, должны быть предприняты специальные меры, чтобы одна из задач не могла каким-либо образом причинить вред другим задачам. В МВК защита данных каждой задачи гарантирована аппаратурой, которая использует для этих целей теги. Какие бы коды ни выполнялись программой в непривилегированном режиме, программа не может обратиться к данным, которые не находятся в ее контексте. Однако при этом могут встретиться операторы с недопустимыми операндами (деление на ноль, индексация массива индексом, большим размера массива, сложение двух меток и тому подобное). В каждой машинной команде аппаратура проверяет правильность типов и значений поданных операндов и при любом несоответствии выдает прерывание. Если пользователь не определил реакцию на такое прерывание, то это приводит к аварийному завершению задачи.

При аварийном завершении должны быть убраны все изменения, сделанные задачей в системе (убраны из системы все следы задачи, то есть закрыты открытые ею файлы, возвращены заказанные массивы в оперативной памяти, завершены инициированные задачей обмены и параллельные процессы, открыты закрытые его системные семафоры) и после этого уничтожен относящийся к задаче стек.

Особое внимание уделяется случаю, когда возникновение ошибки произошло в системной процедуре, выполняющейся на стеке пользователя. В отличие от процедуры пользователя, в контексте системной процедуры находится все системная информация (таблицы, семафоры, очереди, свободная память системы, заявки на обмен и другие). В момент возникновения ошибки заранее неизвестно в какой мере эта информация подвергалась модификации со стороны системной процедуры и поэтому неизвестно что необходимо восстанавливать. Поэтому такая ошибка приводит к полному перезапуску системы. При этом все решаемые в это время задачи перезапускаются заново, если в них не предусмотрены контрольные точки, в которых сохраняются результаты проведенных вычислений. Такие задачи перезапускаются с последней контрольной точки.

Из сказанного следует, что каждая системная процедура, прежде чем предпринимать какие-либо действия с системной информацией, должна тщательным образом проконтролировать подаваемые ей пользователем параметры, так как пользователь имеет возможность подать по ошибке или сознательно любые параметры. Если параметры не соответствуют описанным в интерфейсе системной процедуры, то процедура выдает в стек пользователя, на котором она выполняется, аварийную ситуацию «неверные параметры в системной процедуре». Стандартная реакция на эту ситуацию - аварийное завершение соответствующего процесса. Однако при этом все другие процессы и задачи продолжают выполняться.

При аварийном завершении процесса пользователю выдается распечатка стека, по которой можно проследить всю историю вызова процедур, параметры, с которыми вызвана каждая процедура, место в последней процедуре, откуда была выдана аварийная ситуация. Как показывает опыт эксплуатации МВК, этой информации оказывается вполне достаточно, чтобы определить то место в программе, где произошла аварийная ситуация и установить ее причину.

Из приведенного рассмотрения принципов работы ОС МВК можно сделать вывод, что ОС старается полностью загрузить все ЦП. Что касается внешних устройств, то они загружаются по мере поступления заявок на обмен. Такой подход оправдан, если в МВК выполняются задачи в основном счетного харак-

тера, в которых производятся сложные вычисления над относительно небольшим объемом данных, расположенных главным образом в ОП. Однако имеется и другой класс многочисленных информационных задач, в которых производятся относительно простые вычисления над данными, расположенными на различных ВУ. В этом случае основное время затрачивается на подкачку данных в ОП и в ожидании обменов ЦП могут простаивать. Чтобы ускорить обработку информации в этом случае, могут быть запущены один или несколько процессов, которые стараются не только полностью загрузить ПВВ, но и позволяют уменьшить время обработки считанной в ОП информации за счет уменьшения накладных расходов ОС. Обработка считанной с ВУ информации производится непосредственно на стеках этих процессов. Такой режим работы ОС может быть назван перевернутой ОС [8].

Литература

1. Э.Дийкстра. Синхронизация параллельных процессов. В сб.: Языки программирования под ред. Женюи Ф. М., Мир, 1972.
2. А.П.Иванов, С.В. Семенихин. Операционная система МВК «Эльбрус». Препринт №2. ИТМ и ВТ. М., 1977, 30с.
3. В.П.Торчигин. Система простых файлов МВК «Эльбрус». Препринт №3. ИТМ и ВТ. М., 1977, 20с.
4. С.М.Зотов, СВ. Семенихин. Взаимодействие процессоров в многопроцессорном вычислительном комплексе «Эльбрус». Препринт №1. ИТМ и ВТ. М,, 1981, 20с.
5. В.П.Торчигин. Система файлов МВК «Эльбрус». Препринт №3, ИТМ и ВТ. М., 1981.
6. Л.Е.Пшеничников, Ю.Х.Сахин. Архитектура многопроцессорного вычислительного комплекса «Эльбрус». Препринт №7 ИТМ и ВТ. М., 1977.
7. А.П.Иванов, В.С.Шевяков. Управление памятью в МВК Эльбрус-1. ИТМ и ВТ, 1979.
8. В.П.Торчигин. Организация групповых обменов в МВК «Эльбрус». ИТМ и ВТ. М., 1985.
9. В.С.Бурцев, В.П. Торчигин. Неформальное описание системы команд МВК «Эльбрус». В данной книге.
10. В.С.Бурцев. Принципы построения многопроцессорных вычислительных комплексов «Эльбрус». В данной книге.

Векторный процессор МВК "Эльбрус-2"

В.С.Бурцев, Е.А.Кривошеев, В.Д.Асриэли, П.В.Борисов, К.Я.Трегубов

1. Принципы построения векторного процессора и его работа в составе МВК "Эльбрус-2"

Разработка МВК "Эльбрус-2" в первую очередь была направлена на получение предельной производительности комплекса при решении сложных научных информационно-вычислительных задач в режиме коллективного пользования и задач реального времени. Работы велись в следующих направлениях:

- обеспечение предельной производительности центральных универсальных процессоров за счет распараллеливания вычислителького процесса во времени путем динамического распределения ресурсов, таких как исполнительные устройства, быстрые регистры устройств сверхоперативной памяти и так далее;
- решение проблемы пропорционального повышения производительности комплекса с увеличением числа его универсальных центральных процессоров;
- широкое использование в многопроцессорной системе специализированных процессоров, к которым в первую очередь относится векторный процессор, Для того, чтобы понять особенности использования в многопроцессорной архитектуре наряду с универсальными центральными процессорами и специализированных процессоров, рассмотрим основные принципы их работы прежде всего в части возможности ускорения вычислительного процесса.

Необходимо отметить, что специализированный векторный процессор способен ускорить вычислительный процесс, представляющий собой параллельный граф не связанных между собой однородных операций над данными. Любую операцию над данными (сложение, умножение, деление и так далее) можно разбить на взаимосвязанную последовательность более мелких операций, таких как вычитание порядков, выравнивание порядков, обращение числа, получение полусуммы и переноса и других микроопераций. В том случае, если операции не взаимосвязаны по данным, например необходимо сложить покомпонентно векторы, можно, не завершая операции над первой парой компонент векторов, после окончания какой-либо микрооперации начинать сложение следующей пары компонент векторов и так далее. Если операция сложения разбита на $N_{моп}$ последовательных микроопераций, времена выполнения которых ($\tau_{;моп}$) равны между собой, то через время $N_{моп} * \tau_{;моп}$ все устройства, выполняющие микрооперации, будут загружены работой над компонентами вектора. В этом случае производительность устройства сложения будет в $N_{моп}$ раз выше, чем при работе его с одиночными взаимосвязанными данными.

Эффект возможного увеличения производительности при работе с векторизированными данными может быть увеличен, если рассмотреть одновременное выполнение над векторами не одной, а нескольких различных (в том числе и взаимосвязанных по данным) операций. Так, например, если необходимо произвести следующую последовательность действий над векторами:

$$R := \frac{A \times B + C \times D}{E}$$

где: K, A, B, C, D, E - векторы, имеющие n компонент. Здесь и далее имеются ввиду компонентные операции с векторами.

Можно организовать "конвейер" обработки векторов так , как показано на Рис.1. В этом случае через время $20\tau_{моп}$ загружаются все устройства, выполняющие микрооперации. Предполагается, что устройство умножения выполняет операции за шесть, устройство сложения за четыре, а устройство деления за десять $\tau_{М0П}$, и все они имеют соответствующее количество последовательно работающих блоков микроопераций. Организованный таким образом векторный процессор на приведенном примере будет иметь производительность в 26 раз большую, чем обычный скалярный процессор, последовательно обрабатывающий операции по той же формуле:

$$r := \frac{a \times b + c \times d}{e}$$

где: r, a, b, c, d, e - скалярные величины.

Если даже скалярный процессор будет иметь два устройства умножения и операции умножения будут выполняться одновременно, то все равно он проигрывает векторному процессору в быстродействии в 20 раз. Однако для достижения такой производительности в векторном процессоре необходимо жестко (с точностью до т) синхронизировать подачу операндов и иметь в заданное время свободные для выполнения вычислений устройства. Выполнение этих требований обеспечивается полной монополизацией ресурсов процессора и оперативной памяти в интересах решения одной задачи. Таким образом исключается одновременная работа нескольких процессоров на общую оперативную память и многопрограммный режим с динамическим распределением ресурсов в процессе выполнения задачи. В этих условиях появляется возможность при использовании широкоформатной команды, состоящей фактически из нескольких операций, осуществлять в процессе трансляции (статически) распределение ресурсов во времени с точностью до интервала $\tau_{моп}$ Для вычисления различных векторных выражений, в дальнейшем называемых макросами.

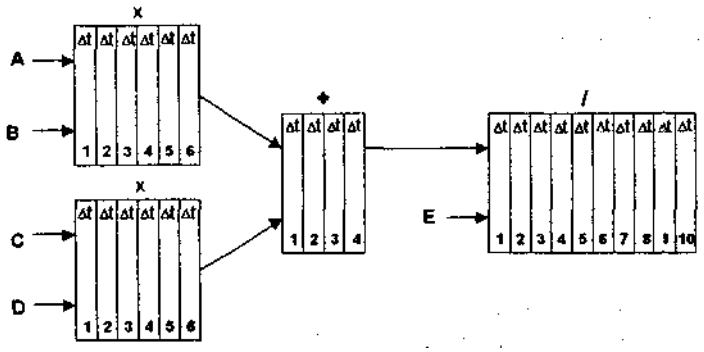


Рис.1. Конвейер обработки векторов.

Неизбежные простои оборудования происходят при переходе от обработки одного макроса к обработке другого. Поэтому, чем длиннее векторы, тем эти потери меньше. Так, при длине вектора в двадцать элементов эффективность использования оборудования для рассмотренного примера не может превышать 50%, а следовательно, и производительность процессора уменьшается в два раза. Потери производительности связаны с простоем оборудования в период заполнения конвейера и его освобождения, другими словами - во время "разгона" и "торможения". Безусловно, эти времена зависят от времени считывания вектора из памяти - времени доступа к памяти ($T_{дост}$). Так, в CRAY-1 с целью сокращения времени разгона и торможения между памятью и процессором введены восемь прямоадресуемых быстрых векторных регистров (на 64 элемента каждый), что существенно повысило производительность комплекса на коротких векторах по сравнению с ЭВМ Cyber-205 [13, 14, 15, 16]. Однако введение прямоадресуемых регистров в таком малом количестве ограничило возможности GRAY-1 в части создания различных конфигураций конвейера, адаптированных под макросы, и сильно усложнило программирование. Поэтому более предпочтительной является работа процессора непосредственно на локальную оперативную память при условии, что времена разгона и торможения будут значительно сокращены.

Используя жесткую детерминированность системы во времени, можно возложить на транслятор задачу распределения ресурсов аппаратуры на время выполнения макроса, а также задачу подготовки следующей конфигурации процессора на фоне работы старой. Имеется в виду предварительный выбор операндов из локальной памяти и использование исполнительных устройств по мере их высвобождения - совмещение во времени этапа торможения с этапом разгона нового макроса.

Необходимо отметить, что эффективность этого метода во многом определяется качеством транслятора, так как распределение ресурсов процессор-память осуществляется во время трансляции до выполнения программы. Задача транслятора существенно усложняется при малых размерах векторов, когда $n < N_{моп}$ где n - количество элементов вектора, выполняемого макроса, а $N_{моп}$ - число микро-операций конвейера, адаптированного под этот макрос. При n = 1 эффективность этого метода существенно снижается, так как для достаточно полного использования конвейера ежетактно должна происходить реконфигурация конвейера.

Необходимо обратить внимание еще на одну особенность работы векторного процессора. Предположим, что работает конвейерная конфигурация, изображенная на Рис.1, тогда за каждый такт ($\tau_{моп}$) из памяти должно быть считано пять чисел и записано одно. Если $\tau_{моп}$ = 12,5 нс (как в CRAY-1), то темп обращения к памяти должен быть $\approx$ 2 нс.

Как известно, для решения больших задач требуется память в десятки миллионов слов. Реализовать память требуемого объема, работающую с темпом в единицы наносекунд, не потеряв существенно во времени доступа к памяти ($T_{дост}$), практически невозможно. Поэтому встает вопрос об объеме сверхоперативной памяти векторного процессора, который должен быть таким, чтобы за счет повторного использования данных, находящихся в ней, темп обмена со следующим уровнем памяти был хотя бы на порядок меньше.

Если в качестве аналога взять скалярный процессор, то это условие выполняется на объеме сверхоперативной памяти, памяти КЭШ, в несколько тысяч слов. Соответственно для векторного процессора при средней длине вектора 100 элементов можно предположить объем сверхоперативной памяти, равный нескольким сотням тысяч слов. Таким образом, векторный процессор должен иметь сверхоперативную память, значительно большую, чем память скалярно-

го процессора. Естественно, что столь различные методы организации вычислительных процессов для скалярного и векторного счета требуют принципивычислительных средств, а именно:

- скалярный процессор должен иметь минимальное время выполнения операций и время доступа к памяти. Векторный процессор должен, в первую очередь, иметь максимальный темп вычислений и максимальную пропускную способность памяти [1];

- аппаратные средства скалярного процессора должны обеспечивать эффективную работу его в мультипрограммном режиме. Векторный процессор должен быть ориентирован на решение больших задач в монопрограммном режиме;

- наиболее эффективным способом распределения ресурсов (исполнительных устройств, регистров и так далее) скалярного процессора, как показала практика эксплуатации МВК "Эльбрус", является динамический (в процессе выполнения программ), реализуемый, как правило, аппаратными средствами. Эффективное выполнение программ на векторном процессоре требует статического распределения ресурсов во время трансляции программ, что должно обеспечиваться совершенно другими аппаратными решениями при построении процессора.

Совмещение в одном процессоре столь противоположных требований приведет к малосовместимым схемотехническим конструкциям построения процессора. Неизбежные компромиссные решения могут приниматься только за счет снижения предельной производительности либо векторной, либо скалярной части процессора. Поэтому решение задачи достижения предельной производительности наиболее полно может быть реализовано на многопроцессорном комплексе с широким использованием наряду с универсальными процессорами (ориентированных на скалярные вычисления) специализированных векторных процессоров. Этот принцип векторизации МВК "Эльбрус-2" был выбран при его создании.

Аналогом подобного принципа построения комплекса может служить суперЭВМ Cyber - 205, которая в полной комплектации имеет четыре векторных и один универсальный процессор. МВК "Эльбрус-2" с векторными процессорами имел бы определенные преимущества перед Cyber - 205 за счет возможности создания комплекса, состоящего из любого сочетания векторных и скалярных процессоров, и наличия при каждом векторном процессоре большой сверхбыстродействующей локальной памяти. Благодаря этому производительность МВК "Эльбрус-2" с векторным процессором на многих задачах была бы соизмеримой с производительностью Cyber - 205, несмотря на то, что элементная база последнего почти в три раза более быстродействующая.

Архитектурные особенности векторного процессора должны учитываться при работе его в комплексе. Так, если загрузка центральных процессоров в МВК "Эльбрус-2" осуществляется в процессе выполнения вычисления за счет аппаратной реализации принципа их обезличенной работы, то загрузка векторного процессора должна осуществляться комплексом посредством программного управления. Для обеспечения возможности жесткой синхронизации данных, поступающих на входы исполнительных устройств, а также для существенного снижения темпа работы с общей памятью, векторный процессор должен сопрягаться с комплексом через локальную память достаточно больших размеров. Блок-схема векторного процессора в составе МВК "Эльбрус-2" показана на Рис.2.

При проектировании векторного процессора чрезвычайно важно было определить необходимые объемы памяти Q_1, Q_2 с тем, чтобы были удовлетворены приемлемые для решения большинства задач-соотношения

$K_1 = E_1 / E_2$ и $K_2 = E_2 / E_3$.

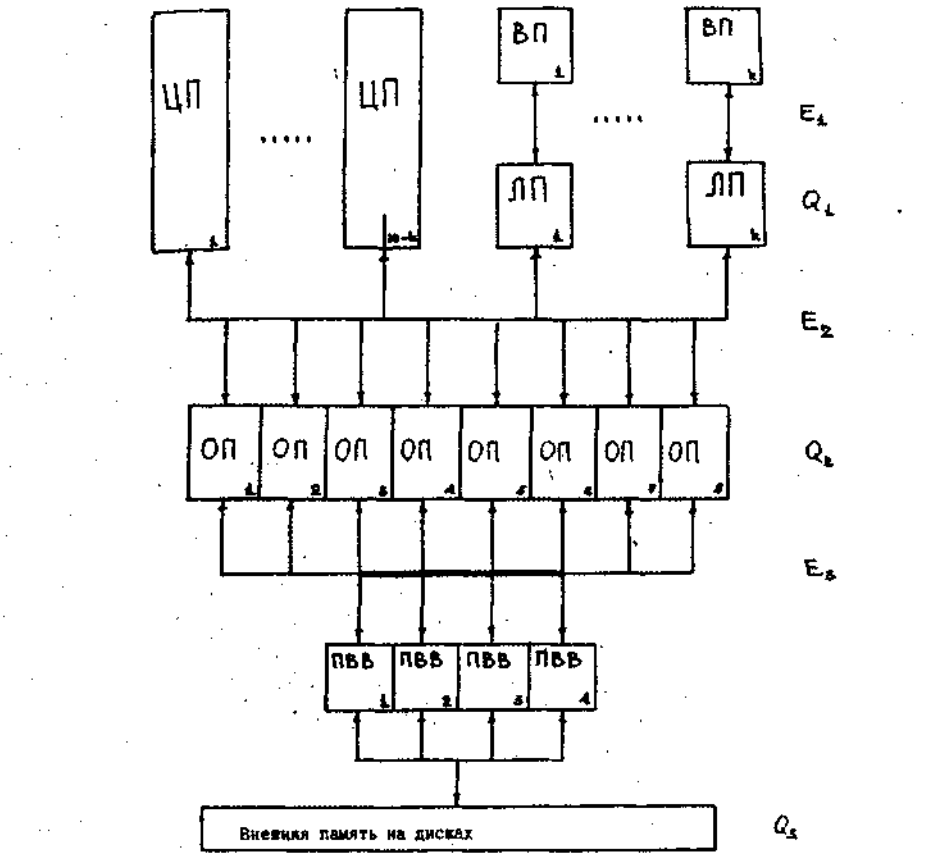


Рис.2. Блок-схема МВК «Эльбрус-2» с векторным процессором.
ЦП - центральный процессор; ВП - векторный процессор; ЛП - локальная память векторного процессора; ОП - оперативная память; Е1 , Е2 , Е3 - средняя пропускная способность каналов связи;Q1 , Q2 , Q 3 , объемы локальной, оперативной и внешней памяти.

Максимальная пропускная способность векторного процессора E_1 = 1 слово/5 нс может быть определена из работы ВП при максимальной загрузке его исполнительных устройств (Рис.3). Величина E_2 не должна превышать максимального темпа обмена центрального процессора (E_2 = 1 слово/80 нс) |3]. Таким образом, K_1 должно быть больше 16.

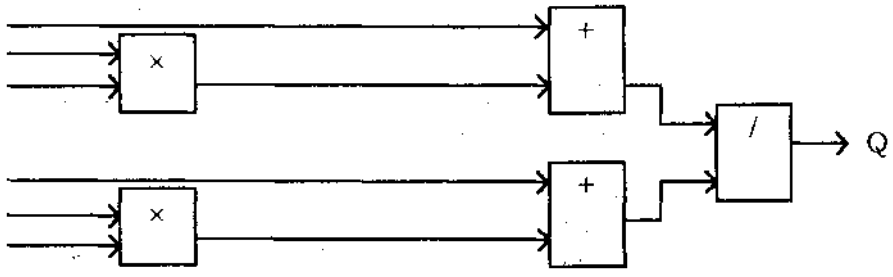


Рис.3. Конфигурация конвейера векторного процессора, предельно использующая динамический ресурс оперативной памяти [1].

Если допустить, что на один векторный процессор может быть обеспечена одновременная работа пяти дисков, то Ез будет определено общим средним темпом обмена в 2 мкс на слово и величина K_2 должна быть более 25. Возможность выполнения этих соотношений была проверена путем моделирования задачи Навье-Стокса [7], где и определились объемы локальной Q_1 = 500 тыс. слов и оперативной Q_2 = 16 млн. слов памяти.

Выделение векторных вычислений в отдельный процессор имеет, безусловно, и отрицательные стороны, одна из которых состоит в том, что не бывает чисто векторных или чисто скалярных вычислений, поэтому могут возникнуть потери при передаче данных от векторного процессора к скалярному и обратно. Уменьшение этих потерь способствуют следующие факторы:

а) в центральном процессоре МВК "Эльбрус-2" реализована аппаратная поддержка выполнения векторных вычислений, за счет чего производительность на циклах обработки векторов увеличена на 10-20%, а векторный процессор имеет возможность выполнять на тех же исполнительных устройствах скалярные вычисления;

б) время передачи данных от векторного процессора к скалярному и обратно максимально сокращено: передача 10 тыс. слов из векторного процессора в центральный и обратно занимает всего 1 млс (максимальная пропускная способность всей оперативной памяти 5 нс[3]);

в) успешно ведутся работы по повышению степени векторизации алгоритмов с одновременной их локализацией. В этом случае скалярная часть вычислений может производиться на фоне векторной [10,11].

При разработке векторного процессора были предложены новые оригинальные архитектурные решения, существенно повышающие эффективность использования его исполнительных устройств, к которым в первую очередь относятся: аппаратно-программные принципы реконфигурации процессора с целью адаптации его к выполнению различных макросов; совмещение во времени подготовки последующей конфигурации процессора на этапе завершения работы предыдущей конфигурации - совмещение во времени этапов разгона и торможения; решение задачи выполнения команды "считывание вектора" непосредственно после команды "запись" и так далее.

Новым и безусловно оригинальным является язык высокого уровня, обеспечивающий наряду с естественностью описания алгоритмов, эффективное использование оборудования векторного процессора. Разработаны новые методы организации оптимизирующего транслятора. Разработка выполнена с учетом опыта отладки и эксплуатации больших вычислительных систем [5,6].

Все эти новые решения, безусловно, будут широко использованы в перспективных отечественных разработках суперЭВМ.

2. Архитектурные особенности векторного процессора

Наибольшее практическое использование в настоящее время получили два векторных процессора CRAY-1 и Cyber - 205. Поэтому целесообразно особенности построения векторного процессора относить именно к этим архитектурным решениям [13,14,15,16].

Как уже было показано, одним из методов повышения производительности векторного процессора является организация параллельной работы его исполнительных функциональных арифметических устройств за счет соединения их в обрабатывающий конвейер так, чтобы выходы одних были непосредственно подключены к выходам других устройств (Рис.1). В дальнейшем этот метод будем называть зацеплением.

В CRAY-1 имеются три векторных функциональных устройства, которые могут быть зацеплены. Зацепление устройств происходит аппаратно (программист об этом может и не знать), поэтому промежуточные результаты, несмотря на зацепление, должны записываться в векторные регистры, что снижает общую производительность векторного процессора.

В Cyber - 205 высокая производительность достигается за счет параллельной работы нескольких обрабатывающих конвейеров (до четырех). В самом обрабатывающем конвейере можно осуществить зацепление двух устройств: для этого требуется явное указание в программе - команда LINK перед двумя зацепляемыми операциями.

В рассматриваемом векторном процессоре (ВП) возможна одновременная работа до шести функциональных устройств:

- двух устройств сложения;
- двух устройств умножения;
- устройства деления;
- логического устройства.

В векторном процессоре программирование какой-либо формулы осуществляется не последовательностью операций, а заданием соединений между набором функциональных устройств, выполняющих эти операции. Входы каждого из шести устройств могут быть подключены непосредственно к выходу любого устройства. Для покомпонентного вычисления, например, векторного выражения

$$G := \frac{A + B \times C}{D + E \times F} \qquad (1)$$

где A,B,C,D,E,F,G - векторы, имеющие n-компонент, можно построить обрабатывающий конвейер, конфигурация которого приведена на Рис.3. Общая производительность ВП в этом случае 5/ τ., где τ - темп работы каждого функционального устройства.

Векторный процессор, как и Cyber-205, работает непосредственно "из памяти в память" (операнд считывается из ОП и результат пишется в ОП), что существенно расширяет его возможности и облегчает программирование по сравнению с CRAY-1. Однако в отличие от Cyber-205, в МВК "Эльбрус-2" каждый процессор ВП имеет свою локальную память - память векторного процессора, которая по стандартному каналу процессор-память соединена с общей оперативной памятью последнего (Рис.2). Благодаря этому в ВП имеется шесть потоков считывания из памяти и два потока записи в память. Этого достаточно для построения практически любых разумных конфигураций из имеющегося набора функциональных устройств.

Два потока записи позволяют построить два параллельно работающих обрабатывающих конвейера (Рис.4) для вычисления, например, двух выражений: D:=AxB+C и H:=ExF+C.

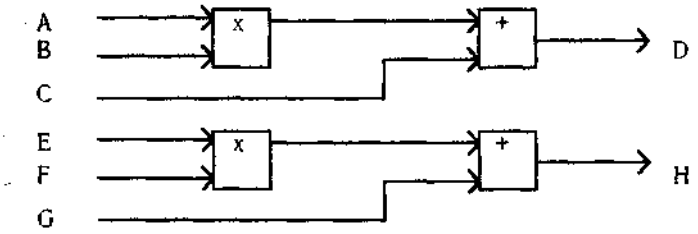


Рис.4. Два параллельно работающих конвейера.

Пропускная способность памяти векторного процессора, которая должна обеспечивать суммарный поток обрабатывающего конвейера, достигается расслоением памяти (интерливингом) по младшим разрядам адреса на модули, работающие параллельно.

Чтобы не накладывать ограничения на взаимное расположение в памяти векторов, с которыми работает обрабатывающий конвейер, потоки обслуживаются поочередно, порциями по к чисел, где к - число модулей памяти. Но это возможно только при определенном шаге, с которым компоненты вектора размещены в памяти.

В Cyber-205 нельзя обращаться одновременно к модулям памяти по независимым адресам, поэтому связь обрабатывающего конвейера с памятью осуществляется суперсловами с шагом, равным единице. Это существенное ограничение, которое снижает эффективность при работе с многомерными массивами.

В памяти ВП реализовано обращение к модулям по независимым адресам - тем самым существенно ослабляются требования к шагу вектора (имеется в виду разность адресов последовательных компонент вектора). При количестве модулей памяти, равном степени двойки, шаг вектора должен быть числом нечетным. Это гарантирует расположение любых последовательных к компонент вектора в различных модулях памяти. Такая организация взаимодействия с оперативной памятью облегчает эффективную работу с многомерными массивами данных по всем измерениям. Например, если необходимо работать со строками и столбцами матрицы, то, возможно, потребуется завести фиктивный столбец с тем, чтобы шаг по строке и шаг по столбцу были нечетными числами.

Для каждого потока считывания и записи в ВП имеется свой буфер. На буфер входного потока из памяти приходит одновременно k чисел, а выдается на вход какого-либо функционального устройства по одному числу за такт. В буфере выходного потока, наоборот, накапливается по одному слову с выхода функционального устройства, а выдается на запись в память по k чисел.

Одним из существенных недостатков Cyber-205 является малая эффективность при работе с короткими векторами. Так, при длине векторов n, равной 25, Cyber-205 развивает 20% от максимальной производительности, при длине 100 - около 50%, при длине 1000 - около 90%. Такая асимптотическая зависимость является следствием большой инерционности конвейера и потерь при разгоне и торможении.

В векторном процессоре особое внимание было уделено сведению к минимуму потерь производительности, связанных с освобождением конвейерных устройств по окончании работы на одной конфигурации (фаза торможения) и заполнением конвейера другой конфигурации (фаза разгона) за счет введения программно-аппаратных средств совмещения фаз разгона и торможения, что увеличивает эффективность ВП при работе с короткими векторами.

Рассмотрим функциональную схему векторного процессора (Рис.5) с целью пояснения реализации новых принципов обработки векторной информации.

В обрабатывающий конвейер входят:

- шесть конвейерных арифметических функциональных устройств (СМО, СМ1, УМО, УМ1, Д, ЛОГ);
- шесть буферов входных потоков (БВХПО, ... , БВХП5);
- два буфера выходных потоков (БВЫХПО, БВЫХП01);
- четыре вещественных регистровых файла (РФО,...., РФ3), предназначенных для хранения констант и скалярных переменных.

Связи между перечисленными устройствами при построении обрабатывающего конвейера устанавливаются конфигурантом (КНФ), который позволяет соединить каждый вход функциональных устройств, регистровых файлов или буферов выходных потоков с выходом любого функционального устройства, регистрового файла или буфера входного потока.

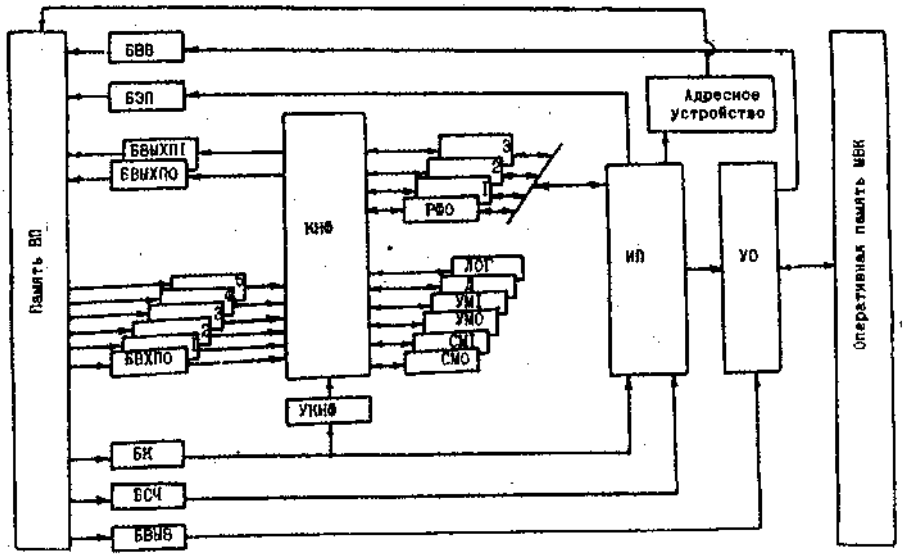


Рис.5. Функциональная схема векторного процессора.

Управляет конфигурантом устройство управления конфигурацией (УКНФ) в соответствии с потоком команд изменения конфигурации, поступающих из буфера команд (БК). Остальные команды поступают из БК в индексный процессор (ИП), который представляет собой универсальный целочисленный процессор, предназначенный для организации вычислительного процесса, предварительной обработки адресных команд и команд обмена. Он имеет 32-раз-рядное арифметико-логическое устройство, собственную внутреннюю память емкостью 1К слов, которую можно переписывать в память ВП через буфер записи (БЗП) и загружать из памяти ВП через буфер считывания (БСЧ). Кроме того, ИП имеет доступ к вещественным регистровым файлам.

Устройство обмена (УО) осуществляет обмен между памятью ВП через буферы ввода (БВВ) и вывода (БВЫВ) и оперативной памятью МВК, к которой УО подключается через стандартный интерфейс вместо одного из центральных процессоров МВК.

Память ВП при цикле 40 не имеет расслоение на восемь независимо адресуемых модулей. Запросы к памяти ВП обслуживаются в соответствии с их приоритетами. Высшим приоритетом обладают потоки записи, затем идут поток команд, потоки считывания, потоки откачки и закачки памяти индексного процессора, поток обмена.

Адресное устройство (АУ) предназначено для формирования физических адресов обращения к модулям памяти ВП от всех абонентов с использованием для формирования восьмерки адресов текущего начального адреса и шага обслуживаемого в данный момент потока.

В векторном процессоре приняты следующие представления чисел, система адресации и система команд.

1.*Представление чисел.* Для представления чисел в системе команд ВП имеются два формата:

- 64-разрядный формат чисел с плавающей запятой;
- 32-разрядный формат целых чисел.

Все конвейерные функциональные устройства работают только с 64-разрядными числами.

2. *Система адресации.* Векторный процессор обладает памятью двух типов: оперативной векторной памятью и регистровыми файлами. К регистровым файлам может обращаться как индексный процессор, так и обрабатывающий конвейер.

Информация в векторной памяти располагается массивами. Программирование всех обращений в векторную память ведется в математических адресах. Физические адреса команд и данных формируются путем суммирования математического адреса со значением соответствующего регистра базы.

3. *Типы команд.*

а) Команды индексного процессора включают:

- целочисленные арифметические операции;
- логические операции;
- команды условных и безусловных переходов;
- команды пересылок, адресуемые к регистровым файлам;
- команду загрузки счетчика длины вектора;
- специальные команды аппаратной поддержки системы программирования

б) Адресные команды позволяют адресоваться к скалярам, одномерным и двумерным массивам. При адресации к одномерному массиву задаются начальный адрес, шаг и количество, при адресации к двумерному массиву задаются начальный адрес, шаг по первому измерению, количество по первому измерению, шаг по второму Измерению и количество по второму измерению.

в) Команда обмена, так же как и адресная пара, задает массив обмена в векторной памяти и аналогично массив обмена в оперативной памяти МВК, а также направление обмена (ввод-вывод).

г) Команды изменения конфигурации описывают во времени процесс изменения конфигурации обрабатывающего конвейера.

Рассмотрим для примера Программу вычисления выражения

$D := (A + B) \times C,$

где A, B, C, D - векторы длины n. Программа состоит из трех частей.

В первой части содержатся адресные команды для вычисления последовательности адресов компонент каждого вектора и загрузки счетчика длины вектора.

Вторая и третья части состоят из команд изменения конфигурации: вторая часть описывает фазу разгона - последовательный процесс построения конфигурации для вычисления данного выражения, третья описывает фазу торможения - последовательный процесс освобождения устройств по окончании вычисления выражения. В дальнейшем такую программу вычисления векторного выражения, содержащую перечисленные три части, будем называть макросом. Обрабатывающий конвейер для вычисления макроса по заданной формуле приведен на Рис.6.

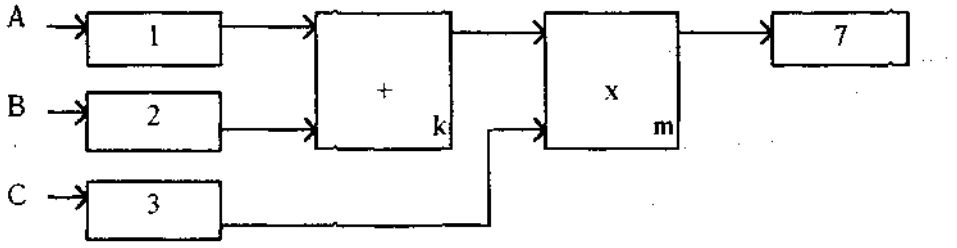


Рис.6. Конвейер вычисления макроса.

Пусть длина конвейера устройства сложения равна к тактам, длина конвейера устройства умножения - m тактам. Фазу разгона макроса можно описать следующим образом.

1. Подключить входы устройства сложения к буферам входных потоков 1 и 2, разблокировать потоки 1 и 2.
2. Задержка на k тактов.
3. Подключить первый вход устройства умножения к выходу устройства сложения, второй вход устройства умножения - к буферу входного потока 3, разблокировать поток 3.
4. Задержка на m тактов.
5. Подключить выход устройства умножения к буферу выходного потока 7, разблокировать поток 7.

Через n тактов после начала фазы разгона должна быть запущена фаза торможения макроса, которую можно описать следующим образом.

1. Заблокировать выдачу из буферов входных потоков 1 и 2, освободить устройство сложения по входу.
2. Задержка на k тактов.
3. Заблокировать выдачу из буфера входного потока 3, освободить устройство умножения по входу.
4. Задержка на m тактов.
5. Освободить по входу буфер выходного потока 7.

Команды изменения конфигурации включают:

а) команды частичного изменения конфигурации фазы разгона (ЧИК), они описывают во времени динамику изменения конфигурации фазы разгона. В команде указано, какие связи должны быть установлены между устройствами и сколько тактов установленная конфигурация не должна меняться;

б) команда временной диаграммы фазы разгона макроса (ВДФР), в ней для каждого задействованного в обрабатывающем конвейере устройства указана задержка в тактах от начала фазы разгона до момента, когда данное устройство потребуется в конфигурации;

в) команда временной диаграммы фазы торможения макроса (ВДФТ), в ней для каждого задействованного в обрабатывающем конвейере устройства указана задержка в тактах от начала фазы торможения до момента освобождения по входу каждого устройства.

Команда ЧИК действует локально, то есть изменяет только те связи в обрабатывающем конвейере, которые явно указаны в ней (во всем остальном конфигурация остается прежней). Свойство локальности позволяет совмещать независимые фазы разгона и торможения. Фаза разгона макроса обеспечивает постепенное построение конфигурации - от буферов входных потоков до необходимых функциональных устройств именно в тот момент, когда они начинают фактически использоваться. Это позволяет начинать строительство следующей конфигурации, не дожидаясь окончания вычисления по предыдущей конфигурации, и использовать устройства в следующем такте после их освобождения по входу предыдущей конфигурации.

Команды временных диаграмм фаз разгона и торможения специально введены для реализации возможности аппаратного совмещения двух смежных макросов. Фаза торможения макроса начинается через n тактов после начала фазы разгона, где n- длина вектора.

Естественным требованием является независимость программы изменения конфигурации макроса от длины вектора. Однако на коротких векторах возможно совмещение фазы разгона и фазы торможения макроса. Эта ситуация графически изображена на Рис.7а. В интервале Д надо выполнять одновременно два потока команд изменения конфигурации.

Чтобы совместить во времени фазу торможения текущего макроса с фазой разгона следующего макроса, одновременного выполнения только двух пото-

ков команд изменения конфигурации уже недостаточно, так как возможна ситуация, изображенная на Рис.76. В интервале Δ должны выполняться одновременно три потока команд изменения конфигурации.

В векторном процессоре реализованы одновременно три потока команд изменения конфигурации. При этом разгон следующего макроса может начинаться после завершения фазы разгона текущего макроса, то есть длина векторов, на которых возможна работа без потерь, равна длине обрабатывающего конвейера (порядка 20-30).

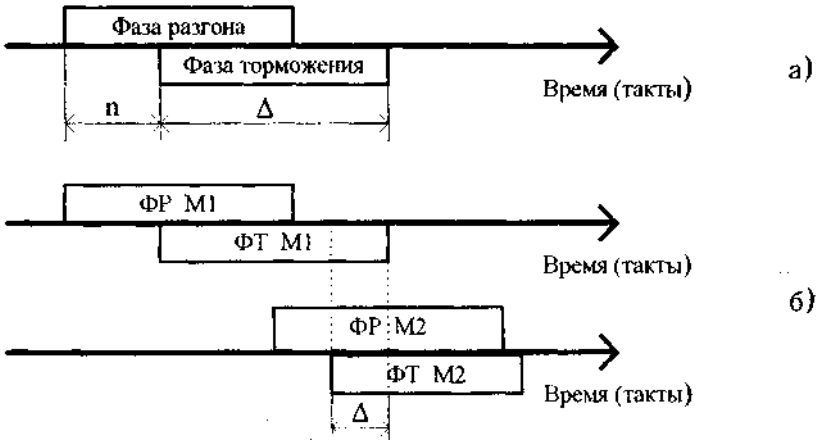


Рис.7. Временные диаграммы работы векторного процессора на этапе разгона и торможения: а) - один макрос; б) - два последовательных макроса.

Как и для любого конвейера, работающего "из памяти - в память", в ВП возникает проблема считывания после записи (ПСПЗ), суть которой состоит в следующем. Пусть макрос использует результат предыдущего макроса. Для правильной работы необходимо, чтобы исполнение второго макроса не начиналось до тех пор, пока не опустошится конвейер и все результаты первого не запишутся в память, то есть совмещение разгона и торможения таких двух макросов недопустимо.

В ВП проблема СРЗ решается программно-аппаратным способом. Для каждого массива в адресной команде содержится информация о диапазоне изменения адресов (Д). Система программирования гарантирует, что адреса всех элементов массива содержатся в отрезке [АН, АН+Д], где АН - адрес начала массива. Если отрезок адресов массива считывания следующего макроса пересекается с отрезком адресов массива записи предыдущего макроса, то считывание элементов данного массива аппаратно откладывается до тех пор, пока все результаты предыдущего макроса не будут записаны в память. В остальных случаях совмещение макросов разрешается. Сравнение отрезков производится аппаратно, до обращения в память за операндами.

Принятые архитектурные решения обеспечивают на элементно-конструкторской базе МВК "Эльбрус-2" (τ = 40 нс) максимальную производительность процессора равную 125 мегафлопам в секунду.

Эксплуатационный такт МВК "Эльбрус-2" τ = 44 нс. Расчетный такт работы векторного процессора -τ = 40 нс. При моделировании работ векторного процессора на уравнениях Навье-Стокса было принято τ = 44 нс.

Реальная производительность была оценена на моделировании решения уравнений Навье-Стокса в трехмерной области. На этой же задаче были уточнены необходимые объемы векторной и оперативной памяти (Q_1 и Q_2) и требуемые пропускные способности каналов связи (E_1, E_2 и E_3) (Рис.2).

3. Результаты моделирования решения уравнений Навье-Стокса в трехмерной области на ВП

Выбор задачи решения уравнений Навье-Стокса в трехмерной области в качестве тестовой для векторного процессора был обусловлен следующими основными факторами.

Во-первых, существует и является актуальной проблема решения трехмерных уравнений Навье-Стокса с увеличенным числом узлов, недоступным пока для существующих ЭВМ.

Во-вторых, в настоящее время разработаны алгоритмы распараллеливания решения трехмерных уравнений Навье-Стокса, пригодные для реализации на векторном процессоре [8,9].

В-третьих, задача Навье-Стокса носит черты типичной задачи с регулярной сеткой, что придает данному опыту программирования методическую ценность.

Программирование задачи Навье-Стокса было выполнено на базовом языке программирования и преследовало следующие цели:

1) оценить возможность конвейерного распараллеливания задачи;
2) оценить объем векторной памяти, необходимый для эффективной реализации алгоритма;
3) оценить время решения задачи;
4) оценить эффективность использования аппаратуры процессора при решении задачи.

Следует отметить, что решение такой задачи на сетке 10x10x15 узлов на БЭСМ-6 занимает примерно один час, причем увеличить число узлов не позволяет малая емкость ее оперативной памяти. Решение задачи на БЭСМ-6 на сетке 50x50x50 узлов практически невозможно, а именно такие задачи становятся весьма актуальными, и эта проблема сейчас рассматривается. Алгоритм решения основан на методе расщепления по пространственным переменным и физическим процессам и использует алгоритм, разработанный в ИТМ и ВТ АН СССР [8,9].

В алгоритме специальным преобразованием координат физическая область с неравномерной сеткой приводится к кубу со стороной N с равномерной сет-кой, в котором ведется счет на установление. Каждый временной шаг решения разбивается на шесть последовательных дробных шагов. Шесть дробных шагов включают в себя по два шага в каждом из трех ортогональных направлений. Уравнения каждого дробного шага включают в себя первые и вторые частные производные неизвестных лишь по одной пространственной переменной. Таким образом, выполнение одного дробного шага сводится к независимому решению N x N одномерных задач.

Для решения используется неявная разностная схема с пространственным семиточечным шаблоном, каждый дробный шаг сводится к решению от одной до пяти систем линейных уравнений с трехдиагональной матрицей

$$B_1 Y_1 + C_1 Y_2 = F_1,$$

$$A_1 Y_{i-1} + B_i Y_i + C_i Y_{i-1} = F_{i:} \quad i = 2, \ldots\ldots, N\text{-}l,$$

$$A_N Y_{N-1} + B_N Y_N = F_N.$$

Коэффициенты и правые части этих уравнений могут вычисляться независимо друг от друга и параллельно по всей расчетной области. Расчет коэффициентов и правых частей включает вычисление первых и вторых производных от известных переменных. Трехдиагональные системы уравнений решаются методом скалярной прогонки в каждом дробном шаге, причем независимо могут выполняться N^2 скалярных прогонок.

Структура процессора накладывает определенные ограничения на возможности распараллеливания алгоритма. Ограниченная емкость векторной памяти не позволяет помещать в нее одновременно все данные задачи, и поэтому данные хранятся в памяти второго уровня, а в векторную память вызывается информация, относящаяся только к некоторому слою расчетной области, и векторный процессор ведет счет по слоям. На Рис.8 куб изображает расчетную область, а выделенные внутренние плоскости изображают данные, хранящиеся в векторной памяти.

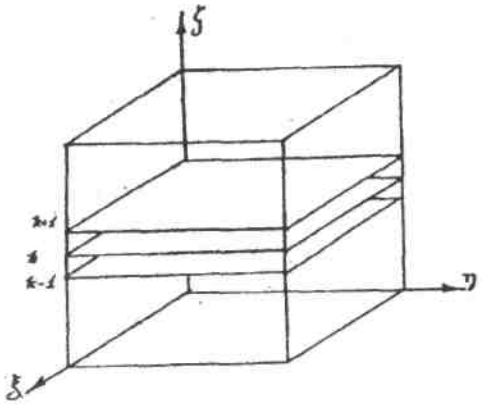


Рис.8. Отображение данных, хранящихся в векторной памяти.

Пусть 1-й дробный шаг связан с производными от неизвестных по ξ,, 2-й - по η, 3-й - по ζ, 4-й - снова по ξ, 5-й - по *η,* 6-й - по *ζ*,. Тогда с участием данных k-1 и k+1 плоскостей для k-й плоскости вычисляются коэффициенты и правые части трехдиагональных систем 1-го дробного шага. Для этих вычислений требуются три смежные параллельные плоскости, поскольку частные производные первого и второго порядка вычисляются не только по ξ и *η,* но и по нормали, то есть вдоль ζ.

Для вычисления коэффициентов и правых частей уравнений 1-го дробного шага ведется конвейерный счет по матрице k-й плоскости. Трехдиагональная система решается экономичным методом скалярной прогонки. В k-й плоскости необходимо выполнить 50 скалярных прогонок вдоль *ξ.*

Отдельная скалярная прогонка плохо поддается конвейерному распараллеливанию, поскольку и прямой ход прогонки

$$\alpha_{i+1} = - C\,i/A_i\,\alpha_i + B_i\,, \qquad \beta_{i+1} = (F_i - A_i\beta_i) / (A_i\alpha_i + B_i)$$

и обратный

$$Y_{i-1} = \alpha_i Y_i + \beta_i$$

используют счет по рекуррентным формулам.

Однако одновременное выполнение многих (пятидесяти) независимых прогонок позволяет применить конвейерное распараллеливание.

Таким образом проводится "вектор" скалярных прогонок вдоль *ξ* , (сначала прямых, а затем обратных). На этом вычисление 1-го дробного шага заканчивается.

Вычисление 2-го дробного шага ведется вдоль *η* на той же k-й плоскости. Аналогично 1-му дробному шагу вычисляются коэффициенты, а затем проводится прогонка вдоль η.

Введя в векторную память информацию следующей k +2 плоскости, можно провести расчет 1-го и 2-го дробных шагов на следующей k+1 плоскости. Переходя от одной плоскости к другой, проводят расчет 1-го и 2-го дробных шагов по всей расчетной области.

При выполнении 3-го дробного шага вдоль ζ (по нормали к $\xi\eta$) используется другой способ распараллеливания.

По мере перехода от k-й к k+1-й плоскости при расчете 1-го и 2-го дробных шагов проводятся вычисления коэффициентов для всех 50x50 систем уравнений третьего дробного шага и проводится вычисление k-го рекуррентного члена для каждой из 50x50 прямых прогонок вдоль ζ. Все эти вычисления по k-й плоскости ведутся в конвейерном режиме. Таким образом, после завершения 1-го и 2-го дробных шагов по всей расчетной области оказывается, что одновременно выполнена "матрица" прямых прогонок вдоль ζ. Двигаясь по плоскостям в обратном направлении, можно выполнить обратную прогонку и завершить 3-й дробный шаг во всей области.

При программировании завершение 3-го дробного шага в k-й плоскости совмещается с выполнением в этой плоскости 4-го и 5-го дробных шагов. Кроме того, при перемещении с плоскости на плоскость во время выполнения 4-го и 5-го дробных шагов вычисляются коэффициенты уравнений 6-го дробного шага и вычисляется k-й рекуррентный член всех 50x50 прямых прогонок 6-го дробного шага. Заключительный проход вдоль ζ завершает 6-й дробный шаг и весь временной шаг в целом.

Более тщательное рассмотрение процесса счета показывает, что в векторной памяти должны одновременно находиться данные не трех, а четырех плоскостей расчетной области. Кроме того, в векторной памяти должна быть отведена буферная зона обмена для данных одной плоскости.

Обмен данными с МВК "Эльбрус" должен вестись массивами - матрицами, соответствующими следующей плоскости расчетной области. Каждый узел сетки в процессе счета описывается 28 параметрами. В их число входят плотность ρ, температура T, компоненты скорости u, v, w, приращения этих величин, вычисляемые в данном временном шаге, ξ_ρ, $\xi_т$, ξ_u, ξ_v, ξ_w, длина радиуса-вектора r, теплопроводность λ вязкость μ., коэффициенты преобразования координат z_1, z_2, z_3,, y_2, y_3 и прогоночные коэффициенты α_ρ, α_u α_v, α_w, $\alpha_т$, β_ρ, β_u, β_v, β_w, $\beta_т$.Таким образом, данные задачи представляют собой четырехмерный массив 50x50x50x28.

В векторной памяти во время счета присутствуют четыре трехмерных массива, соответствующие четырем плоскостям, то есть размером 50x50x28, а также несколько рабочих массивов размером 50x50. Задача запрограммирована на базовом языке высокого уровня. Средства языка программирования позволяют обращаться к нужным подмассивам в векторной памяти. Программа имеет модульную структуру и содержит три основных модуля.

Первый модуль включает в себя расчет в текущей плоскости первого и второго дробных шагов, а также коэффициентов, и вычисление очередного рекуррентного члена третьего дробного шага по всей плоскости. В первом модуле содержатся также команды обмена, отправляющие на хранение результаты счета по предыдущей плоскости и готовящие данные следующей плоскости. В модуле сравнительно много вычислений и немного обменов. Обмен полностью совмещается со счетом.

Второй модуль завершает третий дробный шаг, включает расчет четвертого и пятого дробных шагов, а также содержит вычисление очередного рекуррентного члена шестого дробного шага по всей текущей плоскости. При работе модуля ведется обмен с памятью "Эльбрус-2". В модуле сравнительно много обменов и счет не полностью совмещен с обменом.

Третий модуль завершает шестой дробный шаг и весь шаг времени в целом. Вычислений мало, и время работы модуля определяется обменом.

Программа трех модулей насчитывает 819 векторных операторов присваивания, что соответствует приблизительно 2700 обычным арифметическим скалярным операциям с плавающей запятой на узел за один шаг по времени. Объем вычислений всей задачи составляет 10^{11} операций. Составленная программа, и известная элементная электронная база, и детально разработанная структурная схема векторного процессора позволяют сделать некоторые количественные оценки его производительности.

Во-первых, можно оценить время выполнения любого этапа вычислений: время выполнения одного векторного оператора присваивания равно произведению длительности такта конвейера на число элементов массива (для данной задачи число элементов 50x50 = 2500, и временем разгона можно пренебречь). Время обмена оценивается отношением объема переданной информации к пропускной способности канала обмена. Данные по модулям (для одновременного шага) программы решения уравнений Навье-Стокса приведены в Табл.1 при длительности такта 44 нс и темпе обмена 4 млн.слов в секунду. Таким образом, задача, включающая 300 временных шагов, будет выполнена за 27 минут.

Таблица 1

Номер модуля	Число векторных операторов	Время счета (мс)	Число обменов	Время обмена (мс)	Время работы модуля (мс)
1	667	3670	28	880	3670
2	142	780	34	1170	1170
3	10	55	14	440	440
Итого	819	4505	76	2490	5280

Во-вторых, можно получить оценку минимального объема векторной памя-ти, необходимого для реализации описанной программы. В векторной памяти должны размещаться по крайней мере данные пяти плоскостей расчетной области, то есть 50x50x50x28 = 350 $*10^3$ слов. Такой объем позволяет без дополнительных обращений к внешней памяти выполнить прогонку по двум ортогональным направлениям плоскости. При использовании достаточно быстрой векторной памяти производительность процессора в этом случае определяется темпом работы обрабатывающих устройств. При меньшем объеме векторной памяти производительность конвейера станет явно зависеть от темпа обмена (E_2) и резко упадет. Другими словами, векторная память должна обеспечить редукцию темпа обращений к внешней памяти, то есть к памяти МВК "Эльбрус-2", при этом данные всей задачи объемом 3,6 млн. слов размещаются в оперативной памяти МВК "Эльбрус-2". Сравнительная оценка производительности БЭСМ-6 и векторного процессора при решении уравнений Навье-Стокса иллюстрируется в Табл.2.

Таблица 2

Сравниваемый параметр	БЭСМ-6	Векторный процессор
Размерность сетки	10x10x15	50x50x50
Объем данных задачи (в числах)	$3x10^4$	$3,5x10^6$
Время решения задачи (в часах)	1	0,5
Относительная производительность (к БЭСМ-6)	1	200

Разработанная программа решения уравнений Навье-Стокса позволила оценить эффективность использования аппаратуры векторного процессора. В ча-

стности, оказалось, что из пяти функциональных устройств с плавающей запятой (двух сумматоров, двух умножителей и одного делителя) в среднем работают 3,34 устройства, развивая среднюю производительность приблизительно 80 Mflops. Гистограмма относительной частоты появления конфигураций с различным числам функциональных устройств приведена на Рис.9.

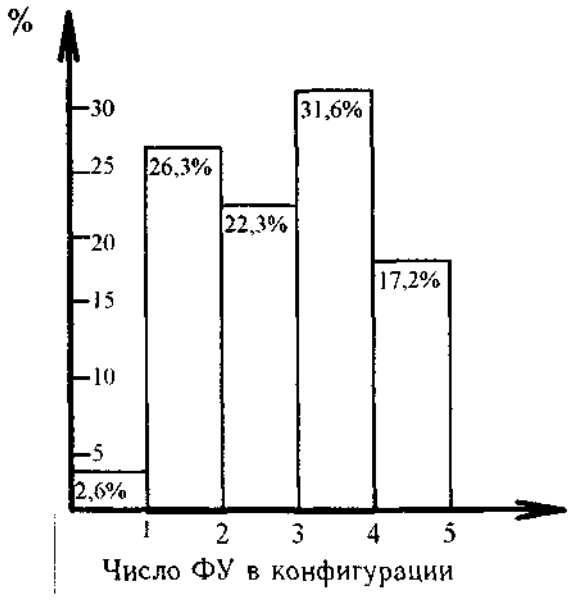


Рис.9. Гистограмма относительной частоты появления конфигураций (в %) с различным числам функциональных устройств.

Интересно отметить сравнительно высокую загрузку устройства деления: оно занимает 38,4% времени счета. Это объясняется тем, что хотя деление в данной задаче, как и во многих научно-технических задачах, составляет примерно 10% арифметических операций, одновременная работа двух устройств сложения и двух устройств умножения примерно вчетверо повышает частоту обращения к устройству деления.

Полученные результаты показывают хорошую и довольно равномерную загруженность функциональных устройств, что свидетельствует о рациональном выборе комплекса функциональных устройств.

Литература

1. ВС.Бурцев. Тенденции развития высокопроизводительных систем и многопроцессорный вычислительные комплексы. М, 1977. (Препринт ИТМиВТ).

2. В.С.Бурцев. Принципы построения многопроцессорных вычислительных комплексов "Эльбрус". М., 1977. (Препринт ИТМиВТ № 1).

3. В.С.Бурцев. Анализ результатов испытаний МВК "Эльбрус-2" и дальней шие пути его развития. М., 1988 (Препринт ОВМ АН СССР № 208).

4. В.С.Бурцев, Е.А.Кривошеев, В.Д.Асриэли и др. Векторный процессор с программируемом структурой. - В кн.: Вычислительные процессы и системы. Вып.2. М.: Наука, 1985.

5. В.Д.Асриэли, А.В.Мясников, Т.А.Попов. Опыт создания программного обеспечения векторного процессора. М., 1989 (Препринт ОВМ АН СССР).
6. В.Д.Асриэли, Т.А.Попов. Базовый язык программирования векторного процессора "МАССИВ". М., 1989 (Препринт ОВМ АН СССР).
7. В.Д.Асриэли, П.В.Борисов. Опыт программирования для векторного процессора решения уравнений Навье-Стокса в трехмерной области. - В кн: Вычислительные процессы и системы. Вып.2. М.: Наука, 1985.
8. В.М.Ковеня, Н.Н.Яненко. Метод расщепления в задачах газовой динамики. - Новосибирск: Наука, 1981.
9. В.М.Ковеня, А.С.Лебедев. Численное решение уравнений Навье-Стокса для течения над боковой поверхностью тела и в следе за ним. Новосибирск, ИТПМ СО АН СССР, 1982.
10. А.С.Оленин. Некоторые структурные и алгоритмические аспекты скалярно-векторной модели вычисления. М., 1989 (Препринт ОВМ АН СССР).
11. А.С.Оленин. Скалярно-векторные процессы в ленточных системах. М., 1987. 29с. (Препринт ОВМ АН СССР № 156).
12. Н.С.Фетисов, М.В.Твердохлебов, А.А.Саблуков, А.Н.Крамаренко. Архитектура и организация сервисного диагностического комплекса векторного процессора МВК Э-2. М., 1989 (Препринт ОВМ АН СССР).
13. M.J.Kascis. Vector Processing on the Cyber-205 Infotech state of the Art Report supercomputers, 1979, Vol.2, p.237-270
14. R.M.Russel. The Cray-1 Computer system. Comm.ACM, 1978, Vol.21, N 1, p.63-72.
15. J.R.Thompson. The Cray-1, the Cray X-MP, the Cray-1 and beyond: the Supercomputers Class VI. systems, Hardware and Software. North-Holland, 1986, p.69-81.
16. CDC Cyber-200 model 205. Technical Description Control Data Corporation. 1980, Nov.

Характеристики надежности многопроцессорных комплексов и анализ надежности МВК "Эльбрус-2"

В.С.Бурцев

Замечание

В статье недостаточно акцентировано внимание на одном чрезвычайно важном параметре характеристики надежности, которым должны отличаться вычислительные средства, работающие в системах управления технологическими процессами в масштабе реального времени, особенно в интересах военного использования.

Таким специальным параметром надежности должна быть величина, характеризующая вероятность выдачи ложной информации в систему управления объектом. Требования к величине этого параметра в различных случаях использования может существенно меняться.

Так, если выдача ложной информации приводит только к невыполнению задачи, этот параметр может быть соизмерим с общей надежностью комплекса. Если же выдача ложной информации может привести к тяжелым последствиям, таким как уничтожение объекта, уничтожение личного состава или даже принести ущерб населению, требования к достоверности выдаваемой информации управления объектом должны быть достаточно высокими. В этом случае этот специальный параметр должен на порядок или более превосходить общую надежность комплекса.

В настоящее статье подразумевается, что комплекс имеет практически полный аппаратный контроль правильности вычислительного процесса, который исключает выдачу ложной информации в цепи управления.

По этим требованиям МВК "Эльбрус-2" испытывался по самостоятельной программе, которая в статье не рассматривается. Испытания прошли успешно и показали достаточную полноту аппаратного контроля вычислительного процесса на уровне всех устройств комплекса. Статья написана исходя из того, что 100%-ая достоверность выдаваемых комплексом данных результатов гарантируется.

Анализ характеристик надежности МВК "Эльбрус-2" проводится на основании официальных данных Госкомиссии, которая закончила работу в конце 1985 года.

Август 1998 г. *В.С.Бурцев*

Введение

В соответствии с двумя основными существенно различными сферами использования вычислительных средств - в составе информационно-вычислительных центров и в системах реального времени - по-разному формулируется требование надежности к ним. В первом случае требования по надежности определяются коэффициентом снижения производительности информационно-вычислительного комплекса за счет отказов и сбоев аппаратуры, во втором - вероятностью выполнения необходимого технологического цикла в заданный промежуток времени.

Большие вычислительные комплексы, как правило, создаются для обеих сфер использования, поэтому при задании исходных данных на разработку подобных комплексов и при определении методики их испытания необходимо попытаться описать требования по надежности едиными параметрами.

При испытании модульных многопроцессорных вычислительных комплексов возникает также проблема определения надежностных характеристик различных конфигураций комплекса, что при достаточной их сложности путем натурных испытаний выполнить практически невозможно. Последнее обстоятельство накладывает дополнительное требование к измеряемым параметрам надежности, а именно их независимость от конфигурации комплекса. Решению этих проблем и посвящен первый раздел настоящей статьи.

При построении многопроцессорных вычислительных комплексов возникает вопрос рентабельности создания таких систем, с учетом того, что пропорционально количеству процессоров увеличивается объем оборудования, а следовательно, и количество отказов и сбоев аппаратуры комплекса. Может оказаться, что увеличение числа процессоров не будет увеличивать производительность комплекса или вероятность выполнения технологического цикла комплекса для системы реального времени не будет обеспечена. Очевидно, существуют критерии в части надежностных характеристик, определяющих границу, выше которой наращивание аппаратуры не имеет смысла.

Второй раздел настоящей работы посвящен определению допустимых количественных соотношений показателей надежности и методики их испытаний.

На основании требований надежности, обоснованных в первых двух разделах, приведены в качестве примера и проанализированы надежностные характеристики МВК "Эльбрус-2" (третий раздел). На основании анализа сделаны выводы об ограничениях областей применения МВК "Эльбрус-2" в части надежностных характеристик в двух основных сферах его использования.

Намечены основные направления работ по повышению надежностных характеристик комплекса, которые существенно расширят область применения МВК "Эльбрус-2" в составе информационно-вычислительных центров и в системах реального времени.

1. Параметры надежности модульного многопроцессорного комплекса и унификации их для различных сфер использования

Надежностные характеристики вычислительных средств и комплекса в целом К во многом определяют его реальную производительность, а при использовании в масштабе реального времени - вероятность выполнения технологического цикла работ в заданный интервал времени Р.

Учитывая то обстоятельство, что в многопроцессорных комплексах количество оборудования увеличивается с увеличением числа процессоров, а частота появления сбоев и отказов пропорциональна объему аппаратуры, вопрос обеспечения необходимых надежностных характеристик имеет особое значение.

1.1. Показатели надежности при работе вычислительных средств в режиме информационно-вычислительного комплекса

Производительность комплекса с учетом надежностных характеристик процессоров, но без учета прерываний вычислительного процесса, вызываемых их отказами и сбоями (Snp), при условии абсолютной надежности остальной аппаратуры комплекса может быть определена следующей формулой:

$$S_{пр} = \sum_{к=1}^{N} \Psi(к)\, C_N^{пр} P_{пр}^{к}\, q_{пр}^{N-к} \qquad (1)$$

где: $\Psi(к)$ - функция, определяющая производительность К исправных процессоров из общего количества процессоров комплекса N, без учета их характеристики надежности;

$P_{пр} P_{пр}$ - вероятность того, что процессор находится в рабочем состоянии;

$q_{пр} = 1 - P_{пр}$..

Положим, что производительность имеет линейную зависимость от числа исправных процессоров

$\Psi(к) = V_{пр}\, к$

где: $V_{пр}$ - производительность одного процессора без учета его показателей надежности.

Принимая во внимание, что:

$$\sum_{к=1}^{N} K\, C_N^{к} P^{к} q^{N-к} = pN \quad , \qquad (2)$$

получим:

$$S_{пр} = V_{пр} N \quad P_{пр} \qquad (3)$$

Реальная производительность комплекса S с учетом надежностных характеристик всех аппаратных средств комплекса меньше величины Snp и может быть определена как:

$$S = S_{пр} P_{Soc} \prod_{i=1}^{I_i} \sum_{к_i=1}^{n_i} f_i(к_i)\, C_{n_i}^{к_i}\, P_i^{к_i}\, q_i^{n_i - к_i} \qquad (4)$$

где: I - количество типов модулей в комплексе;

I_1 -количество типов модулей в комплексе без центральных процессоров;

i - индекс, определяющий тип модуля;

n_i - количество модулей i-го типа в комплексе;

$к_i$ количество работающих модулей i-го типа;

Ps_{oc} - коэффициент снижения производительности за счет прерываний вычислительного процесса, вызванных отказами и сбоями всего оборудования комплекса (включая центральные процессоры);

$f_i(k_i)$ - функция, определяющая снижение прозводительности комплекса в зависимости от числа работающих модулей (k_i) i-го типа, кроме модулей ЦП, в предположении, что все остальные аппаратные средства комплекса исправны;

P_i - вероятность того, что модуль i-го типа находится в рабочем состоянии

$$P_i = \frac{T_{oi}}{T_{oi} + T_{bi} + T_{профi}};$$

T_{oi} - среднее время наработки на отказ i-го типа модуля;

T_{bi} - среднее время восстановления i-го модуля;

$Tпрофi$ среднее время профилактики i - го типа модуля в расчете на один отказ;

$q_i = 1 - P_i$

В большинстве практических случаев для всех типов устройств, кроме центральных процессоров, можно считать, что:

$$f_i(к_i) = \frac{к_i}{n_i}.$$

Тогда, с учетом соотношения (2):

$$S = S_{пр} P_{Soc} \prod_{i=1}^{I_i} P_i = V_{пр} N P_{Soc} \prod_{i=1}^{I} P_i \qquad (5)$$

Выражение $\prod_{i=1}^{I} P_i = P_s$ определяет коэффициент снижения производительности комплекса за счет простоя аппаратных средств в результате ремонтов и профилактики его модулей. Тогда:

$$S = V_{пр} N P_{soc} P_s \qquad (6)$$

Учитывая, что полная производительность комплекса, состоящего из абсолютно надежных модулей, равна $V_{пр}$ N, можно определить коэффициент К снижения производительности комплекса за счет отказов и сбоев его аппаратуры.

$$K = \frac{S}{V_{пр} N} = P_{Soc} P_S \qquad (7)$$

Таким образом:

- отказ аппаратуры характеризуется затратами времени на ремонт аппаратуры и, как правило, прерыванием вычислительного процесса. Время ремонта устройств при их отказе учитывается в коэффициенте Ps, а время восстановления вычислительного процесса - коэффициентом Psoc;
- выход модуля на профилактику происходит без прерывания вычислительного процесса и учитывается только в коэффициенте Ps;
- сбой характеризуется прерыванием вычислительного процесса без затраты времени на ремонт и учитывается только в коэффициенте Psoc.

Формулы (1) и (4) не учитывают взаимного влияния на производительность отказов и сбоев модулей, совпадающих по времени, и предполагают выполнение следующих условий:

а) диагностика с точностью до модуля и исключение отказавшего модуля (или модуля, требующего профилактики) из рабочей конфигурации происходит практически мгновенно;

б) ремонт и профилактика осуществляется на фоне работы исправных модулей, объединенных в рабочую конфигурацию;

в) комплекс практически не имеет оборудования (или надежность этой аппаратуры близка к единице), отказ или профилактика которого требовали бы прекращения функционирования всего комплекса на время проведения ремонта или профилактики этой аппаратуры.

Коэффициент снижения производительности в результате прерываний вычислительного процесса при отказах и сбоях аппаратуры Psoc при Тз << Т ooc может определяться следующим образом:

$$P_{Soc} = 1 - \gamma_s \frac{T_з}{T_{Ooc}} = 1 - \gamma_s T_з (1 + K_{со}) n_{отк}, \qquad (8)$$

где:

$$T_{0ос} = \frac{1}{n_{отк} + n_{сб}}; \quad n_{отк} = \sum_{i=1}^{I} n_i n_{откi}; \quad n_{сб} = \sum_{i=1}^{I} n_i n_{сбi};$$

$$К_{со} = \frac{n_{сб}}{n_{отк}};$$

$n_{откi}$, $n_{сбi}$ - количество отказов и сбоев соответственно в единицу времени устройства i-го типа;

$n_{отк}$,, $n_{сб}$ - количество отказов и сбоев соответственно всей аппаратуры в единицу времени;

T_{Ooc} - среднее время наработки на отказ или сбой всей аппаратуры комплекса;

Тз - среднее время, потерянное в решении одной задачи в результате прерывания вычислительного процесса, обычно в среднем равно половине интервала фиксации промежуточных результатов решения данной задачи;

Кco - коэффициент, характеризующий: а) способность комплекса локализовать неисправность, появляющуюся в виде сбоя с учетом квалификации обслуживающего персонала; б) степень завершенности разработки и качества отладки аппаратуры;

γ_S - коэффициент, характеризующий возможности системы по уменьшению снижения производительности комплекса от прерываний вычислительного процесса (качество восстановления вычислительного процесса)

$$\gamma_1 = \sum_{m=1}^{C_{rm}} r_m S_m \qquad (9)$$

C_{rm} - количество различных видов восстановления;

r_m - весовой коэффициент m-го вида восстановления, учитывающий ущерб, нанесенный вычислительному процессу по сравнению с перезапуском всех задач, решаемых в данный момент на комплексе;

S_m - процент m-го вида восстановления.

Как правило вычислительные комплексы имеют несколько видов восстановления, такие как полное нивелирование последствий сбоя, перезапуск задачи, перезапуск нескольких задач, перезапуск всех задач, перезапуск системы. В зависимости от этого легко определяется коэффициент r. Так, при перезапуске одной, задачи из *m* задач, находящихся в вычислительной системе, r = 1/*m*.

1.2. Показатели надежности комплекса, работающего в масштабе реального времени

Аналогичным образом может быть определена вероятность P выполнения в любой заданный момент времени технологического процесса длительностью времени Δ t

$$P=P_rP_ц \qquad (10)$$

$P_г$ - вероятность того, что необходимое для выполнения технологического цикла рабочая конфигурация будет готова к выполнению необходимого вычислительного процесса в любой заданный момент времени.

$$P_г = \prod_{i=1}^{I} \sum_{к_i=к_{i\,доп}}^{n_i} C_{n_i}^{k_i} P_i^{ki} q_i^{n_i-k_i}, \qquad (11)$$

где: $К_{i\,доп}$, - минимальное количество модулей i-го типа, обеспечивающее прохождение вычислительного процесса.

Формула (11) справедлива при выполнении тех же трех условий, которые необходимы для использования формул (1 и 4).

Рц - вероятность того, что технологический цикл продолжительностью Δ t будет выполнен.

$$P_{ц} = 1 - \gamma_{ц} \frac{\Delta t}{T_{Ooc}} = 1 - \gamma_{ц} \Delta t (1 + K_{co}) n_{0\,тк} \qquad (12)$$

при $T_{Ooc} >> \Delta t$

$\gamma_{ц}$ - коэффициент, характеризующий защитные свойства комплекса от воздействий на вычислительный процесс сбоев и отказов аппаратуры

$$\gamma_{ц} = \frac{n_{н\,ц}}{n_{отк} + n_{сб}}; \qquad (13)$$

где: $n_{нц}$ - количество невыполненных технологических циклов в единицу времени по причине сбоев и отказов аппаратуры.

Цикл считается невыполненным, если время восстановления вычислительного процесса Твп было больше $Твп_{доп}$, допустимого для данного технологического комплекса. Таким образом, основной параметр надежности Р одного и того же вычислительного комплекса для различных технологических комплексов, имеющих различные $Твп_{доп}$, будет различным. Величина $n_{ц}$ подсчитывается аналогично γ_s с той лишь разницей, что коэффициент r имеет значение 0, когда время восстановления Твп >$Твп_{доп}$ и значение 1 в случае, если Твп < $Твп_{доп}$

$$n_{н\,ц} = \sum_{m=1}^{c_{r\,m}} r S_m$$

1.3. Единая методика определения надежностных характеристик

Основные характеристики надежности информационно-вычислительных комплексов (К) и комплексов, работающих в масштабе реального времени (Р), имеют одинаковую структуру и состоят из сомножителей, определяющих потери за счет времени ремонта и профилактики аппаратуры (Ps и Pr), и сомножителей, характеризующих потери в результате прерывания вычислительного процесса при отказах и сбоях аппаратуры (Psoc и Рц).

Первая пара коэффициентов, характеризующих надежность комплекса (Ps и Pr), целиком определяется совокупностью общих параметров Pi. Интересной особенностью информационно-вычислительных комплексов является то обстоятельство, что Ps не зависит от конфигурации комплекса, в то время как параметр Pr во многом определяется глубиной резервирования однотипных модулей. Как правило, за счет резервирования 1 - Pr<<1 - Ps.

Как показал опыт, нет необходимости определять значения Ps и Pr чисто экспериментальным путем, так как экспериментально-расчетный способ дает достаточно достоверные результаты.

Экспериментальным путем достаточно определить общие для обоих случаев использования значения Pi модулей.

Вторая пара надежностных коэффициентов комплексов (Psoc и Рц) определяется двумя общими параметрами Кco и S_m, где S_m представляется таблицей значений. Различие в определении параметров Ps_{oc} и Рц состоит в том, что для Psoc каждому значению таблицы Sm должно быть определено соответ-свующее значение величины r_m а для определения Рц каждому значению S_m должно соответствовать определяемое время восстановления Твп.

Если учесть, что величины Кco и S_m при $n_i > 2$ мало зависят от конфигурации комплекса, а величины r_m и $Твп_{доп}$ могут быть вполне определенным образом скорректированы в соответствии с увеличением или уменьшением от-

дельных типов модулей комплекса, то в проведении испытаний на различных конфигурациях нет необходимости.

Таким образом, определение надежностных характеристик вычислительных комплексов, имеющих совершенно различные сферы использования, можно вести экспериментально-расчетным способом практически по единой методике испытаний в соответствии со следующей схемой (Рис.1).

Исходя из вышеизложенного, можно предложить следующую систему задания требований на разработку и методику испытаний.

1. Задаются параметры Ps и Pr, при этом указывается схема резервирования для определения Pr.

2. Задаются величины Psoc и Рц, при этом для определения Psoc указывается значение Тз, а для определения Рц - величина Δt и допустимое время восстановления $Твп_{Д0П}$. С целью улучшения качества разработки и упрощения эксплуатации вычислительных средств целесообразно задавать коэффициент Кco, оговаривая при этом уровень квалификации обслуживающего персонала (средний технический, инженер высокой квалификации, разработчик). Этот коэффициент одновременно со степенью завершенности разработки характеризует возможности комплекса в части локализации самых сложных неисправностей - сбоев.

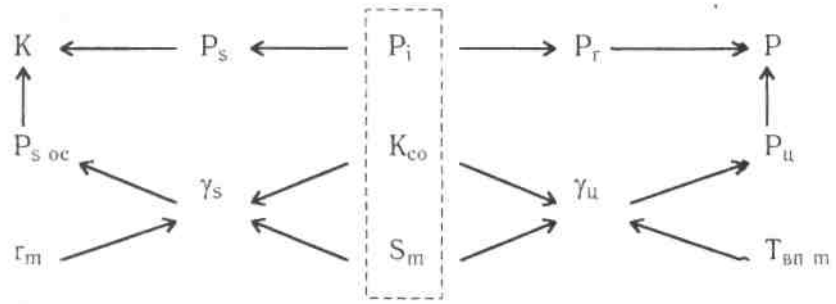


Рис.1. Общие измеряемые параметры.

В соответствии с этими требованиями методика испытаний должна иметь следующую последовательность.

1. Испытания комплекса на удовлетворение требований справедливости формул (1, 4 и 11): практически мгновенная локализация неисправности до модуля, обеспечение восстановления модуля на фоне работы рабочей конфигурации, отсутствие аппаратных средств, требующих прекращения функционирования комплекса для их восстановления и профилактики.

2. Замер параметров To_i , $Тв_i$ и $Тпроф_i$ в различных режимах эксплуатации комплекса для определения значений P_i.

3. В режиме непрерывной эксплуатации на различных пакетах задач при конфигурации комплекса, отвечающей условию $n_i \geq 2$, определить параметры $n_{отк}$, Кco, Sm, r_m и $Твп_m$

4. На основании экспериментальных данных произвести расчет показателей надежности комплекса.

Задание правильных количественных соотношений основных коэффициентов показателей надежности оказывает существенное влияние на формирование архитектуры комплекса и его системное программное обеспечение. Рассмотрим различные подходы к решению данного вопроса.

2. Количественные соотношения коэффициентов, определяющих надежность многопроцессорных комплексов

Безусловно, с увеличением количества процессоров в многопроцессорных комплексах количество оборудования будет увеличиваться, что повлечет за собой увеличение общего числа отказов всего оборудования комплекса $n_{отк}$.

Для большинства вычислительных комплексов при различных конфигурациях будет справедливо следующее соотношение:

$Q \le NA$, где A - количество оборудования, приходящееся на один процессор при минимальной комплектации комплекса, а Q - количество оборудования комплекса, содержащего N центральных процессоров.

Если через λ обозначить среднюю частоту отказов единицы оборудования, то величина $n_{отк}$ для комплекса с N центральными процессорами будет равна

$$n_{отк} = \lambda A N \qquad (14)$$

Очевидно, минимальным требованием к надежностным характеристикам многопроцессорных комплексов ($К^м$ и $Р^м$) будет требование получения показателей надежности не хуже, чем аналогичные характеристики однопроцессорного комплекса ($К°=Р°_г$ $Р°s_{ос}$ и $Р°=Р_г°$ $Р_ц°$). Выполнение этого минимального требования позволит обеспечить исключение ограничений со стороны показателей надежности по линейному росту производительности комплекса от числа центральных процессоров. Учитывая, что $Р°_s = Р^M_S$, а при резервировании устройств комплекса легко достигается равенство $Р°_г$ и $Р^м_г$, для выполнения требования $К^0 = К^м$ и $Р° = Р^м$ необходимо, чтобы $Р°_{soc} = Р^м_{soc}$ и соответственно $Р°_ц = Р^м_ц$.

Подставляя в формулы (8) и (12) значение $n_{отк}$ из формулы (14), получим:

$$P^м_{soc} = 1 - \gamma^м_s T_з (1 = K^м_{со}) AN\lambda; \qquad P^м_ц = 1 - \gamma^м_ц \Delta t(1 + K^м_{со}) AN\lambda$$

$$P^o_{soc} = 1 - \gamma^o_s T_з (1 + K^o_{со}) A\lambda; \qquad P^o_ц = 1 - \gamma^o_ц \Delta t(1 + K^o_{со}) A\lambda$$

Если разработка завершена, система полностью отлажена и специальные средства локализации сбоев отсутствуют, то $К^м_{со} \approx К°_{со}$, то есть практически все неустойчивые неисправности переходят в отказ. (В этом случае коэффициент $Кс_0$ в большей степени характеризует физику отказа).

Таким образом, минимальные требования к надежностным характеристикам многопроцессорных комплексов, обеспечивающим соотношения $К^м \ge К^0$ и $Р^м \ge Р°$, будут выполнены при реализации следующих соответствующих неравенств:

$$\gamma^m_s \le \frac{\gamma^o_s}{N}; \qquad \gamma^м_ц \le \frac{\gamma^o_ц}{N} \qquad (15)$$

Практика эксплуатации однопроцессорных систем показывает, что $\gamma_s° = \gamma_ц° = 0{,}3$-$0{,}5$.

Другой подход к определению количественных показателей надежностных характеристик при формулировке исходных данных на разработку может базироваться на "равнопрочности" систем в части показателей надежности. Так, естественно потребовать, чтобы величины (l-Ps) и (l-Psoc),(l-Pr) и (1-Рц) были одного порядка. Коэффициент Ps при современном состоянии элементной базы может колебаться в пределах Ps = 0,9 - 0,99, в то время как для коэффициента Pr с учетом резервирования вполне достижимыми являются значения 0,99 - 0,999.

Этот подход предъявляет достаточно жесткие требования к системе восстановления, в особенности для систем реального времени. Так, при $n_{отк} = 0,1$ за час (десятипроцессорный комплекс) Кco = 10 и $\Delta t = 1$ час $\gamma_ц$ не должно превосходить 10^{-2} -10^{-3} , то есть допускается, что операционная система не справится с восстановлением вычислительного процесса за время $Твп_{доп}$ один раз из ста или один раз из тысячи случаев.

Эти требования нельзя считать невыполнимыми, так как многопроцессорный комплекс в части качества восстановления вычислительного процесса имеет существенные преимущества перед однопроцессорным в том, что комплекс, имеющий стопроцентный аппаратный контроль, производит восстановление вычислительного процесса всегда на исправном оборудовании с возможностью автоматического набора статистики по сбойным ситуациям; в то время как однопроцессорный комплекс этих возможностей лишен.

Свойство многопроцессорного комплекса восстанавливаться только на исправном оборудовании можно использовать для набора статистики по сбоям с целью быстрейшей локализации неисправности, что приведет к уменьшению коэффициента Кco и к дополнительному повышению в первую очередь коэффициентов Рco и Рц.

3. Анализ надежности МВК "Эльбрус-2"

С учетом изложенного в разделах 1 и 2 был проведен анализ экспериментальных результатов, полученых при государственных испытаниях МВК "Эльбрус-2", с целью определения возможности использования его как в информационно-вычислительных центрах общего назначения, так и в системах реального времени.

Для определения надежностных характеристик комплекса расчетно-экспериментальным способом необходимо:

- проверить выполнение условия применимости формул (1), (4) и (11) (малое время локализации неисправности до модуля и возможность его ремонта на фоне работы комплекса и так далее);
- экспериментально определить параметры $n_{отк}$;, $Тв_i$, $Тпроф_i$, Кco, γ_s и $\gamma_ц$

Статистические данные по параметрам $n_{0ТК}$.$Тв_i$ и $Тпроф_i$ могут набираться во время различных режимов опытной эксплуатации и проведения испытаний при нормальных внешних условиях работы комплекса. Набор статистических данных для определения параметров Кco, γ_s и $\gamma_ц$ связан с объективной фиксацией поведения комплекса при возникновении сбойных ситуаций, что можно обеспечить только на пакете текстовых задач (ПТЗ) при работе комплекса в режиме коллективного пользования.

Время наработки на отказ устройств всецело зависит от основного параметра надежности комплектующих его схем и узлов, частоты отказов *X* сопротивлений, конденсаторов, транзисторов, интегральных схем (ИС), больших интегральных схем (БИС), печатных плат, разъемных соединений и так далее.

Результаты испытаний показали чрезвычайно низкую надежность ИС, многокристальных больших интегральных схем (МБИС), разъемов, резисторных сборок, транзисторов блоков питания, особенно на первом этапе государственных испытаний *(X* многих компонент не превышала 10^{-5}).

Узлы завода-изготовителя изделия: печатные платы, коммутационные конструкции с использованием паек, вели себя достаточно надежно. Заводами и институтами Министерства электронной промышленности в кратчайшие сроки была проделана большая работа по повышению надежности элементной базы более чем на порядок.

Распределение отказов по компонентам на заключительном этапе госиспытаний в расчете на 10^5 интегральных схем приведено в Табл.1.

Таблица 1. Распределение отказов центральной части МВК "Эльбрус-2".

Устройство	Отказы		
	МБИС	ИС	Печатные платы
Центральный процессор	2	11	-
Процессор ввода-вывода и оперативная память	-	3	1

Время восстановления неисправности состоит из времени локализации ее с точностью до модуля и времени восстановления модуля. Время локализации с точностью до модуля осуществляется в МВК "Эльбрус-2" аппаратно-программным способом, в основе которого лежит аппаратный контроль каждого модуля. От полноты аппаратного контроля всецело зависит вероятность локализации неисправного модуля в процессе работы его на системных программах и программах пользователя. В то же самое время полнота аппаратного контроля модуля обеспечивает высокую достоверность выдаваемых результатов и исключает необходимость повторных просчетов.

Испытания МВК "Эльбрус-2" в части обеспечения полноты аппаратного контроля показали, что все модули обладают практически полным аппаратным контролем. В то же время локализовать неисправность с точностью до одного модуля при отказах в цепях связи модулей (смежные неисправности) при помощи только аппаратного контроля модуля не представляется возможным. В случаях смежных неисправностей операционная система вызывает специальные тесты, которые совместно со средствами аппаратного контроля локализуют неисправный модуль. Указание на то, что отказ или сбой произошел в смежных цепях, формирует аппаратный контроль.

Результаты испытаний показали, что время локализации неисправного модуля, включая случаи смежных неисправностей, не превышает несколько секунд, а время исключения неисправного модуля из рабочей конфигурации измеряется десятками микросекунд

Таким образом, время локализации неисправности с точностью до модуля практически не влияет на надежностные характеристики комплекса, определяемые коэффициентами Ps и Pr. Время восстановления самого модуля складывается из времени локализации неисправности с точностью до типового элемента замены (ТЭЗ) и времени восстановления самого ТЭЗ.

Восстановление модуля осуществляется на фоне прохождения вычислительного процесса на рабочей конфигурации комплекса с использованием либо автономных средств локализации неисправности - инженерных пультов, осциллографов и других приборов, либо в составе ремонтной конфигурации, для чего из рабочей конфигурации в ремонтную переводятся необходимые устройства. Важно, чтобы процесс образования ремонтной конфигурации и все действия, связанные с ремонтом усройства (включение и выключение ремонтируемого устройства, приборов и замена ТЭЗ), не влияли на вычислительный процесс, проходящий в рабочей конфигурации.

Немаловажным свойством комплекса является быстрый ввод в рабочую конфигурацию исправного устройства без нарушения вычислительного процесса. В МВК "Эльбрус-2" это время не превышает несколько миллисекунд. Особенно важно это свойство комплекса для систем реального времени, так как если время ввода устройства не превышает допустимого времени готовно-

сти системы, время профилактики устройства может не учитываться при определении Pi.

Все эти свойства ремонтопригодности комплекса были тщательно проверены госкомиссией. Испытания подтвердили возможность проведения восстановительных и профилактических работ отдельных модулей на фоне непрерывного вычислительного процесса в рабочей конфигурации.

На фоне непрерывного вычислительного процесса создавались дополнительные рабочие и ремонтные конфигурации комплекса и производился ввод исправных устройств в рабочую конфигурацию.

Госкомиссия отметила, что десятипроцессорный комплекс может образовывать три рабочих и одну ремонтную конфигурацию. Время восстановления модуля, в особенности при обслуживании комплекса персоналом средней квалификации, во многом определяется качеством тестовых программ, построенных на базе аппаратного контроля. Срабатывание аппаратного контроля модуля при прохождении тестовых комбинаций локализует неисправность с точностью до группы ячеек, охваченных едиными цепями аппаратного контроля, с последующим уточнением до одной или нескольких ячеек.

Испытания показали, Что персонал средней квалификации (инженеры, проработавшие на комплексе 1-2 года) устраняет любую неисправность с помощью тестов и инструкции направленного поиска за время Тв < 1 часа. В процессе испытаний были зафиксированы следующие времена восстановления наиболее сложных устройств на искусственно вносимых неисправностях: ЦП Тв = 0,3, ОЗУ Тв = 0,5, ПВВ Тв = 0,8.

Как показывает практика, локализация нестационарных неисправностей (сбоев) создает наибольшие трудности в процессе эксплуатации вычислительных средств.

Устройство считается неисправным, если частота сбоев $n_{сбi}$ превзошла определенный критерий. Этот критерий для различных случаев использования комплекса может быть различным и вводится как параметр, по которому операционная система делает вывод об исправности того или иного модуля. Обычно модуль считается исправным, если период между сбоями превышает 15-30 минут.

Локализовать такую неисправность чрезвычайно трудно, если учесть, что она проявляется только на определенных тестовых комбинациях или программах пользователя (кодозависимая неисправность и другие). Для локализации такого рода неисправности в МВК "Эльбрус-2" предусмотрен инженерный пульт, который после прихода прерывания от аппаратного контроля модуля автоматически переключает свои входы на считывание информации с того процессора, от которого пришел сигнал аварии. Неисправный модуль по сигналу аварии автоматически останавливается. Вся информация о сбившемся модуле, включая и информацию, зафиксированную инженерным пультом, считывается рабочей конфигурацией вычислительных средств во внешнюю память. Таким образом, если модуль по критерию повышенного числа сбоев выводится операционной системой из рабочей конфигурации, то для него из архива системы может быть выведена информация о сбойных ситуациях за последний период эксплуатации устройства. Анализируя эту информацию, инженер может определить довольно точно место неисправности и заменой одной или нескольких ячеек ликвидировать ее.

В отдельных случаях При сложных неисправностях или с целью набора статистики для локализации нестационарной неисправности может быть создана ремонтная конфигурация. Создание ремонтной конфигурации позволяет вести ремонт устройств на тестах с использованием инструкции направленного поиска,

Что существенно снижает требования к квалификации обслуживающего персонала, даже в том случае, когда возникает необходимость локализации нестационарной неисправности.

Как уже отмечалось, благодаря тому, что ввод исправных устройств в рабочую конфигурацию не превышает десятков миллисекунд, создание ремонтной конфигурации не ухудшает надежностных показателей работы комплекса в системах реального времени.

Достаточно много хлопот доставляет при эксплуатации уход параметров вторичных источников питания и средств инженерного обеспечения комплекса - системы охлаждения, кондиционирования, электропитания и другие. Для контроля за этими параметрами в составе МВК "Эльбрус-2" разработан универсальный инженерный пульт (УИП), который автоматически контролирует уход параметров вторичных источников питания, температуру окружающей среды, температуру охлаждающего воздуха в каждом модуле, напор воздуха или жидкости и многие другие параметры, от которых во многом зависят надежностные показатели работы этой вычислительной системы.

Наряду с контролем этих параметров УИП производит фиксацию данных о поведении контролируемых параметров в период перед сбоем или отказом аппаратуры. Госиспытания УИП в составе комплекса прошли успешно.

На основании опыта эксплуатации МВК "Эльбрус-2" в различных системах и в период госиспытаний комиссия сделала вывод, что аппаратные средства совместно с тест-диагностическими программами и Инструкцией по эксплуатации позволяют вести круглосуточную эксплуатацию комплекса инженерно-техническим персоналом средней квалификации, обеспечивая заданные параметры надежности.

Надежностные показатели модулей, полученные за время государственных испытаний, приведены в Табл. 2.

Таблица 2. Показатели надежности устройств центральной части МВК "Эльбрус-2"

Устройство	Средняя наработка на отказ (час) To	Среднее время восстановления (час) Tв	P_i
Центральный процессор	92	0,6	0,993
Оперативная память	1263	0,29	0,9998
Процессор ввода-вывода	565	0,30	0,995

Примечание: За время испытаний (несколько больше 400 часов) не было необходимости в проведении профилактики устройств (Tпрофi = 0).

Как уже говорилось, коэффициенты Кco, γ_s и $\gamma_ц$ достаточно достоверно мо-гут быть определены при работе комплекса в режиме коллективного пользования на пакете тестовых задач. С этой целью в программе госиспытаний был предусмотрен пятисуточный прогон на достаточно мощном, с точки зрения загрузки аппаратуры, пакете тестовых задач (ПТЗ). Период прохождения одного пакета был равен полутора часам. За время одного цикла решались: шесть задач "Обтекание" (решение уравнений Навье-Стокса методом матричной прогонки и методом малого параметра), два раза задача вычисления потенциала пространственного заряда в электронной линзе методом функций Грина (УИР), задача обращения матриц размером 200*200 и ряд других задач.

Одновременно с ПТЗ решалась тестовая задача, состоящая из фрагментов обмена, вычислений и смешанных задач (ЗКГ).

В общей сложности за один цикл прохождения пакета решалось около 49 задач, а за пять суток было выполнено 80 пакетов контрольных задач.

Средняя продолжительность задач не превышает 20 минут (Тз = 10 мин). Коэффициент многопрограммности (М) изменяется в пределах 3-9. Таким образом, средний коэффициент многопрограммности Мср = 6.

Пятисуточный прогон проводился на рабочей конфигурации, состоящей из пяти центральных процессоров (ЦП), пяти модулей оперативной памяти (636 тыс. слов), двух процессоров ввода-вывода (ПВВ), двух процессоров приема и передачи данных (ППД) и восьми накопителей на магнитных барабанах.

За время пятисуточного прогона (120 часов) зафиксировано 29 сбоев ЦП, три сбоя ОЗУ, один сбой системы из-за магнитного барабана и три сбоя из-за ОСПО. За это же время отмечено 12 отказов ЦП и один отказ ППД.

Таким образом, за время прогона $n_{отк} = 0{,}11$ и Кco = 3.

В МВК "Эльбрус-2" предусмотрены несколько уровней восстановления вычислительного процесса при сбоях и отказах устройств, которые "поддержаны" аппаратными средствами.

Первый уровень предусматривает повторение операций в центральном процессоре либо повторное обращение к оперативной или внешней памяти и так далее. Этот уровень восстанавливает вычислительный процесс, практически не прерывая его.

Второй уровень восстановления предусматривает прерывание вычислительного процесса всех центральных процессоров, остановку процессора, вызвавшего прерывание, и требует разбора ситуации, в результате чего может быть перезапущена одна из задач и выведено из рабочей конфигурации устройство, если оно отказало. Время восстановления вычислительного процесса в этом случае целиком определяется создавшейся ситуацией и качеством работы операционной системы. Аппаратное время прерывания и, при необходимости, вывода из рабочей конфигурации устройства не превышает нескольких десятков микросекунд.

Третий уровень восстановления аппаратно обеспечивает исключение из рабочей конфигурации устройства, в котором произошел сбой или отказ, и считывание из внешней памяти (барабана или диска) в ОЗУ резидентной части операционной системы, необходимой для восстановления вычислительного процесса. Максимальное время работы аппаратуры в этом случае определяется в основном максимальным временем доступа к внешней памяти и временем считывания необходимой информации. В случае использования магнитного барабана это время не превышает 20 мс. Полное время восстановления всецело определяется создавшейся ситуацией, особенностями работы операционной системы, ее помехоустойчивостью и рядом других характеристик.

Необходимо отметить, что при третьем режиме восстановления, в случае многократного прерывания вычислительного процесса, аппаратные средства обеспечивают соответствующую деградацию рабочей конфигурации, причем деградация рабочей конфигурации идет до минимально возможной комплектации - один ЦП, один модуль памяти и один ПВВ.

Четвертый уровень восстановления предусматривает автоматический перезапуск системы со считыванием резидентной части операционной системы и автоматическим исключением из конфигурации устройств, в которых произошел сбой или отказ в тех случаях, когда произошло зацикливание или останов программы. Этот уровень восстановления необходим для систем реального времени с целью сокращения времени восстановления вычислительного процесса. При работе комплекса в режиме вычислительного центра восстановление в этом случае может быть произведено ручным перезапуском системы.

Для обеспечения четвертого уровня восстановления от управляющей программы комплекса или ОСПО требуется выход в течение зафиксированного промежутка времени на определенный участок программы, который проверя-

ется аппаратно. Аппаратное время восстановления вычислительного процесса в этом режиме определяется фиксированным интервалом контроля и временем считывания резидентной части ОСПО с внешней памяти в ОЗУ.

Наиболее естественным является такой режим работы системы, при котором, как правило, предусматривается первый и второй режимы восстановления, а само восстановление вычислительного процесса ведется с выходом, в случае сбоя или отказа, на третий уровень восстановления.

В тех случаях, когда время восстановления жестко ограничено и время восстановления $Тв_{доп}$ допускает считывание информации с внешней памяти, чтобы избежать возможного повторного прерывания, целесообразно работать с выходом на третий уровень восстановления, обеспечивающим автоматическую реконфигурацию системы.

Полный аппаратный контроль модулей и аппаратная "поддержка" гарантирует проведение восстановления вычислительного процесса всегда на исправной аппаратуре, что с учетом модульной многопроцессорной архитектуры построения комплекса делает его практически неуязвимым к сбоям и отказам отдельных устройств.

Это свойство комплекса было проверено на тестовой программе (неисправности в модули вносились программным путем) и на различных типах неисправностей, вносимых в модули искусственным путем. Случаев неправильной отработки алгоритмов не было. Кроме этого, система восстановления испытывалась во время пятисуточного прогона при решении ПТЗ и других контрольных задач на естественных сбоях и отказах модулей. Ввиду малой помехозащищенности алгоритмов операционной системы и ее особенностей, в распределении ресурсов памяти был реализован следующий алгоритм восстановления: второй уровень восстановления предусматривался только при сбоях и отказах аппаратуры на программе пользователя, при сбоях на процедурах общего системного обеспечения восстановление вычислительного процесса происходило при помощи третьего уровня восстановления с полным восстановлением информации, хранящейся в операционной системе и перезапуск всех решаемых задач.

В том случае, когда причиной прерывания вычислительного процесса была ОСПО (зацикливание программы), перезапуск системы осуществлялся ручным способом (четвертый уровень восстановления не использовался).

За время испытания в режиме прогона на ПТЗ (несколько больше 200 часов) зафиксировано следующее распределение различных уровней восстановления системы (Табл.3). На основании этих данных можно определить γ_s и $\gamma_ц$, а затем Ps и Рц.

Для вычислительных комплексов, загруженных задачами типа ПТЗ,

$$\gamma_s = \sum_{m=1}^{4} r_m S_m = 0{,}67$$

Надежностные характеристики пятипроцессорного и десятипроцессорного комплексов, полученные расчетно-экспериментальным путем, приведены в Табл.4. Данные Табл.4 убедительно говорят о том, что эффективность работы многопроцессорного вычислительного комплекса, измеряемая коэффициентами К и Р, практически определяется качеством системы восстановления комплекса, измеряемой коэффициентами γ_s и $\gamma_ц$. Настоящий комплекс будет работать эффективно только для тех систем реального времени, в которых время восстановления больше десяти минут.

Для использования комплекса в режиме вычислительного центра, полученные коэффициенты К=0,94 для пятипроцессорной конфигурации и К=0,89 для десятипроцессорного комплекса, также нельзя считать удовлетворительными.

Таблица 3. Распределение различных уровней восстановления вычислительного процесса.

Наименование метода восстановления	S_m	r_m для γ_s	$T_{вп}$	$T_{доп}$ для γ_u					
				10 мс r_m	10 с r_m	10 мин r_m	10 мс γ_u	10 с γ_u	10 мин γ_u
Первый уровень восстановления (повтор операции)	0,137	0	мкс	1	1	1			
Второй уровень восстановления "мягкий рестарт"	0,232	1	млс	1	1	1	0,63	0.12	1
Третий уровень восстановления "жесткий рестарт"	0,512	1	сек	0	1	1			
Ручной перезапуск	0,119	1	мин	0	0	1			

Таблица 4. Характеристики надежности МВК "Эльбрус-2."

Наименование конфигурации	Вычислительный центр			Комплекс реального времени						
	Ps	Psoc	K	Pг	Pц			P		
					Tдоп			Tдоп		
					10мс	10с	10мин	10мс	10с	10мин
Десятипроцессорный комплекс	0,99	0,9	0,89	0,9997	0,45	0,89	1	0,4592	0,89	0,9997
Пятипроцессорный комплекс	0,99	0,95	0,94	0,9996	0,02	0,946	1	0,62	0,946	0,9996

Примечание: Pг пятипроцессорного комплекса при резерве: 1ЦП, 1ОЗУ, 1ППВ, 1ППД;
Pг десятипроцессорного комплекса при резерве: 2ЦП, 2ОЗУ, 2ППВ, 2ППД;
Δt=1 час.
Во время испытаний отказа комплекса не было.

К недостаткам испытания системы в режиме вычислительного центра следует отнести следующие:

- средняя продолжительность выполняемых задач 20 минут в расчете на один процессор, принятая на испытание, слишком мала. В пересчете на десять процессоров это время составит всего две минуты. Для десятипроцесорного комплекса Тз должно быть на порядок больше;
- не учитывались потери времени на "откаты" (при потере архива, когда $r_{ПП} > 1$);
- штатный объем оперативной памяти составлял 1 млн вместо 16 млн. слов;
- виртуальная математическая память была реализована без использования дисковой памяти.

Поэтому следует ожидать, что реальный коэффициент К для десятипроцессорного комплекса будет ниже полученного. Тот факт, что только в 23% случаев ОСЛО восстанавливает вычислительный процесс без перевызова системы и полного восстановления информации, хранящейся в ОЗУ, говорит не только о неотлаженности ОСПО в этой части. Требование восстановления всей информации, записанной в ОЗУ, при перевызове системы является принципиальным для настоящего ОСПО. Это требование диктуется той принципиальной особенностью построения ОСПО, которая предусматривает размещение основной таблицы ОСПО (таблицы соответствия математических адресов физическим) по всему ОЗУ без наличия локализованной информации о ее размещении.

Такой принцип построения ОСПО существенно снижает возможности комплекса в части эффективности восстановления вычислительного процесса при сбоях и отказах, так как недоступность возможности контроля и восстановле-

ния основной таблицы ОСПО вынуждает перезапускать всю систему с полным восстановлением информации в ОЗУ.

Выводы

1. Показатели надежности вычислительного центра (К) и комплекса, работающего в масштабе реального времени (Р), определяются расчетно-экспериментальным способом на базе единых замеряемых параметров Pi, Kco и Sm и не требуют проведения испытаний на различных конфигурациях.
2. Вопрос обеспечения высокого качества восстановления вычислительного процесса при сбоях и отказах аппаратуры имеет первостепенное значение для достижения высоких показателей надежности многопроцессорного комплекса.
3. Приведенные формулы подсчета надежностных характеристик многопроцессорного комплекса позволяют существенно уточнить методики испытания подобных комплексов и более четко сформировать исходные данные на их разработку.
4. Архитектура построения многопроцессорного комплекса за счет:

- модульной структуры построения с возможностью резервирования однотипных модулей;
- практически стопроцентного аппаратного контроля работы модулей;
- автоматической локализации неисправного модуля и вывод его из рабочей конфигурации;
- четырехуровневой аппаратной поддержки системы восстановления вычислительного процесса с обеспечением восстановления только на исправном оборудовании;
- обеспечения восстановления неисправных модулей на фоне вычислительного процесса, проходящего в рабочей конфигурации;
- отсутствия аппаратуры, требующейся для ее ремонта и профилактики остановки всего комплекса,

и ряда других аппаратных решений позволяют обеспечивать высокую структурную надежность комплекса.

5. Тест-диагностические программы совместно с аппаратными средствами позволяют персоналу средней квалификации локализовать неисправности с точностью до ТЭЗ, включая сложные неисправности, проявляющиеся в виде сбоев.
6. Предъявленный на госиспытаниях МВК "Эльбрус-2" может достаточно надежно работать в системах реального времени, где $Тв_{доп} > 10$ мин и в вычислительных центрах при решении задач, имеющих малый интервал времени счета между контрольными точками.

Для расширения возможностей комплекса необходимо провести большую работу по снижению коэффициентов и хотя бы до минимальных требований, предъявляемых к многопроцессорным комплексам

$$\gamma_s^м \le \frac{\gamma_s^o}{N} \quad \gamma_ц^м \le \frac{\gamma_ц^o}{N}$$

Для достижения требований по надежности, определенных соотношениями

$$\gamma_s^м \le 10^{-1}\text{-}10^{-2}, \quad \gamma_ц^м \le 10^{-2} - 10^{-3},$$

основные концепции существующего ОСПО по распределению ресурсов, очевидно, должны быть пересмотрены.

Анализ результатов испытаний МВК "Эльбрус-2" и дальнейшие пути его развития

В.С.Бурцев

Введение

Основополагающим требованием при разработке вычислительных средств по тематике "Эльбрус" было обеспечение их предельной производительности при решении сложных научно-технических и информационных задач в режимах коллективного пользования и задач реального масштаба времени.

Исходя из этих требований, были приняты следующие основные концепции разработки:

- достижение предельной производительности одного процессора;
- создание модульной многопроцессорной системы с предельными возможностями по производительности.

В соответствии с этим в настоящей работе сначала исследуются параметры отдельных модулей комплекса, определяющие их быстродействие, а затем рассматривается производительность комплекса в целом при различных режимах работы на реальных задачах. В ходе анализа основных параметров комплекса в целом особое внимание уделено оценке его производительности с учетом надежностных характеристик. Характеристики надежности комплекса изложены в [1]. С учетом результатов испытаний и возможностей дальнейшего развития МВК "Эльбрус-2" определены основные направления работ по улучшению, в первую очередь, эксплуатационных характеристик комплекса.

1. Основные параметры центральных модулей комплекса

1.1. Быстродействие центрального процессора

Времена выполнения основных операций, замеренных в машинных тактах, приведены в Табл. 1. Период машинных тактов равен 44 нс. В процессоре организована независимая работа исполнительных устройств, таких как сложение, умножение, логические операции, деление, индексация, обработка строк, устройство

подпрограмм, устройства вызова и запись операндов. Наложение во времени циклов работы этих устройств при работе по различным программам увеличивает реальное быстродействие процессора в два-три раза. Это подтверждается проверкой работы центрального процессора в двух режимах - штатном, при котором обеспечивается одновременная работа исполнительных устройств, и технологическом, когда устройства работают последовательно.

Таблица 1

№№ п/п	Операция	Время выполнения (такты)	
		МВК "Эльбрус-2"	"Cyber-205"
1.	Сложение целых	2	
2.	Сложение с плавающей запятой	3	5
3.	Умножение целых	3	
4.	Умножение с плавающей запятой (32 разряда)	3	5
5.	Умножение с плавающей запятой (64 разряда)	4	5
6.	Загрузить в стек адрес	1	
7.	Провести индексацию	2	
8.	Загрузить операнд	2	
9.	Деление целых (32 разряда)	11	30
10.	Деление с плавающей запятой (64 разряда)	24	54
11.	Вход в подпрограмму без замены контекста	8	-
12.	Вызов процедуры	34	-
13.	Выход из процедуры	30	-
14.	Логическое сложение и умножение	2	3

Насколько удачно удалось схемотехнически решить задачу обеспечения предельного быстродействия одного процессора, можно судить по сравнению времен выполнения идентичных операций в тактах машины МВК "Эльбрус-2" с наиболее быстродействующим зарубежным универсальным процессором ЭВМ "Cyber-205" фирмы CDC (США). Из Табл.1 видно, что основные операции с плавающей запятой и ряд других операций в МВК "Эльбрус-2" реализуются за меньшее количество тактов.

Такт машины "Cyber-205" при времени задержки ($\tau_э$) элемента 0,5 - 0,7 нс составляет 20 нс, в то время как такт МВК "Эльбрус-2" при $\tau_э$ = 2-3 нс составляет 44 нс. Последние соотношения можно объяснить относительно меньшими потерями в конструкции машины МВК "Эльбрус-2" на связь между элементами.

В качестве буфера между исполнительными устройствами и оперативной памятью МВК "Эльбрус-2" имеет сверхбыстродействующую виртуальную память на быстрых регистрах и сверхоперативную буферную память объемом несколько больше 16 Кбайт. Последняя состоит из четырех совершенно поразному организованных в соответствии с их назначением устройств:

- сверхоперативная память команд объемом 512 слов - обеспечивает предварительную подкачку команд из оперативной памяти и позволяет организовывать циклы вычислений без обращения за командами в оперативную память;
- сверхоперативная память локальных данных (буфер стека) объемом 256 слов - обеспечивает автоматическое расположение данных верхушки стека в быстрой памяти и работает по принципу прямой адресации;
- сверхоперативная память массивов данных объемом 256 слов - осуществляет опережающую подкачку данных и работу по ассоциативному принципу;

- сверхоперативная память глобальных данных объемом 1024 слова организована по ассоциативному принципу с автоматическим стиранием, в случае необходимости, устаревших данных.

Обеспечена параллельная работа каждой сверхоперативной памяти с темпом работы 44 нс. Подобная организация сверхоперативной памяти исключает необходимость обращения к ней других процессоров комплекса в том случае, когда они производят запись данных в оперативную память, как этого требует принцип работы сверхоперативной памяти типа "КЭШ".

Процессор имеет возможность группового обмена данными с каждым из восьми модулей оперативной памяти в режиме "интерливинга" с четырехкратным распараллеливанием; каждый модуль памяти при этом работает независимо [3].

С целью расширения динамических ресурсов ОЗУ [2] каждый процессор имеет небольшое количество регистров, на которых выстраивается очередь обращений к памяти, и в случае возникновения конфликтной ситуации по одному из модулей памяти происходит обработка следующей по очереди заявки к свободному модулю памяти.

Все эти аппаратные решения резко снижают вероятность конфликтных ситуаций при обращении к общей оперативной памяти, что чрезвычайно важно для обеспечения линейного роста производительности от числа центральных процессоров.

1.2. Основные характеристики процессора ввода-вывода (ПВВ) и внешней памяти

Производительность процессора ввода-вывода характеризуется прежде все-го пропускной способностью его каналов. Каждый ПВВ имеет восемь независимых по времени каналов, к которым могут подключаться через устройства управления диски или барабаны. Измеренная максимальная пропускная способность каждого канала составляет 80 млн. бит/с. Таким образом, суммарная пропускная способность каждого ПВВ оценивается в 640 млн.бит/с. Однако эта пропускная способность ограничена пропускной способностью памяти или ее динамическими ресурсами [2]. Обмены такой мощности, безусловно, могут создавать конфликтные ситуации при обращении центральных процессоров к оперативной памяти. В отдельных специальных случаях может использоваться максимальная пропускная способность четырех ПВВ, превышающая 2 млрд.бит/с, при этом во время обмена возможно существенное замедление производительности комплекса.

Кроме быстрых каналов, ПВВ может осуществлять одновременную работу по 16 стандартным каналам ЕС ЭВМ, обеспечивающим подключение магнитных лент, устройств ввода-вывода и других внешних устройств ЕС ЭВМ в соответствии с ОСТ 4 ГО 304000 и ОСТ 4 ГО 304001.

По четырем специальным каналам к каждому ПВВ могут быть подключены процессоры приема передачи данных (ППД) и обеспечена одновременная их работа. Суммарная пропускная способность всех этих каналов на порядок ниже пропускной способности быстрых каналов.

Как уже указывалось, комплекс через быстрые каналы четырех ПВВ может обеспечить одновременную работу 32-х и подключение 512-ти внешних устройств памяти. Особенностью подключения внешних устройств памяти является тот факт, что к одному и тому же устройству может быть осуществлен доступ от нескольких ПВВ, а выбор канала обмена производится аппаратно.

Принятая система резервирования внешней памяти позволяет при достаточном резерве внешних устройств сделать надежность внешней памяти на порядок выше, чем надежность центральной части - ЦП, ОЗУ, ПВВ [1].

1.3. Внешняя память на барабанах и дисках

К быстрому каналу через специальные устройства управления (УБ) подключаются барабаны. В зависимости от коммутатора к одному УБ может быть подключено от 1 до 16 барабанов.

Память на барабанах имеет следующие характеристики:

- полезная емкость 4,2 млн.байт;
- среднее время доступа 5-5,5 мс;
- темп обмена 2,5 мкс/слово.

Госиспытания проводились на дисках полезной емкостью 5,68 млн.байт со средним временем позиционирования 60 мс и темпом обмена 50 мкс (20 тыс. слов/с).

Ввиду того, что интерфейс быстрого канала не стандартизирован, подключение новых дисков емкостью 100 и более мегабайт возможно только после создания специальной системы управления УВП. УВП не участвовало в госиспытаниях, так как в это время проходили его автономные испытания.

1.4. Оперативная память

Центральным устройством многопроцессорной системы является оперативная память, которая состоит из восьми независимых модулей. Каждый модуль памяти имеет четыре автономных блока управления *по* считыванию и записи, а также коммутатор связи, обеспечивающий высокий темп обмена процессор-ОЗУ.

Из-за задержки внедрения штатной электронной полупроводниковой памяти (ЭПП) испытания МВК "Эльбрус-2" проводились на памяти МВК "Эльбрус-1" П4К. Последняя имеет существенно худшие параметры по максимальному темпу обмена данными и объему хранимой информации. Основные характеристики П4К и ЭПП приведены в Табл.2.

Таблица 2. Параметры П4К и ЭПП

Параметр / Тип памяти	Разрядность слова (бит)	Количество слов в модуле (тыс.)	Время цикла запись/считывание (нс)	Миним. время обмена модуля на слово (нс)	Миним. время обмена памяти на слово (нс)	Общий объем памяти в словах (тыс.)
П4К	72	128	820/600	120-200	40	1000
ЭПП	72	2000	800/600	40	5	16000

Одним из принципиальных вопросов создания многопроцессорных систем является вопрос взаимных помех процессорных модулей при работе их на общую память. Поэтому вопрос снижения производительности комплекса вследствие взаимных помех центральных процессоров и процессоров ПВВ при испытаниях комплекса было уделено много внимания.

2. Оценка производительности комплекса

2.1. Производительность ЦП-ОЗУ

Производительность ЦП-ОЗУ была оценена на смеси "ГИБСОН 3", в соответствии с ГОСТ 163 СТ-76, на скалярных задачах и задачах с преобладанием векторных операций. Испытания проводились на минимальной конфигурации - один ЦП, один модуль памяти. В качестве задач с преобладанием скалярных операций использовались:

- задачи численного интегрирования,
- решение системы дифференциальных уравнений,
- вычисление массивов данных 100x100 с преобладанием скалярных вычислений (Старт 5М).

В качестве векторных задач были представлены:
- задача обращения матриц,
- задача умножения матриц,
- вычисления определителя.

Результаты испытаний сведены в Табл.3.

Таблица 3. Производительность ЦП-ОЗУ на различных задачах

Разрядность числа	Характер задачи			
	Скалярные операции (млн.оп/с)	Векторные операции (млн.оп/с)	"ГИБСОН 3" (млн.оп/с)	
			(алгоритмич.)	(общее количество)
32	9,3	15,6	-	-
64	8,5	13,8	5	8,5
128	3,25	6,1	-	-

Сравнение по производительности МВК "Эльбрус-2" с БЭСМ-6 и МВК "Эльбрус-1" в идентичных условиях на скалярных задачах приведено в Табл.4.

Таблица 4. Производительность различных ЭВМ

Разрядность числа	Отношение производительностей	
	Э-2 / БЭСМ-6	Э-2 / Э-1
32	14	8,4
64	13,4	8

2.2. Производительность системы ЦП-ОЗУ на различных конфигурациях

Прежде всего была проверена зависимость изменения производительности системы ЦП-ОЗУ при подключении второго процессора при одном и двух модулях ОЗУ. Результаты показали, что изменение производительности при различных конфигурациях не превышает трех процентов. Затем общая производительность пятипроцессорного комплекса с пятью модулями ОЗУ была оце-нена на достаточно представительных счетных задачах УИР и ДНК.

Задача УИР вычисляет пространственный потенциал заряда в аксиально-симметричной системе электродов электронной линзы методом функций Грина. Каждая задача требует 5000 слов памяти и семь минут времени счета одного процессора. На БЭСМ-6 эта задача решается за 1,5 часа. Отличительной особенностью структуры задачи УИР является то, что ее данные хорошо локализуются в сверхоперативной памяти процессора. На пятипроцессорном комплексе запускались одновременно пять задач.

Задача. ДНК требует 450 тыс. слов, расчет ведется по десяти математическим областям. Задача предварительно была адаптирована к пятипроцессорному комплексу посредством вставок на языке Эль-76 с целью оптимальной загрузки процессоров в течение всего времени счета. Выбранный фрагмент ДНК является наиболее вычислительно емким участком задачи. На ЕС-1066 для расчета данного фрагмента требуется более 11 часов. Задача имеет достаточно большой темп обмена процессор-ОЗУ - средний темп обмена на один процессор оценивается

в 35-40 Мбит/с. Замеры времени счета проводились на комплексе с разным числом центральных процессоров.

Результаты измерений для задач УИР и ДНК приведены в Табл.5. В той же таблице приведены коэффициенты увеличения производительности комплекса, состоящего из N процессоров, по отношению к производительности комплекса с одним процессором ($К_{мцп}$) в однопрограммном режиме

$$Кмцп = T_1 / T_N , \quad (1)$$

где T_N - время решения задачи с N процессорами;

T_1- время решения задачи с одним процессором.

Таблица 5. Зависимость производительности комплекса от числа ЦП

Количество ЦП (N)	Время T_N (мкс), требуемое на решение задачи:		Коэффициент увеличения производительности комплекса при решении задачи:	
	УИР	ДНК	УИР	ДНК
1	2107	3117	1	1
2	1052	1575	2	1,98
3	702	1056	3	2,95
4	528	802	3,99	3,89
5	422	644	4,99	4,84

Полученные результаты подтверждают практически линейный рост производительности от числа процессоров. На основании этих данных путем аппроксимации полученных результатов комиссия сделала вывод о том, что $К_{МЦП}$ для десятипроцессорного комплекса в среднем будет не хуже 9,87. Последующие испытания десятипроцессорного комплекса при проверке его по техническим условиям подтвердили этот результат.

В МВК "Эльбрус-2" предусмотрена возможность изменения "расслоения" памяти путем объединения двух модулей. Таким образом, в штатном режиме внутри каждого модуля памяти реализовано "расслоение" памяти на четыре, а при объединении двух модулей обеспечивается "расслоение" памяти на восемь.

На задаче ДНК было испытано влияние изменения "расслоения" памяти на производительность комплекса. Наибольшее увеличение производительности при "расслоении" на восемь получилось на пятипроцессорной конфигурации и не превышало трех процентов.

2.3. Влияние работы ПВВ на производительность системы ЦП-ОЗУ

Для оценки влияния на общую производительность комплекса обменов с внешней памятью за счет конфликтных ситуаций, возникающих при обращении к общей памяти, было проведено несколько испытаний комплекса по различным методикам, которые подтвердили, что в худшем случае можно ожидать уменьшение производительности не более чем на три процента.

Наиболее эффективными в этом направлении и близкими к реальности были испытания на задаче ДНК. Трудность организации достоверных испытаний состоит в том, чтобы найти задачу, на которой бы шел одновременный обмен с десятью и более внешними устройствами. Используя тот факт, что интерфейс и схемотехника подключения центральных процессоров и процессоров ввода-вывода в МВК "Эльбрус-2" идентичны, эквивалентный темп обмена ПВВ-ОЗУ можно имитировать обменами ЦП-ОЗУ.

Таким образом, учитывая, что темп обмена ЦП-ОЗУ на задаче ДНК составляет 35-40 Мб/с, можно за счет включения центрального процессора, работающего на этой задаче, имитировать работу ПВВ с некоторым количеством дисков или барабанов одновременно. В Табл. 6 приведены результаты влияния на время счета той же задачи ДНК помех, создаваемых другими процессорами.

Таблица 6. Снижение производительности комплекса за счет имитации работы внешней памяти

Количество процессоров ЦП, имитирующих работу ПВВ	Время решения задачи, с	Фоновой поток обмена с ОЗУ, создающим помехи	Эквивалентное количество дисков	Уменьшение производительности, %
0	607	0	0	0
1	616	35-40	4	1,4
2	622	70-80	8	2,4
3	625	105-120	12	2,94
4	626	140-160	16	3,1

Учитывая, что максимальный темп, с которым ведет обмен диск, не превышает 0,6 Мб/с, можно сказать, что даже одновременная работа 16 дисков не вызовет значительного снижения производительности комплекса.

Полученная практически линейная зависимость роста производительности комплекса и малая зависимость ее от интенсивности обмена с внешними устройствами продемонстрирована впервые в мировой практике. Эти результаты получены благодаря следующим архитектурным и схемотехническим решениям, реализованным в МВК "Элъбрус-2":

1. Решен вопрос индивидуальной для каждого ЦП сверхоперативной памяти, исключающей необходимость стирания в ней информации при записи других процессоров в ОЗУ, как это необходимо делать в других вычислительных комплексах, использующих сверхоперативную память типа "КЭШ".
2. Резко увеличена пропускная способность каналов ЦП-ОЗУ-ПВВ за счет:

а) обеспечения независимой коммутации каждого модуля ЦП с каждым модулем памяти;

б) расслоения памяти на 32 автономно работающие секции;

в) автономной организации группового обмена по четыре слова;

г) введение специальных буферных регистров в ЦП, исключающих его простой в случае возникновения конфликтной ситуации на одном из модулей ОЗУ.

3. Аппаратно реализованы с минимальным обращением к ОЗУ наиболее часто встречающиеся конструкции языков высокого уровня.
4. Прямоадресуемые регистры ЦП заменены виртуальной памятью на быстрых регистрах, что позволило существенно расширить ее и снизить число обращений к ОЗУ.
5. Реализован ряд других архитектурных решений комплекса, обеспечивающих рациональное использование динамического ресурса памяти [2,3].

3. Общее системное программное обеспечение

3.1. Общие характеристики

В состав общего системного программного обеспечения (ОСПО), представленного на испытания, входили следующие системы:

1. Операционная система, состоящая из управляющей программы, программы управления процессами и памятью и программы управления простыми файлами.
2. Системы программирования на языке высокого уровня - Эль-76 (автокод МВК "Эльбрус"), ФОРТРАН и АЛГОЛ-60 с универсальным комплексатором.
3. Система стандартных и сервисных программ, состоящая из набора программ математических функций, редактирования текстов, программ динамической диагностики и сортировки на дисках.
4. Система учета пользователей.
5. Пакеты тестовых программ, проверяющих правильность функционирования ОСПО и автоматизации выдачи документов.

Кроме перечисленных программ, в состав ОСПО входило программное обеспечение телеобработки и машинной графики, которое испытывалось по самостоятельной методике (результаты испытаний в данной работе не обсуждаются).

Все перечисленные компоненты ОСПО были испытаны при непрерывной пятисуточной работе комплекса в режиме вычислительного центра на пакете тестовых задач, а Также на ряде специальных проверочных задач.

В результате испытаний комиссия пришла к выводу, что ОСПО МВК "Эльбрус-2" является универсальной общецелевой системой и обеспечивает:

а) многопроцессорный, многопрограммный и многозадачный режимы вычислений;

б) динамическое (а при необходимости статическое) распределение ресурсов системы;

в) организацию файлов и универсальных архивов;

г) виртуальную память объемом 32 млрд.байт для каждой задачи;

д) контекстную защиту данных;

е) организацию параллельных вычислений;

ж) использование МВК "Эльбрус-2" в режиме разделения времени и в пакетном режиме местной обработки информации, а также совмещение этих режимов;

з) редактирование текстов построчное, позиционно-символьное, символьное по диапазону с возможностью сдвигов фрагментов и вставки новых;

и) реконфигурацию системы без нарушения вычислительного процесса, включая ввод и вывод модулей на ремонт и профилактику;

к) локализацию неисправности до модуля.

Объем внешней памяти, необходимой для хранения операционной системы, на начальном этапе испытаний составил 131 тыс.слов, из которых 38,6 тыс.слов составляли резидентную часть. Времена выполнения операционной системой основных процедур приведены в Табл.7.

Таблица 7. Времена выполнения процедур ОСПО

Наименование процедуры	Время выполнения	Примечание
Выделение оперативной памяти	470 мкс	1000 слов
Организация процесса	658 мкс	Процедура “ветвь”
Открытие файла на МД	454 мс (127 мс)	В скобках - для барабана
Генерация для непосредственного обмена	24 мс (8,9 мс)	В скобках - для барабана
Вызов процедуры без параметров	5 мкс	
Время реакции на прерывание	47 мкс	От момента возникновения ситуации до выхода на подпрограмму разбора

3.2. Многопрограммный режим

На различных смесях программ был определен один из основных коэффициентов, определяющий качество работы ОС в многопрограммном режиме, - коэффициент многопрограммности $К_{мп}$:

$$К_{мп}=To / T_{мп}, \quad (2)$$

где $T_{мп}$ - время решения задач, при котором комплекс решает одновременно М задач, причем М больше N, где N - число ЦП комплекса;

T_O - время решения задач при последовательном их выполнении комплексом - однопрограммный режим (M = N).

Выигрыш во времени получается за счет совмещения работы при многопрограммном режиме работы центральных процессоров и процессора ввода-вывода.

Одновременно с коэффициентом $К_{мп}$ измерялся коэффициент многопроцессорности в многопрограммном режиме $К_{мцп}$

$$Кмцп = T_{оцп} / T'_{мцп}, \quad (3)$$

где $T_{оцп}$ - время решения М задач на комплексе с одним ЦП в многопрограммном режиме (M > 1);

$T_{МЦП}$ - время решения М задач на комплексе, состоящем из нескольких ЦП, в многопрограммном режиме (M > N).

Получены следующие коэффициенты $К_{МЦП}$ для различных пакетов программы при М = 10 (десятикратное решение одного и того же пакета) при комплектации один ЦП, два модуля ОЗУ и два ЦП, два модуля ОЗУ (Табл.8).

Таблица 8. Значения коэффициентов многопрограммности и многопроцессорности

Количество ЦП	М	Смесь программы		Пакет "факт"		ПУИР	
		$К_{мп}$	$К_{мпц}$	$К_{мп}$	$К_{мцп}$	$М_{мп}$	$К_{мцп}$
1	10	1,072	1	1,042	1	1,018	1
2	10	1,01	1,91	1,035	2	1,01	1,99

Обычно коэффициент многопрограммности колеблется в пределах 1,5-3 и во многом зависит от характера задачи. Задачи, имеющие достаточно большой процент обмена с внешней памятью, в процессе их выполнения могут давать лучший $К_{мп}$. Полученный на испытаниях коэффициент $К_{мп}$ практически равный единице, говорит о том, что вся задача находится в оперативной памяти и не нуждается в обменах с внешней памятью. На испытаниях ставились пакеты с достаточно интенсивными обменами с внешней памятью, в результате чего коэффициент $К_{мп}$ увеличивался, но с увеличением $К_{мп}$ резко снижался коэффициент $К_{мцп}$.

Полученные результаты могут быть объяснены только особенностями работы операционной системы (ОС). Принятая в ОС концепция обмена малыми сегментами (объем сегмента определяется контекстом программы) не может обеспечить, требуемый даже для одного процессора пропускной способности канала обмена с внешней памятью. Необходимый средний темп обмена ОЗУ с внешней памятью ($T_{ОЗУ-ВП}$), обеспечивающий достаточную загрузку процессора, может быть определен следующим соотношением (раздел 4):

$$T_{ОЗУ-ВП} = К_1*К_2 / V_{пр} * \lambda, \quad (4)$$

где: $V_{пр}$ - производительность процессора;

λ - среднее количество обменов на одну операцию;

K_1- средний коэффициент локализации данных в объеме сверхоперативной буферной памяти процессора (снижение темпа обмена за счет сверхоперативной памяти);

K_2 - средний коэффициент локализации данных в объеме ОЗУ (снижение темпа обмена за счет ОЗУ).

По результатам статистического анализа, приводимой фирмой IBM (США), коэффициенты K_1 и K_2 в среднем не превышают 10. В МВК "Эльбрус-2" $\lambda = 2$ и $V_{пр} = 12{,}5$ млн.оп/с. В этом случае $T_{ОЗУ\text{-}ВП}$. вычисленная по формуле (4), должна быть меньше 4 мкс.

Темп обмена, который обеспечивает один барабан при обмене квантами в 100 слов ($n_{гр} = 100$), рассчитанный по формуле (5), равен 52 мкс

$$T_{ОЗУ\text{-}ВП} = (T_{ДОСТ}\, n_{гр}) + Tтемп, \qquad (5)$$

где: $T_{дост}$ - время, необходимое для считывания или записи первого слова ($T_{дост}$ для барабана равно 5 мс);

$T_{темп}$ - время считывания или записи последующих слов ($T_{темп}$ для барабана равно 2 мкс);

$n_{гр}$ - количество слов в группе.

Для внешней памяти на дисках $T_{ОЗУ\text{-}ВП}$ будет превышать 500 мкс.

Таким образом, пропускная способность канала обмена с внешней памятью даже для одного процессора в 10 раз меньше необходимой при использовании барабана и в 100 раз меньше при использовании диска.

Учитывая линейный рост производительности в зависимости от числа процессоров, для десяти процессоров необходимо увеличить пропускную способность каналов обмена с внешней памятью в 100 раз при использовании барабана и в 1000 раз при использовании дисков. Выход из данной ситуации следующий:

- ОС должна формировать обмены с внешней памятью объемом в десятки тысяч слов, то есть на два порядка больше, чем это было реализовано на этапе испытаний;
- при обработке очереди заявок для дисковой памяти необходимо пропускать заявки с учетом расположения головок;
- при записи информации на внешние носители ОС должна так распределять информацию по внешним устройствам, чтобы обеспечивалась максимальная одновременность работы.

3.3. Режим разделения времени

Операционная система (ОС) была испытана в режиме разделения времени при подключении максимально возможного в условиях испытаний количества терминального оборудования - 16 терминалов. Испытания проходили в течение четырех часов. За каждым терминалом проводилось редактирование текстов программ и исполнение тестовых программ. В процессе работы было зафиксировано шесть полных перезапусков системы, два из которых произошли из-за ошибок в ОС и четыре в результате сбоев в ЦП.

Тот факт, что в режиме разделения времени при четырех сбоях аппаратуры ОС ни разу не справилась с этой ситуацией и вышла на полный перезапуск системы и всех решаемых задач, безусловно, характеризует малую помехозащищенность ОС в этом режиме. Примерно тот же результат дали испытания ОС, проводимые на поддержание непрерывности вычислительного процесса при работе комплекса в пакетном режиме в течение 214 часов. Только в 23% случаев ОС справилась со сбойной ситуацией и без прерывания всего вычисли-

тельного процесса перезапустила ту задачу, на которой произошел сбой. Анализ причины малой помехозащищенности ОС подробно рассматривается в [1].

3.4. Реконфигурация комплекса без нарушения вычислительного процесса

В процессе испытаний проверялась возможность комплекса автоматически локализовать неисправный модуль по набору необходимой статистики и прошения вычислительного процесса. Испытания проводились как на искусственно вносимых неисправностях, так и в режиме пятисуточного прогона при работе комплекса в пакетном режиме. Операционная система успешно локализовала все неисправности с точностью до модуля и осуществляла вывод и ввод модулей из рабочей конфигурации без нарушения вычислительного процесса. Одновременно были проверены возможности комплекса образовывать несколько параллельно функционирующих рабочих и ремонтную конфигурации и сбор комплекса в единую конфигурацию без нарушения вычислительного процесса. Испытания прошли успешно.

3.5. Система программирования

Отличительной особенностью системы программирования является то, что она обеспечивает разработку и отладку программ только в терминах языков высокого уровня. Это свойство, безусловно, обеспечивает высокую комплексную производительность системы, имея в виду, что в понятие "комплексная производительность" входит время решения задачи от момента начала программирования задачи до получения результатов ее решения.

В систему программирования МВК "Эльбрус-2" на момент испытания вводили "ФОРТРАН", "АЛГОЛ-60", "ЭЛЬ-76", средства редактирования текстов, многоязыковый комплексатор, словарь употребления идентификаторов в программе, динамическая диагностика и программы отладки. Комиссия отметила, что Эль-76 ориентирован на архитектуру МВК "Эльбрус-2" и обеспечивает высокий уровень и надежность программирования, а также эффективное использование возможностей аппаратуры и ОС. Эль-76 пригоден для создания больших программных комплексов различной проблемной ориентации, включая создание математического обеспечения автоматизированных систем управления объектами в реальном масштабе времени, системного программно -поисковых задач. Эль-76 используется в МВК "Эльбрус-2" как язык управления заданиями.

В результате испытания системы программирования была показана возможность комплексирования программных модулей, оттранслированных с различных представленных языков программирования, и получены следующие характеристики:

- объем текста программы транслятора с Эль-76 составляет 43 тыс. строке
- с ФОРТРАНа - 17 тыс. строк;
- с АЛГОЛа - 22 тыс. строк.

Скорость трансляции программ при работе с внешней памятью на барабане равна:

для Эль-76 - 60 тыс. строк в минуту; для
ФОРТРАНа - 55 тыс. строк в минуту; для
АЛГОЛа-60 - 30 тыс. строк в минуту.

'Для тех же программ были замерены объемы оттранслированных кодов $Q_Э$, $Q_Ф$ и Q_A и времена выполнения программы $T_Э$, $T_Ф$ и T_A соответственно, написанных на Эль-76, ФОРТРАНе и АЛГОЛ-60. На основании этих данных подсчитаны коэффициенты удлинения программ для ФОРТРАНа $К_{УФ} = О_Ф / Q_Э = 1,2$ и для АЛГОЛ-60 Кцд = Q_A / Оэ = 1.53 и, соответственно, коэффициенты замедления прохождения программ: $К_{ЭФ}$= 1,3 и $К_{ЗА}$ = 138.

Результаты испытания системы программирования говорят о том, что основное усилие разработчиков было направлено на оптимизацию программ, написанных на Эль-76. Однако основная задача создания комплекса для широкого применения в народном хозяйстве состояла в обеспечении эффективного использования универсальных языков высокого уровня. Тот факт, что Эль-76 является языком достаточно высокого уровня, обладающим мощными изобразительными средствами, а над оптимизацией универсальных языков фактически не работали, дает основание предполагать, что коэффициенты Кц и $К_З$ для многих универсальных языков, благодаря эффективной аппаратной поддержке типовых языковых конструкций, реализованной в МВК, могут быть получены близкими к единице.

Существует такое мнение, что Эль-76 настолько хорош и перспективен, что он имеет самостоятельное значение и найдет широкое использование в отечественной и зарубежной практике.

Тенденции развития вычислительной техники не подтверждают тот факт, что в направлении принципов создания Эль-76 предполагает последовательное описание алгоритмов и не содержит средств автоматического распараллеливания вычислительного процесса - языковые конструкции синхронизации вычислительных процессов выходят на уровень аппаратных решений.

В то же время развитие вычислительной техники идет в направлении существенного распараллеливания вычислительных процессов, предусматривающего переход к новым не фон-Неймановским принципам организации вычислительного процесса. К тому же распространение этого языка на другие современные машины существенно ограничивается его жесткой ориентацией на аппаратные средства МВК "Эльбрус-2" и, в первую очередь, на использование системы "тегов". Поэтому более правильно квалифицировать этот язык как автокод МВК "Эльбрус".

Эль-76 целесообразно, в первую очередь, использовать как инструментальное средство для эффективной разработки системных программ и программ специализированных систем управления, жестко привязанных к тем аппаратным средствам, которыми они управляют.

4. Вопросы эффективного использования МВК "Эльбрус-2" при реализации задач

Основная задача достижения предельной производительности при проектировании МВК "Эльбрус-2" рассматривалась в первую очередь в плане достижения минимального времени между началом постановки задачи на комплекс и моментом получения результата ее решения. Наиболее полным это решение выглядело бы следующим образом.

1. Пользователь программирует задачу на наиболее подходящем для описания задачи универсальном языке высокого уровня.

2. Оптимизирующий транслятор составляет такую программу в кодах машины, для которой граф вычислительного процесса на всем протяжении решения задачи в достаточной степени совпадал бы со структурой вычислительных средств.

Рассмотрим, насколько близко постановка и реализация задач на МВК "Эльбрус-2" совпадают с вышеописанным идеальным порядком.

Для того, чтобы пользователь программировал только на универсальных языках высокого уровня, необходимо создать трансляторы, обеспечивающие эффективную реализацию программ, написанных на этих языках. Имея в виду особенности архитектуры МВК "Эльбрус-2", и в первую очередь то, что комплекс состоит из N процессоров, а каждый процессор содержит несколько параллельно работающих исполнительных устройств, уточним дополнительные требования, которым должен стремиться удовлетворить пользователь при создании программы для этого комплекса.

Требование совпадения графа вычислительного процесса с архитектурой вычислительных средств в первую очередь предполагает максимальное использование во времени как всех модулей комплекса, включая центральные процессоры, так и исполнительных устройств каждого из них.

Вопрос загрузки N параллельно работающих процессоров комплекса в режиме пакетной работы или в режиме разделения времени решается операционной системой. Загрузка процессоров комплекса в режиме монозадачи в МВК "Эльбрус" требует от пользователя создания такого алгоритма решения задачи, при котором на протяжении всего счета существовало бы N или более независимых процессов обработки информации.

Необходимо отметить, что задачу эффективной загрузки исполнительных устройств процессора для традиционных ЭВМ предполагалось решить путем создания оптимизирующих трансляторов, осуществляющих бесконфликтное распределение ресурсов процессора во время трансляции (статически). Однако эти попытки не увенчались успехом ввиду невозможности учета всех временных соотношений работы устройств процессора, оперативной и внешней памяти в статике.

В МВК "Эльбрус-2" задача бесконфликтного распределения ресурсов устройств центрального процессора решается в динамике вычислений аппаратным способом. Таким образом, аппаратное распределение виртуальных быстрых регистров по исполнительным устройствам, реализованное в МВК "Эльбрус-2", фактически решает задачу динамического отображения графа вычисления на аппаратные средства. При этом коэффициент параллельности использования исполнительных устройств процессора колеблется в пределах 2-3 в зависимости от количества параллельных ветвей счета в вычислительном процессе. В то же время задача эффективного использования аппаратных средств будет решена полностью только в том случае, если в виртуальные быстрые регистры всегда будет обеспечена подкачка новых данных и вывод результатов. Необходимым условием решения этой задачи является выполнение следующего соотношения для каждого уровня иерархии памяти на любом произвольно выбранном интервале времени Δt:

$Q_n K_n \geq E_n \Delta t$, или, учитывая, что $Q_n = E_{n+1} \Delta t$,

$$K_n \geq E_n / E_{n+1}, \qquad (6)$$

где K_n - коэффициент локализации данных в объеме Q_n на интервале Δt (K_n определяет среднее количество переиспользованных ячеек памяти в объеме Qn за время Δt);

Q_n - объем памяти n-го уровня иерархии в структуре МВК;

E_n - пропускная способность канала памяти n-го уровня в направлении ЦП;

E_{n+1} - пропускная способность канала памяти n-го уровня в направлении внешней памяти

Практика показала, что для большинства задач на всех уровнях памяти без вмешательства пользователя выполняется условие $K_n \leq 10$. Достижение боль-

шей локализации данных, как правило, требует вмешательства пользователя. Таким образом, можно считать, что если архитектура ЭВМ обеспечивает для каждого уровня иерархии памяти условие $E_n / E_{n+1} < 10$, то адаптация большинства задач к такой ЭВМ происходит автоматически. В том случае, когда это требование по каким-либо причинам не реализуется (хотя бы для одного из уровней иерархии памяти), резко сужается класс задач, адаптация которых к комплексу происходит автоматически.

Проведем анализ архитектуры МВК "Эльбрус-2" в части выполнения этого требования на всех уровнях памяти. На Рис.1 приведена архитектура МВК "Эльбрус-2" с указанием реальных параметров E_n, Q_n K_n по всем уровням иерархий памяти в расчете на один процессор (1/8 общего ресурса памяти, оперативной и внешней памяти).

В то же время, чем выше реальная локализация данных в задаче, тем быстрее протекает вычислительный процесс, поэтому пользователю небесполезно знать, что буферная сверхоперативная память МВК "Эльбрус-2" состоит из сверхоперативной памяти локальных данных (буфер стека) объемом 256 слов, ассоциативной памяти глобальных данных объемом 1024 слова, памяти опережающей подкачки массивов объемом 256 слов, буфера команд объемом 512 слов.

Пропускная способность канала ОЗУ - внешняя память в соответствии с формулой (5) зависит от типа устройства (барабан или диск), количества параллельно работающих устройств и среднего количества слов, которым за одно обращение обменивается ОЗУ с внешним устройством.

В Табл.9 приведены значения коэффициента K_2 для различных квантов обмена информации ОЗУ с внешней памятью. Из таблицы видно, что условие $K_n \leq 10$ выполняется при использовании барабанов, начиная с $n_{гр} = 8$ тыс.слов, а при использовании дисков - с $n_{гр} > 16$ тыс.слов.

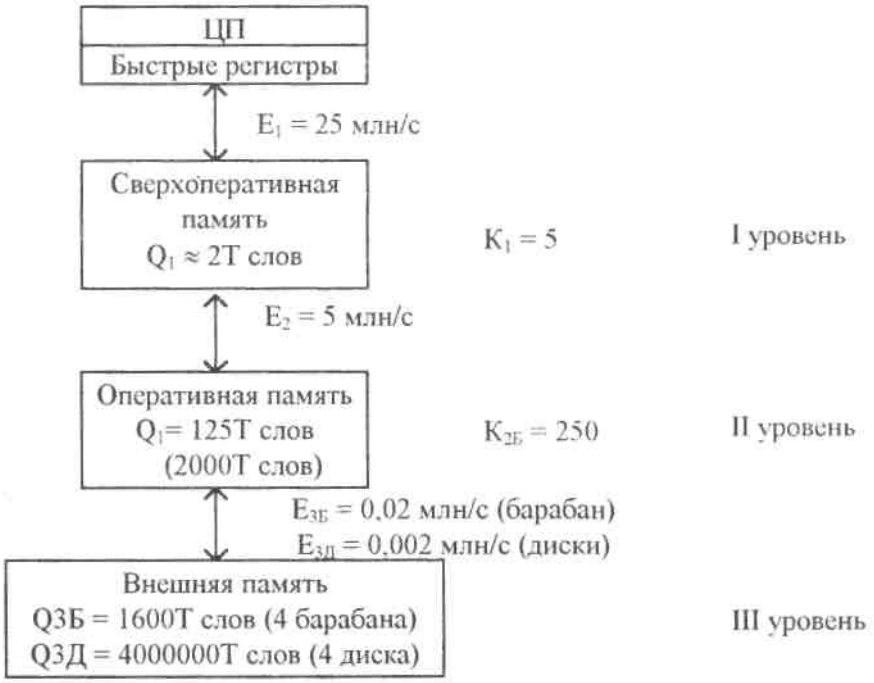


Рис.1. Пропускная способность каналов связи на различных уровнях иерархии памяти

Таблица 9. Зависимость коэффициента локализации K_2 от параметра $n_{гр}$

Параметр	Тип устройства	$n_{гр}$			
		100 слов	1 тыс.слов	8 тыс.слов	16 тыс.слов
K_2	Барабаны	250	40	10	8
	Диски	2500	200	40	30

Как уже было показано, пропускная способность аппаратных средств позволяет обеспечить одновременную работу достаточного количества внешних устройств памяти, работающих на предельных темпах обмена практически без снижения производительности ЦП. Как показали испытания, принцип сегментирования памяти на уровне запросов процедур, приводящий к среднему размеру сегмента в 100 слов и соответствующему обмену данными между ОЗУ и внешней памятью, достаточно хорошо использует статический ресурс оперативной памяти (83% коэффициент заполнения), однако существенно снижает пропускную способность каналов обмена ОЗУ с внешней памятью.

Учитывая, что основная цель сверхоперативной и оперативной памяти не хранение данных, а обеспечение непрерывности работы исполнительных устройств центрального процессора, принцип разбиения данных на малые сегменты и реализации такого обмена между оперативной и внешней памятью, заложенной в ОС, не выдерживает критики. Положение усугубляется тем, что при штатной памяти объемом в 16 млн.слов (ЭПП), память на барабанах объемом 0,4 млн. слов на одном барабане не имеет смысла, а для памяти на дисках время доступа Тдост на порядок больше, чем на барабане.

Операционная система комплекса производительностью в десятки миллионов операций в секунду должна обеспечивать обмен оперативной и внешней памяти сегментами объемом в 8, 16, 64 тыс.слов, при этом обеспечивать такое распределение информации по внешним устройствам, при котором одновременность работы этих устройств была бы максимальной. Очевидно, что в концептуальном плане это приведет к обменам целыми задачами или достаточно локализованными модулями задач.

Представленная на испытания операционная система обеспечивала автоматическую "эффективную постановку задач только в том случае, если коэффициент локализации данных в объеме Q = 125 тыс. слов был более 250 при работе с барабанами и более 2500 при работе с дисками.

Практически таких задач встречается очень мало, поэтому на испытаниях адаптация многих задач выходила на пользователя, не только в целях распараллеливания вычислительного процесса, а также и в части локализации данных задач путем объединения их в укрупненные сегменты. Необходимо отметить, что в ОСПО есть средства, позволяющие объединять различные массивы данных в сегменты больших размеров. Как известно, размещение данных в ОЗУ крупными непрерывными сегментами вызывает большие потери памяти на так называемую "внешнюю фрагментацию". Этот вопрос обеспечения смежности ячеек оперативной памяти при загрузке в нее крупных сегментов решен в МВК "Эльбрус-2" с помощью обычного разбиения математической памяти сегментов на страницы по 512 слов. Такое решение исключает потери на внешнюю сегментацию, а с увеличением объемов сегментов уменьшаются потери на внутреннюю сегментацию (незаполненные последние страницы сегмента).

Предусмотренная в ПВВ операция записи-считывания позволяет в этом случае необходимое количество страниц, разбросанных по оперативной памяти и принадлежащих одному сегменту, записать последовательно на единую зону внешней памяти за одно обращение к нему. Аналогичная операция считыва-

ния единого массива может записать в необходимые места ОЗУ считываемые страницы данных.

В настоящее время задача автоматической адаптации задач на МВК "Эльбрус-2" не решена в полном объеме. Кроме требования распараллеливания вычислительного процесса на N независимых ветвей, пользователь для обеспечения непрерывной загрузки процессоров должен выполнить достаточно сложное условие при создании алгоритмов и задачи в целом -обеспечить высокую степень локализации данных ($К_2 > 2500$) на объеме памяти $Q_2 = 2$ млн.слов в расчете на один процессор.

Таким образом, переход на обмен большими, странично организованными сегментами позволит упростить работу операционной системы, повысит ее помехоустойчивость и обеспечит для большинства задач автоматическую их адаптацию к архитектуре МВК "Эльбрус-2", оставив для пользователя только задачу организации параллельных вычислительных процессов.

Кроме того, переход к обменам большими сегментами с внешней памятью на дисках позволит полностью перейти к стандартному интерфейсу ЕС ЭВМ, в результате чего существенно упростится проблема внедрения новых образцов внешней памяти.

Заключение

Проведенные испытания показали, что задача достижения предельной производительности вычислительных средств на заданной элементной базе решена полностью, а именно:

- достигнута предельная производительность на один процессор;
- решен вопрос одновременной работы многих процессоров (14) на общую оперативную и внешнюю память практически без снижения производительности каждого процессора;
- программирование полностью переведено на языки высокого уровня;
- обеспечена высокая структурная надежность комплекса.

Многопроцессорная модульная архитектура комплекса, предусматривающая широкое использование специализированных процессоров, впервые полностью реализована в МВК "Эльбрус-2". По многим архитектурным решениям МВК "Эльбрус-2" превосходит зарубежные аналоги. Принятая модульная многопроцессорная архитектура в настоящее время находит широкое использование в зарубежной практике построения мощных вычислительных комплексов суперЭВМ.

Учитывая тот факт, что современная элементно-конструкторская база по быстродействию близка к физическому пределу, в результате чего предел производительности одного универсального процессора с одним потоком команд на скалярных операциях не может превышать 100 млн. операций в секунду [2], модульный многопроцессорный принцип Построения вычислительного комплекса позволяет качественно расширить предел производительности универсальных вычислительных комплексов суперЭВМ.

Реально вычислительные процессы не бывают чисто скалярными или чисто векторными. Поэтому только модульный многопроцессорный комплекс с широким использованием векторных процессоров, для которых физический предел производительности значительно выше, позволяет создавать на элементной базе с $\tau_Э = 0{,}1$ не комплексы общей производительностью в десятки миллиардов операций в секунду. Так, МВК "Эльбрус-2" с векторными процессорами, разработанными в его составе на той же эле-

ментно-конструкторской базе, обепечивал бы производительность комплекса, близкую к одному миллиарду операций в секунду [4].

Создание МВК "Эльбрус-2" явилось важным достижением в разработке суперЭВМ и внесло весомый вклад в практику проектирования мощных вычислительных комплексов.

В проекте имелись и неудачные решения, которые в процессе разработки, к сожалению, не было возможности исправить. К ним необходимо отнести следующее:

1. Концепция, принятая в ОС по формированию сегментов, приводит к распределению ресурсов оперативной и внешней памяти сегментами произвольной длины малых объемов, в результате чего:

- появилась необходимость реализации специального, отличного от принятого в ЕС ЭВМ, интерфейса обмена информации ОЗУ с дисками и барабанами;
- значительно усложнилась сама операционная система в части распределения ресурса памяти по сравнению со страничной организацией виртуальной памяти;
- существенно снизился коэффициент многопрограммности;
- система не обладает достаточной помехозащищенностью к сбоям и отказам аппаратуры;
- для большого круга задач существенно усложнился вопрос адаптации задач к вычислительному комплексу;
- существенно затормозился процесс развития внешней памяти МВК "Эльбрус-2" в части внедрения новых устройств, разработанных для ЕС ЭВМ.

2. Универсальным языкам должного внимания уделено не было, в результате чего:

а) представленные на испытания трансляторы языков высокого уровня не имели необходимых блоков оптимизации кодов и были недостаточно обкатаны;

б) вопросом совместимости программ с ЕС ЭВМ и БЭСМ-б, написанных на языках высокого уровня, достаточного внимания уделено не было.

3. К неудачам разработки необходимо отнести также решение руководства о прекращении изготовления векторного процессора производительностью 240 млн, оп/с, разработанного на элементно-конструкторской базе МВК "Эльбрус-2". Решение было принято на основании информации о том, что аналогичный процессор МКП на новой элементной базе значительно большей производительности может быть изготовлен для МВК "Эльбрус-2" и поставлен пользователю в те же сроки.

Исходя из создавшегося положения, наиболее реальным развитием работ по МВК "Эльбрус-2" нам представляется следующее.

1. Необходимо устранить следующие недостатки системного математического обеспечения, сдерживающие его использование в режиме вычислительного центра коллективного пользования (ВЦКП):

- расширить круг задач, эффективно реализуемых в многопрограммном режиме ВЦКП;
- существенно повысить помехоустойчивость комплекса к сбоям и отказам;
- повысить эффективность работы комплекса на универсальных языках высокого уровня;
- вести работы в направлении обеспечения совместимости комплекса по программам, написанным на языках высокого уровня, с высшими моделями ЕС ЭВМ.

2. Исключить из комплекса нестандартные внешние устройства, такие как барабан, специальное управление дисками и перейти полностью на стандартные интерфейсы работы ЕС ЭВМ с внешними устройствами.

3.Возобновить работы по изготовлению векторного процессора на элементной базе МВК "Эльбрус-2" с тем, чтобы в 1990 году он серийно выпускался в составе МВК "Эльбрус-2".

4.Вести исследования по эффективности работы МВК "Эльбрус-2" в режиме ВЦКП с преимущественным решением больших задач и на основании этих данных провести корректировку аппаратных решений и общего системного программного обеспечения.

5.Доработанный по результатам проведенных исследований МВК "Эльбрус-2" с векторными процессорами перевести на новую элементную базу.

Вычислительный комплекс МВК "Эльбрус-2", переведенный на новую элементную базу и дополненный спецпроцессорами: векторным, искусственного интеллекта и управления базой данных на большом диапазоне различного класса задач, будет обладать реальной производительностью, существенно превосходящей разрабатываемые сегодня зарубежные сверхвысокопроизводительные вычислительные комплексы.

Дальнейшее развитие архитектурных вычислительных средств должно идти в направлении предельного распараллеливания вычислительного процесса, очевидно, с переходом на не фон-неймановские принципы реализации вычислений.

Новые архитектурные решения суперЭВМ в этом случае наиболее эффективно могут быть реализованы с использованием элементной базы, построенной на новых физических принципах.

Литература

1. В.С.Бурцев. Надежностные характеристики многопроцессорных комплексов . и анализ надежности МВК "Эльбрус-2". М., 1987 (Препринт ОВМ АН СССР № 169).
2. В.С.Бурцев. Тенденции развития высокопроизводительных систем и многопроцессорные вычислительные комплексы. М., 1977 (ИТМиВТ АН СССР).
3. В.С.Бурцев. Принципы построения многопроцессорных вычислительных комплексов "Эльбрус". М., 1977. с. 53 (Препринт ИТМиВТ №1).
4. В.С.Бурцев, Е.А.Кривошеев, В.Д.Асриэли и др. Векторный процессор для семейства МВК "Эльбрус". - В кн.: Актуальные проблемы развития архитектуры и программного обеспечения ЭВМ и вычислительных систем. Труды Всесоюзной конференции. - Новосибирск: ВЦ СО АН СССР, 1983.
5. В.С.Бурцев, Е.А.Кривошеев, В.Д.Асриэли и др. Векторный процессор с программируемой структурой. - В кн.: Вычислительные процессы и системы. Вып.2. М: Наука, 1985.
6. А.П.Иванов, В.С.Шевяков. Управление памятью в МВК "Эльбрус-1". М., 1981. (Препринт ИТМиВТ № 3).
7. В.П.Торчигин. Система файлов МВК "Эльбрус-1"- М., 1977. (Препринт ИТМиВТ, № 3).
8. А.П.Иванов, С.В.Семенихин. Операционная система МВК "Эльбрус-1". М., . 1977. (Препринт ИТМиВТ № 2).
9. Л.Е.Пшеничников, Ю.Х.Сахин. Архитектура многопроцессорного вычислительного комлпекса "Эльбрус". М., 1977. (Препринт ИТМиВТ № 7).
10. Б.А.Бабаян. Уровень программирования и архитектурные принципы построения ЭВМ. Кибернетика и вычислительная техника, 1986. вып.2.
11. CDC CYBER-200 Model 205 Technical description. Control data Corporation. November 1980.

Содержание

В.С. Бурцев

Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ

МВК "Эльбрус"

Оригинал-макет подготовлен в ИВВС РАН на персональном компьютере IBM PC

Подписано в печать 5.10.98. Заказ 549 Тираж 500 экз.
Отпечатано в типографии издательства "НЕФТЬ И ГАЗ"