

Лекция 3

Управление памятью

Распределение памяти предполагает удовлетворение потребностей как пользователей, так и системных средств. Эти требования в большей части противоречивы.

Системные цели:

- увеличение степени использования оперативной памяти при параллельном развитии нескольких процессов в мультипрограммном режиме,
- защита информации при развитии этих процессов,
- обеспечение взаимодействия между процессами и т. д.

Требования пользователей:

- быстрое выполнение коротких программ,
- выделение оперативной памяти в размерах, превышающих физически существующую,
- легкость и простота взаимодействия с другими программами при использовании, например, общих процедур и т. п.

Функции системы управления памятью :

- учет состояния свободных и уже распределенных областей памяти и модернизация этой информации всякий раз, когда в распределении памяти производятся изменения;
- распределение памяти для выполнения задач (определение, какой задаче, когда и в каком количестве выделить оперативную память);
- непосредственное выделение задаче оперативной памяти; если свободные области оперативной памяти отсутствуют, то предварительное их освобождение путем сохранения информации во внешней памяти.

ПРИМЕР СТАТИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ПАМЯТИ

Исходное состояние:

$V_{\text{оп}} = 10 \text{ МБ}$,

для выполнения программ A , B , C , D требуются следующие объемы памяти: $A - 2 \text{ МБ}$, $B - 1 \text{ МБ}$, $C - 4 \text{ МБ}$, $D - 4 \text{ МБ}$.

A		свободно		свободно		
B		B				D
C		C		C		
свободно		свободно		свободно		свободно

а)

б)

в)

г)

а – начальное распределение;
б – после завершения программы A ;
в – после завершения программы B ;
г – после завершения программы C

Современные системы распределения памяти опираются на две концепции:

- динамического использования ресурсов и
- виртуализации.

При динамическом распределении памяти каждой программе в начальный момент выделяется лишь часть от всей необходимой ей памяти, а остальная часть выделяется по мере возникновения реальной потребности в ней.

Такой подход базируется на следующих предпосылках.

Во-первых, при каждом конкретном исполнении в зависимости от исходных данных некоторые части программы (до 25% ее длины) вообще не используются. Следует стремиться к тому, чтобы эти фрагменты кода не загружались в ОП.

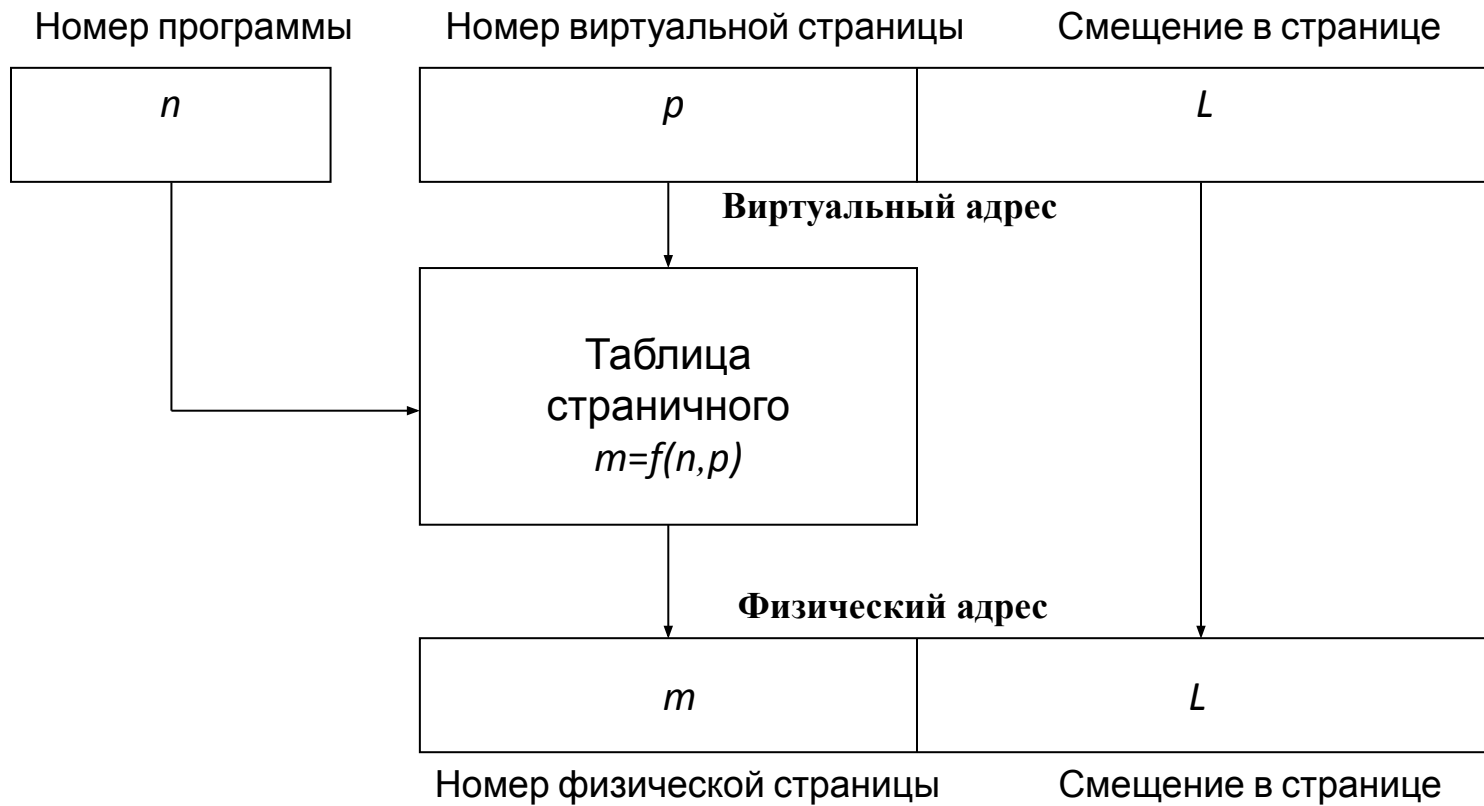
Во-вторых, исполнение программы характеризуется так называемым *принципом локальности ссылок*. Он подразумевает, что при исполнении программы в течение некоторого относительно малого интервала времени происходит обращение к памяти в пределах ограниченного диапазона адресов (как по коду программы, так и по данным). Следовательно, на протяжении этого времени нет необходимости хранить в ОП другие блоки программы.

Виртуальная память, обеспечивая возможность программисту обращаться к очень большому объему непрерывного адресного пространства, предоставляемого в его монопольное распоряжение, обладает обычными свойствами: побайтовая адресация, время доступа, сравнимое со временем доступа к оперативной памяти.

Принцип виртуальной памяти предполагает, что пользователь при подготовке своей программы имеет дело не с физической ОП, действительно работающей в составе компьютера и имеющей некоторую фиксированную емкость, а с виртуальной (кажущейся) одноуровневой памятью, емкость которой равна всему адресному пространству, определяемому размером адресной шины ($L_{\text{ша}}$) компьютера:

$$V_{\text{вирт}} \gg V_{\text{физ}},$$

$$V_{\text{вирт}} = 2^{L_{\text{ша}}}.$$



Принцип преобразования виртуального страничного адреса в физический

Пример преобразования адреса виртуальной страницы в адрес физической страницы

Пусть компьютер использует адресное пространство, предполагающее разбиение на страницы объемом $V_{\text{стр}}=1$, и имеет оперативную память $V_{\text{ОЗУ}}=3$ страницы.

Пусть на компьютере одновременно выполняются четыре программы, имеющие следующее количество страниц: $V_A=2$, $V_B=1$, $V_C=3$, $V_D=2$.

Переключение между программами происходит через $t_k = 1$.

Время выполнения каждой страницы любой программы составляет $t = 2t_k$.

Страницы программ загружаются в оперативную память по мере их необходимости и, по возможности, в свободные области ОЗУ.

Если вся память занята, то новая страница замещает ту, к которой дольше всего не было обращений (механизм LRU).

Тогда таблица загрузки оперативной памяти и таблицы страничного преобразования для каждой программы будут следующий вид.

Страница	Такты															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Динамическое распределение оперативной памяти															
ОЗУ 0	A0	A0	A0	D0	D0	D0	C0	C0	C0	C1	C1	C1	C1	C1	C1	C1
1		B0	B0	B0	A0	A0	A0	D0	D0	D0	D1	D1	D1	D1	D1	D1
2			C0	C0	C0	B0	B0	B0	A1	A1	A1	A1	A1	A1	c2	c2
	Таблица страничного преобразования для программы А															
A 0	0	0	0	-	1	1	1	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	2	2	2	2	2	2	-	-
	Таблица страничного преобразования для программы В															
B 0	-	1	1	1	-	2	2	2	-	-	-	-	-	-	-	-
	Таблица страничного преобразования для программы С															
C 0	-	-	2	2	2	-	0	0	0	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2
	Таблица страничного преобразования для программы D															
D 0	-	-	-	0	0	0	-	1	1	1	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-	1	1	1	1	1	1

Архитектура компьютера различает физическое адресное пространство (ФАП) и логическое адресное пространство (ЛАП).

Физическое адресное пространство – это память, организованная как простой одномерный массив байтов, доступ к которому реализуется аппаратурой памяти по адресу, присутствующему на шине адреса микропроцессорной системы (МПС).

Логическое адресное пространство организуется программистом исходя из своих потребностей. Процессор автоматически транслирует логические адреса в физические, выдаваемые затем на системную шину. Трансляция логических адресов программ в физические осуществляет диспетчер памяти (блок управления памятью *memory manager unit* – *MMU*).

Варианты организации логического адресного пространства :

– плоское (линейное) ЛАП; состоит из массива байтов, не имеющего определенной структуры; трансляция адреса не требуется, так как логический адрес совпадает с физическим;

– сегментированное ЛАП; Состоит из сегментов, каждый из которых в общем случае содержит переменное число байтов; логический адрес содержит 2 части: идентификатор сегмента и смещение внутри сегмента. Трансляцию адреса проводит MMU;

– страничное ЛАП; состоит из страниц, каждая из которых содержит фиксированное число байтов; логический адрес состоит из номера (идентификатора) страницы и смещения внутри страницы; трансляция проводится блоком управления памятью MMU;

– сегментно-страничное ЛАП; состоит из сегментов, которые, в свою очередь, состоят из страниц; логический адрес состоит из идентификатора сегмента и смещения внутри сегмента. MMU производит трансляцию логического адреса в номер страницы и смещение в ней, которые затем транслируются в физический адрес.

Режимы формирования физического адреса:

- режим реальных адресов (реальный режим);
- режим защищенной памяти (защищенный режим).

При работе в реальном режиме возможности процессора ограничены:

- объем адресуемой памяти составляет 1 Мб,
- длина сегмента постоянная и равна 2^{16} байт,
- отсутствует страничная организация памяти,
- отсутствуют механизмы защиты.

Этот режим обычно используется на начальном этапе загрузки компьютера для перехода в защищенный режим.

Формирование физического адреса в реальном режиме работы микропроцессора

ФА - физический адрес

Абаз - базовый адрес сегмента

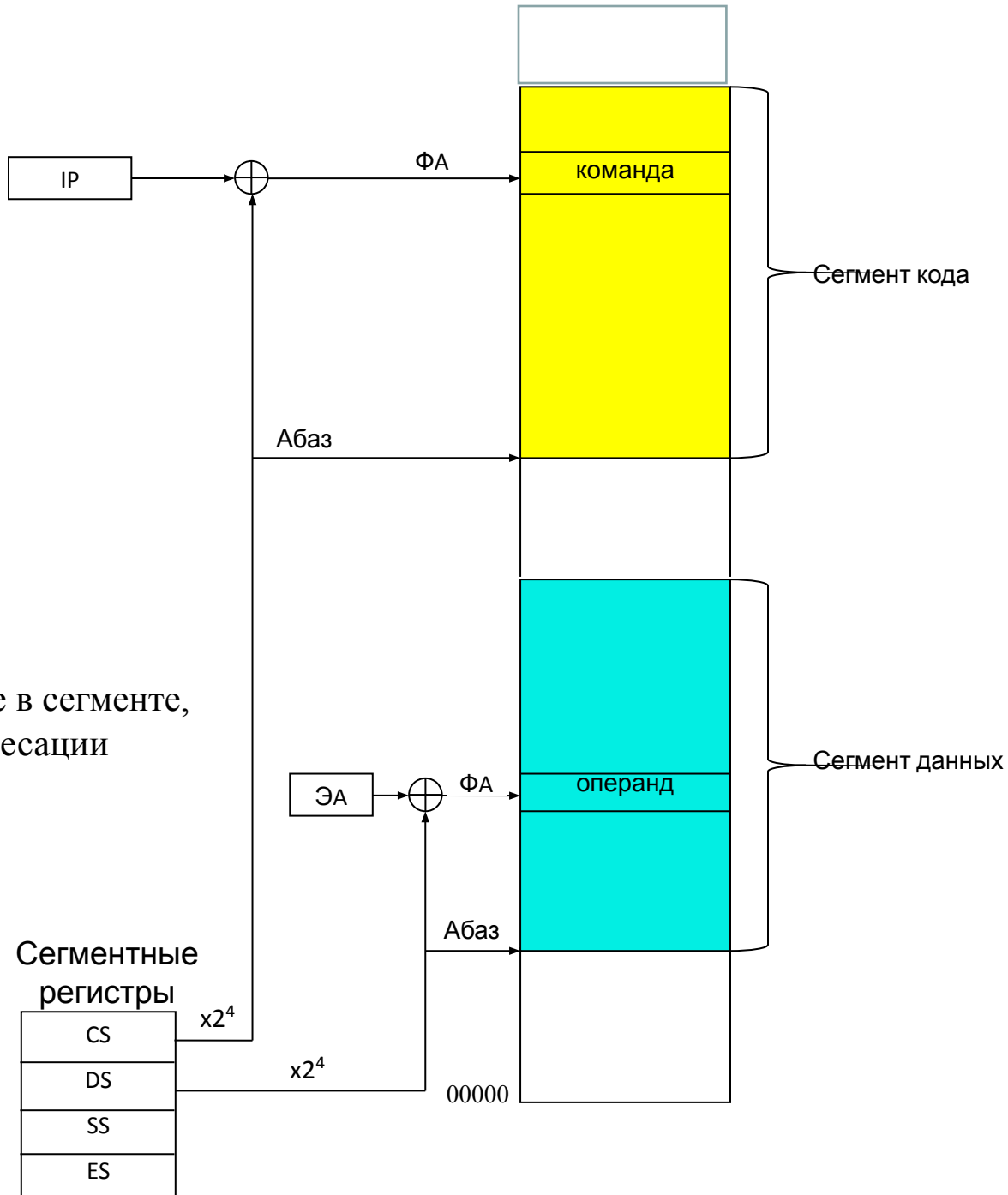
ЭА - эффективный адрес (смещение в сегменте, формируемое исходя из режима адресации операнда)

Для команды

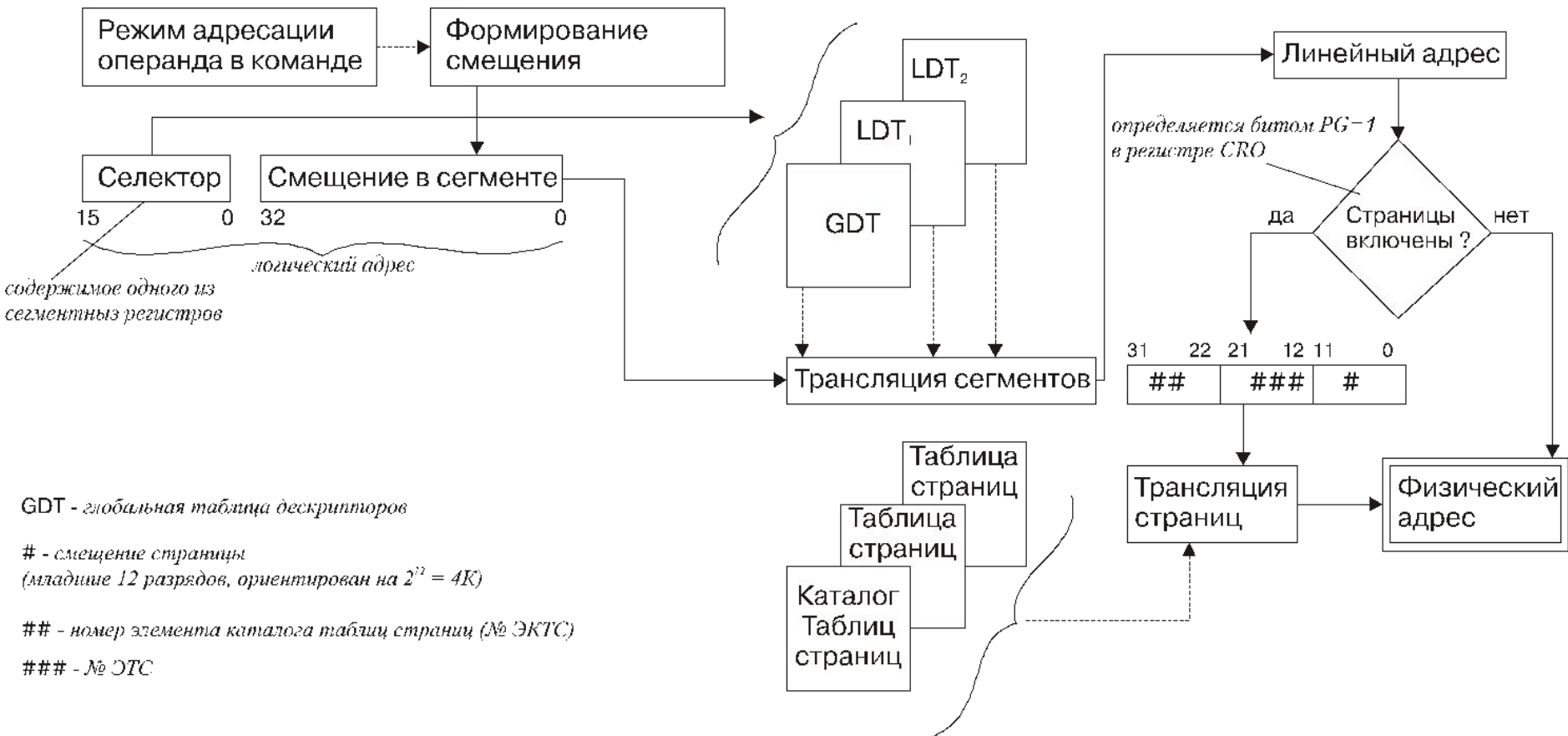
`ADD AX,[CX+DI+25]`

эффективный адрес операнда

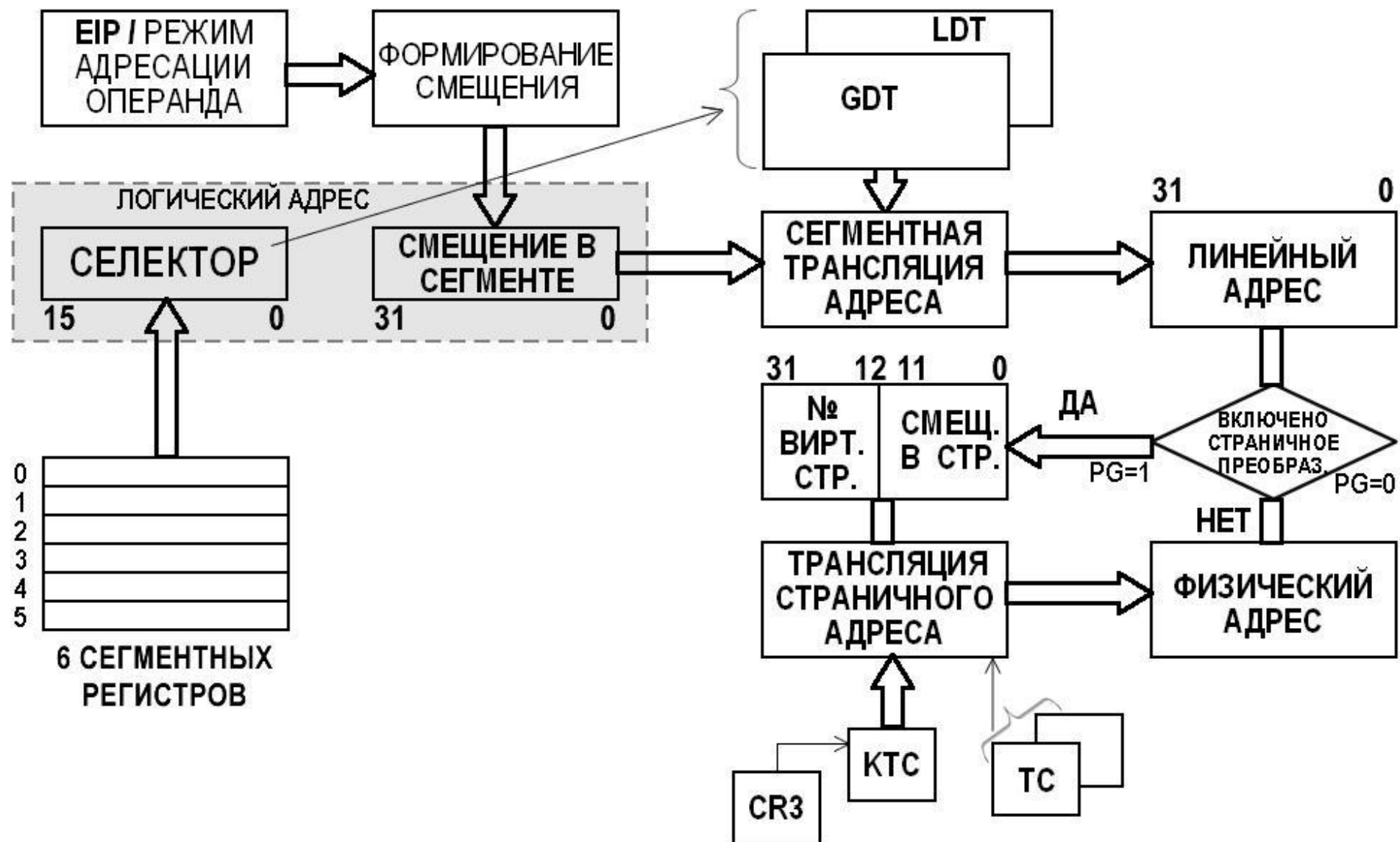
$ЭА = [CX] + [DI] + 25$



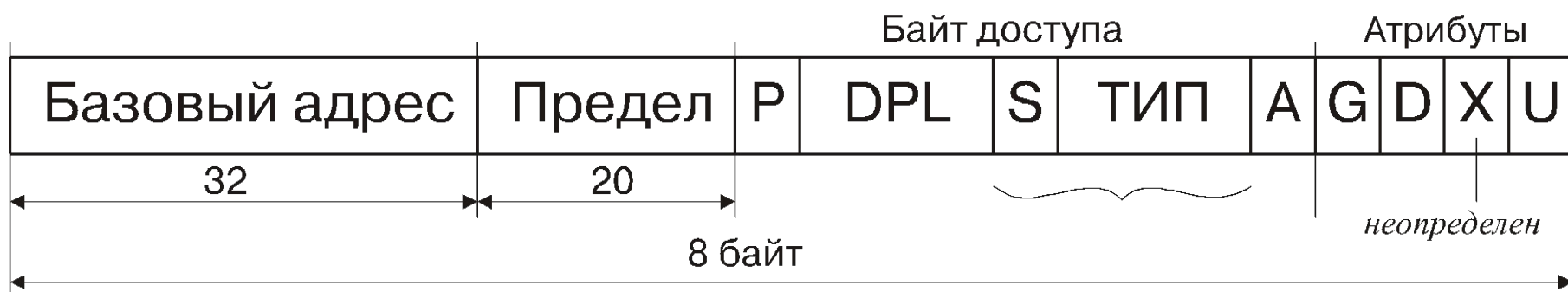
Обобщённая схема формирования адреса при сегментно-страничной организации памяти в защищённом режиме работы



Обобщённая схема формирования адреса при сегментно-страничной организации памяти



Структура дескриптора сегмента



$$G \text{ (бит гранулярности)} = \begin{cases} 0, & \text{длина сегмента в байтах} \\ 1, & \text{длина сегмента в страницах} \end{cases}$$

$$V_{\text{сегм max}} = \underbrace{2^{20}}_{\substack{\text{страниц} \\ =1\text{Мб}}} \cdot \underbrace{2^{12}}_{\substack{\text{байт/стр.} \\ =4\text{Кб}}} = \underbrace{2^{32}}_{\substack{2^{30} \cdot 2^2 \\ 1\text{Гб} \cdot 4}}$$

Бит разрядности D (demention) определяет длину адресов и операндов, используемых в команде по умолчанию.

Бит пользователя U (User) может использоваться системными программистами по своему усмотрению. Доступен только на высшем уровне привилегий. Микропроцессор в своей работе этот бит игнорирует.

Бит обращения A: устанавливается в "1" при любом обращении к сегменту. Используется операционной системой для того, чтобы отслеживать сегменты, к которым дольше всего не было обращений.

Байт доступа определяет основные правила обращения с сегментом.

Бит присутствия P показывает возможность доступа к сегменту.

При $P = 1$ сегмент находится в физической памяти. Если $P=0$ (сегмент отсутствует), поля базового адреса и предела дескриптора игнорируются. При этом процессор отвергает все последующие попытки использовать дескриптор в командах. Операционная система копирует запрошенный сегмент с диска в память (при этом, возможно, удаляя другой сегмент), загружает в дескриптор новый базовый адрес сегмента, устанавливает $P=1$ и осуществляет рестарт, то есть повторное выполнение “виноватой” команды. Описанный процесс называется “свопингом”, или подкачкой.

Поле DPL (Descriptor privilege level) указывает уровень привилегий дескриптора, определяемый возможность доступа к сегменту со стороны тех или иных программ.

ФОРМАТ ПОЛЯ ТИПА БАЙТА ДОСТУПА

4	3	2	1	0	
1	1	C	R	A	Сегмент кода
<hr/>					
1	0	ED	W	A	Сегмент данных
<hr/>					
0		Т и п			Системный объект

В сегменте данных:

бит направления расширения ED (*Expand Down*): ED = 1 – сегмент стека (относительный адрес должен быть больше размера сегмента), при ED = 0 – сегмент собственно данных (относительный адрес должен быть меньше или равен размера сегмента);

бит разрешения записи W (*Writeable*): при W = 1 – разрешено изменение сегмента, при W = 0 запись в сегмент запрещена; при попытке записи в сегмент возникает исключение по защите памяти.

В сегменте кода:

бит *C (Conforming)*: бит подчинения или согласования – определяет дополнительные правила обращения, которые обеспечивают защиту сегментов программ (специальный порядок обращения к подчиненным и неподчиненным сегментам – разрешение или запрещение межсегментных переходов); При $C = 1$ – подчиненный сегмент кода. В этом случае сегмент намеренно лишается защиты по привилегиям. Такое средство удобно для организации в системе библиотек, программы которых должны быть доступны всем выполняющимся в системе задачам. Библиотечные программы оформляются как подчиненные сегменты кода и они могут быть вызваны командой типа *CALL* любой текущей задачей. При $C = 0$ – обычный сегмент кода;

бит *R (Readable)* – бит считывания. Устанавливает, можно ли обращаться к сегменту только на исполнение или на исполнение и считывания (например, констант) как данных с помощью префикса замены сегмента. При $R = 0$ допускается только выборка из сегмента для выполнения через *CS*, при $R = 1$ чтение из сегмента разрешено.

При любой попытке записи в сегмент кода возникает прерывание.

ФОРМАТ СЕЛЕКТОРА



- Index: поле индекса – номер дескриптора в соответствующей таблице дескрипторов.
- TI (Table Indicator): индикатор таблицы – показывает, к какой таблице идёт обращение.

$$TI = \begin{cases} 0 - \text{обращение к GDT} \\ 1 - \text{обращение к LDT} \end{cases}$$

- RPL (Request privilege level) – уровень привилегий запроса. При обращении сравнивается с полем DPL в байте доступа дескриптора. Обращение разрешается, если уровень привилегий запроса не ниже, чем уровень привилегий дескриптора.

ПОЛУЧЕНИЕ ДЕСКРИПТОРА, НАХОДЯЩЕГОСЯ В ГЛОБАЛЬНОЙ ТАБЛИЦЕ ДЕСКРИПТОРОВ

