Архитектура ЭВМ и систем

Лекция № 5

Организация работы процессора

План лекции

1. Классификация и структура команд процессора.
2. Способы адресации данных и команд.

2.1. Способы адресации данных

2.2 Способы адресации команд

3. Вычисление физического адреса.
4. Поток управления и механизм прерываний.

Организация работы процессора

Система команд является одной из важнейших архитектурных характеристик процессора и ВМ в целом. Она определяет совокупность операций, реализуемых процессором.

В понятие *«система команд»* входят:

1) форматы команд и обрабатываемых данных;
2) список команд и их функциональное назначение;
3) способы адресации данных и команд.

Классификация и структура команд процессора

По функциональному признаку все команды процессора можно разделить на следующие группы:

1) команды пересылки данных и ввода – вывода;
2) команды арифметических и поразрядных логических операций;
3) команды передачи управления.

Классификация и структура команд процессора

Команды пересылки данных обеспечивают обмен информацией между регистрами микропроцессора, а также внешние обмены данными при передаче в процессор из памяти или устройства ввода и из процессора в память или устройство вывода.

В этих командах обычно указывается направление передачи, источник и (или) приёмник данных.

В языке Ассемблера к командам этой группы можно отнести команду пересылки *MOV*, команду загрузки *LOAD*, команды записи в порт и чтения из порта УВВ, *IN* и *OUT*, соответственно. Также сюда часто включают команды помещения данных в стек *PUSH* и извлечения данных из стека *POP*.

Примеры:

mov ax,4; *Переслать в регистр **ax** значение **4***

mov al,a; *Переслать в регистр **al** значение по адресу **a***

mov a,al; *Переслать в ячейку по адресу **a** содержимое регистра **al***

in al,61h; *Считать в регистр **al** значение порта **61h***

out 61h,al; *Вывести в порт **61h** содержимое регистра **al***

Порт – адрес устройства**.**

Классификация и структура команд процессора

В число *команд арифметических и поразрядных логических операций* в большинстве случаев входят команды простейших арифметических операций, например, ADD (сложить), SUB (вычесть), а также логических операций, например, AND («И»), OR («ИЛИ») и т.п.

К арифметическим командам относят также команды арифметических и логических сдвигов, а к командам логических операций – команда сравнения COMPARE .

В число команд этой группы могут входить команды сложных арифметических операций: умножение, деление (есть не во всех процессорах), команды обработки данных с плавающей точкой, команды мультимедийной обработки.

Примеры:

add ax,4; *Сложить содержимое регистра* ***ax*** *со значением* ***4***

sub al,a; *Вычесть из регистр* ***al*** *значение по адресу* ***a***

mul a; *Умножить значение по адресу* ***a*** *на содержимое регистра* ***al***

or al,00000001b; *Установить в регистре* ***al*** *значение нулевого*
; *бита в 1*

and ah,0; *Сбросить все биты регистра* ***al*** *в 0*

Классификация и структура команд процессора

Команды передачи управления используются для изменения последовательности выполнения команд при наличии программных ветвлений: команд условных и безусловного (JMP) переходов, обращении к подпрограммам (CALL) и выхода из них (RETURN).

Команды условных переходов реализуют передачи управления в зависимости от значения флагов в регистре PSW.

С их помощью процессор выполняет одну из возможных ветвей продолжения программы. Обычно в системе команд имеется несколько команд условных переходов.

Пример:

…

jmp m1; *Перейти на команду с адресом* ***m1***

…

m1: mul a; *Умножить значение по адресу* ***a*** *на содержимое* **;** *регистра* ***al***

Классификация и структура команд процессора

Современные процессоры также содержат группы команд:

1. расширяющие функциональные возможности процессора по обработке информации и управлению его работой,
2. обеспечивающие реализацию многозадачного защищённого режима работы,
3. дополнительные команды.

Выполнение команды (машинной операции) разделено на более мелкие этапы – ***микрооперации*** (микрокоманды), во время которых выполняются определённые элементарные действия. Конкретный состав микроопераций определяется системой команд и логической структурой ВМ.

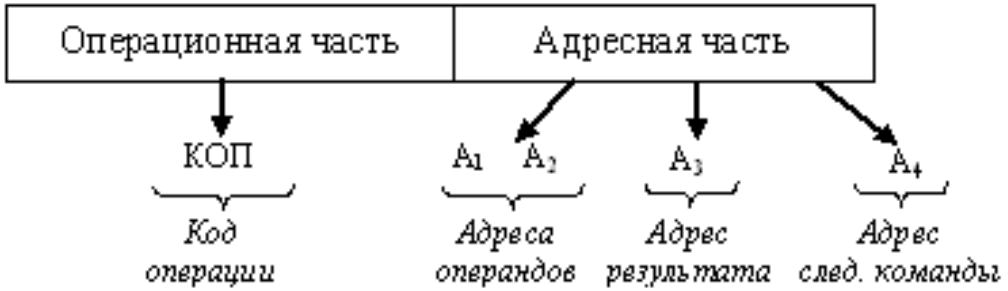
Последовательность микрокоманд, реализующих данную операцию (команду), образует ***микропрограмму операции***.

Интервал времени, в течение которого выполняется одна или одновременно несколько микроопераций, называется ***машинным тактом***. Границы тактов задаются синхросигналами, которые вырабатываются генератором синхросигналов.

Классификация и структура команд процессора

Команда микропроцессора содержит две части:

- *операционную* и
- *адресную.*



В *операционной* части команды содержится код операции (*КОП*), обеспечивающий кодирование 2^n операций (*n* – число разрядов операционной части команды) и определяющий, какие устройства в процессоре или вне его при этом будут задействованы.

В *k-разрядной адресной* части команды в общем случае могут содержаться *4* адресных поля *A1*, *A2*, *A3*, *A4*.

- ***A1, A2*** – адреса операндов, участвующих в операции;
- ***A3*** – адрес результата;
- **A4** – адрес следующей выполняемой команды.

В качестве адресов A1,…,A3 могут использоваться адреса ячеек оперативной памяти и адреса регистров микропроцессорной памяти, в качестве адреса A4 – только адреса ячеек оперативной памяти.

Классификация и структура команд процессора

В зависимости от указываемого числа адресов команды подразделяются на:

- 0-адресные (безадресные).
  nop – ничего не делать.
- 1-адресные.
  inc ax – увеличить содержимое регистра *ax* на 1;
  push ax – поместить содержимое регистра *ax* в вершину стека.
- 2-адресные.
  add ah,al – сложить содержимое регистров *ah* и *al*.
- 3-адресные.

  Операция сложения, например, могла бы выглядеть следующим образом:
  add ah,al,bx - сложить содержимое регистров *ah* и *al* с сохранением результата в регистре *bx.*
- 4-адресные.

  Например, операция сложения могла быть записана:
  add ah,al,bx, 0020 - сложить содержимое регистров *ah* и *al* с сохранением результата в регистре *bx* и последующей загрузкой команды по смещении *0020* в сегменте кода.

Классификация и структура команд процессора

Практически во всех микропроцессорах адрес *A4* исключён. Большинство команд программы выполняются последовательно и могут быть размещены в ячейках памяти с последовательно возрастающими адресами. Тогда для получения адреса следующей команды к начальному адресу сегмента кода достаточно добавить её смещение в сегменте кода, что удобно реализовать с помощью указателя команд *IP*.

Такой способ адресации команд называется ***естественным***, а реализующие его процессоры называются ***процессорами с естественным способом адресации команд***.

При нарушении естественного порядка следования команд (ветвлениях, циклах) используются специальные команды передачи управления, в которых содержится адрес перехода, но не используются адреса операндов.

Процессоры, в адресном поле команд которых используется адрес *A4*, называются ***процессорами с принудительным способом адресации команд***.

Классификация и структура команд процессора

Адрес результата *A3* во многих случаях также оказывается избыточным. Результат арифметических и логических операций над двумя операндами обычно может быть помещён на место одного из операндов.

При этом в 2-адресных командах в адресное поле необходимо вводить дополнительные разряды, показывающие, кто из них является источником, а кто – приёмником информации.

В процессорах с аккумуляторной архитектурой число адресов в адресной части команды уменьшено до одного. В них один из операндов, размещённых в аккумуляторе, неявно задаётся кодом команды, и результат помещается в аккумулятор.

В *безадресных* командах осуществляется неявное задание операнда. К таким командам относятся команды управления процессором (например, пуска, останова и т.д.) и команды для работы со стеком (операнд, адресуемый указателем *SP*, неявно задаётся кодом команды).

Безадресные команды имеют предельно сокращённый формат, но не могут самостоятельно образовать функционально полную систему команд и поэтому применяются только вместе с адресными.

Классификация и структура команд процессора

Формат команд влияет на время решения задач, затраты памяти, сложность процессора и зависит от класса решаемых задач.

Для научно- технических расчётов, в которых большой объём занимают многошаговые вычисления, более эффективными оказываются 1-адресные команды, а при использовании стекового процессора – и безадресные команды.

Для задач управления, где большую долю составляют пересылки и логические операции, эффективными являются 2-адресные команды.

В современных процессорах обычно используются безадресные, 1-адресные и 2-адресные команды. 3-адресные команды используются крайне редко, а 4-адресные не используются совсем.

Способы адресации данных

1. *Непосредственный*.

Позволяет задавать фиксированные значения операнда непосредственно в адресной части команды, т.е., искомое значение является частью команды.

Такой режим адресации удобен при работе с константами.

Непосредственный операнд может быть задан только как операнд-источник.

Недостатком непосредственной адресации является необходимость расширения формата команд за счёт указания самого операнда в адресном поле команды.

Непосредственная адресация данных

mov ax, 5564h; Переслать в регистр *ax* значение *5564*
; в шестнадцатеричной системе счисления

mov ah, 'A'; Переслать в регистр *ah* символ *A*

add al, 11010011b; Сложить содержимое регистра *al* с числом
; *11010011* в двоичной системе счисления

Способы адресации данных

2. *Прямой*.
   Адрес операнда (смещение) содержится в коде команды.
   Используется при работе с переменными и константами, местоположение которых в памяти не меняется в процессе выполнения задачи.

Прямая адресация

```
d_s segment
 mm dw 3154h
d_s ends
c_s segment
assume ds:d_s, cs:c_s
begin:
 …
 mov ax, mm ;по адресу mm
            ;пересылается 3154h
 …
c_s ends
end begin
```

После выполнения выделенной подчёркиванием команды в регистре *ax* будет записано значение по адресу *mm* в памяти, т.е., число *3154h*.

Способы адресации данных

3. Регистровый. Искомое значение операнда содержится в определённом командой регистре, т.е., в адресном поле команды указывается адрес регистра.

Отличительная особенность регистровой адресации – все операнды команд являются регистрами. Такие команды являются наиболее компактными и выполняются быстрее других типов команд, поскольку отсутствуют обращения к памяти.

Регистровая адресация

mov ax, cx; Переслать в регистр *ax* содержимое
; регистра *cx*

add ah, al; Сложить содержимое регистров *ah* и *al*

Способы адресации данных

4. *Регистровый косвенный*.

Адрес, указываемый в команде, является указателем ячейки, содержащей смещение операнда в памяти.

Фактически в команде указывается адрес адреса, в качестве регистра адреса может выступать базовый регистр *BP* или индексные регистры *SI* или *DI*. В разных процессорах для размещения адреса могут использоваться и другие регистры.

Косвенная адресация является более эффективной, чем прямая, поскольку указывается только адрес регистра, который короче полного адреса операнда в памяти. Однако при этом требуется предварительная загрузка регистра косвенным адресом памяти, на что расходуется дополнительное время.

Косвенную адресацию удобно, когда не меняя адрес регистра в команде, можно изменять содержимое ячейки с этим адресом.

Регистровая косвенная адресация

Способы адресации данных

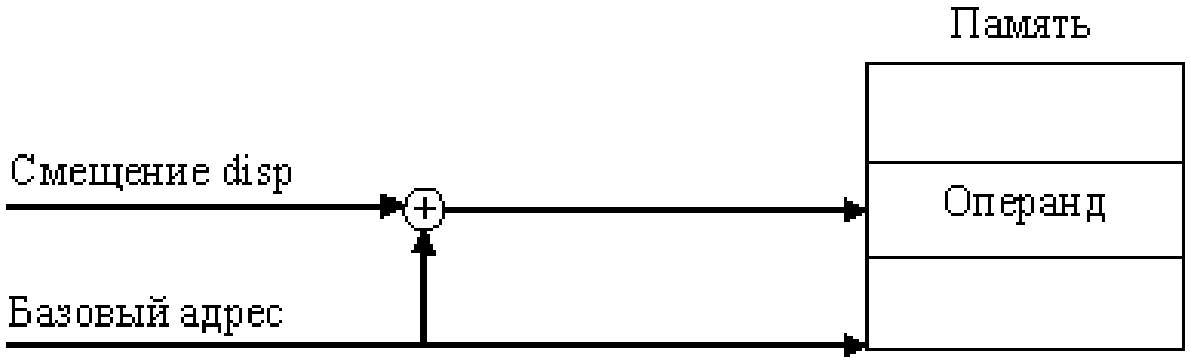
```
d_s segment
 mm dw 3154h
d_s ends
c_s segment
assume ds:d_s, cs:c_s
begin:
…
 lea si, a;        загрузить в регистр si адрес ячейки mm
 mov ax, [si] ; в регистр ax пересылается значение по адресу,
                ; указанному в регистре si (т.е. число 3154h)
…
c_s ends
end begin
```

Способы адресации данных

5. *Регистровый относительный*.

Является обобщением методов адресации, обеспечивающих вычисление эффективного адреса (*EA*) или смещения операнда в памяти в виде суммы базового значения адреса и «смещения» *disp*, указываемого в команде.

Относительную адресацию широко применяют как для адресации памяти, представленной в виде блоков (например, сегментов), так и для адресации специальных структур данных: массивов, записей и др.



Формирование смещения при относительной адресации

$$EA = \begin{Bmatrix} BP \\ BX \\ SI, DI \end{Bmatrix} + \{Смещение\ disp\}.$$

Вычисление смещения при относительной адресации

Способы адресации данных

В зависимости от способа использования адресуемого в команде регистра различают *базовый* и *индексный* режимы адресации.

5.1. *Индексный*.

Применяется для обработки упорядоченных массивов данных; при этом каждый элемент массива определяется собственным номером. Тогда базовый адрес массива задаётся смещением *disp*, указываемым в команде, а значение индекса (номер ячейки с элементом массива) определяется содержимым индексного регистра.

Индексная адресация удобна, если необходимо записать или считать список данных из последовательных ячеек памяти не подряд, а с некоторым шагом, указанным в индексе.

$$EA = \begin{Bmatrix} SI \\ DI \end{Bmatrix} + disp.$$

Вычисление смещения операнда
при индексном режиме адресации

Способы адресации данных

```
d_s segment
 mas db 3,5,1,8,9,'$'; По адресу mas определена последовательность
                       ; из 6 однобайтовых элементов
                       ;(с учётом символа $)
                       ; $ - признак конца последовательности
d_s ends
c_s segment
assume ds:d_s, cs:c_s
begin:
…
 mov si,3 ;в si – смещение элемента (8) относительно начала
          ; массива, т.е., адреса mas
mov ah, mas[si] ;mas- смещение массива в сегменте данных
                ;в ah – значение элемента массива mas
                ;со смещением в si, т.е. 8
…
c_s ends
end begin
```

Начало массива определяется адресом *mas*; требуемый элемент (*8*) является третьим (индексация начинается с нуля); каждый элемент массива занимает 1 байт в памяти. Для нахождения данного элемента в сегменте данных в индексный регистр *SI* помещается номер ячейки в массиве с нужным элементом (*3 = 1×3*) и складывается с начальным адресом массива *mas*.

Способы адресации данных

5.2. *Базовый*.

Применяется для доступа к структурам данных переменной длины. Тогда базовый адрес, определяющий начало набора элементов, хранится в базовом регистре, а смещение в команде определяет расстояние до определённого элемента.

Этот режим адресации удобно использовать для записей – структур данных, содержащих поля различной длины и, возможно, различных типов.

В записях с полями различной длины содержимое адресуемого регистра соответствует началу записи, а смещение в команде – расстоянию в записи (номеру ячейки относительно начала записи, с которой начинается требуемое поле записи).

$$EA = \begin{Bmatrix} BP \\ BX \end{Bmatrix} + disp.$$

Вычисление смещения операнда при базовом режиме адресации

Способы адресации данных

```
worker   struc      ;информация о сотруднике
 nam       db       30          dup       (' ')       ;фамилия, имя, отчество
 position  db       30 dup (' ')                      ;должность
 age       db       2 dup(‘ ’)                        ;возраст
 standing           db         2 dup(‘ ’)             ;стаж
 salary    db       5 dup(‘ ’)                        ;оклад в рублях
worker   ends
d_s segment
 ;описание одного сотрудника
 sotr1 worker <‘Иванов Пётр Сергеевич’,   ‘программист’, ‘30’, ‘8’, ‘15000’>
d_s ends
c_s segment
assume ds:d_s, cs:c_s
 begin:
  …
  ;загружаем в bx адрес начала записи (базовый адрес)
  lea bx, sotr1
  ;в ax – значение по адресу bx+смещение (адрес ячейки) по полю age
  ;т.е., от начала записи находим ячейки, содержащие информацию о возрасте
  mov ax, word ptr [bx].age
   …
c_s ends
end begin
```

Чтобы определить возраст сотрудника, необходимо знать адрес начала записи в сегменте данных (*sotr1*) и номер ячейки в записи, с которого начинается информация о возрасте (*age*).

Способы адресации данных

6. *Базово-индексный*.

Используется для доступа к элементам массива, адресуемого указателем. Базовый адрес массива задаётся указателем базы (базовым регистром), а номер элемента массива – содержимым индексного регистра.

В отличие от индексной адресации, где начальный адрес массива задаётся прямо в команде в виде смещения, в базово-индексном режиме начальный адрес массива предварительно загружается в один из базовых регистров

Как и при косвенной адресации, такой режим адресации данных удобно использовать при работе со сложными структурами, когда по неизменному адресу ячеек изменяется их содержимое.

$$EA = \begin{Bmatrix} BP \\ BX \end{Bmatrix} + \begin{Bmatrix} SI \\ DI \end{Bmatrix}.$$

Вычисление смещения операнда

при базово-индексном режиме адресации

Способы адресации данных

```
d_s segment
 mas db 3,5,1,8,9,'$'; По адресу mas определена последовательность
                                                ; из 6 однобайтовых элементов
                                                ; (с учётом символа $)
                                                ; $ - признак конца последовательности
d_s ends
c_s segment
assume ds:d_s, cs:c_s
begin:
…
 mov si,3 ;в si – смещение элемента (8) относительно начала
                          ; массива, т.е., адреса mas
lea bx,mas; в регистр bx загружается адрес начала массива mas
mov ah, bx[si] ; bx- смещение (адрес) массива в сегменте данных
                ; в ah – значение элемента массива mas
                ; со смещением в si, т.е. 8
…
c_s ends
end begin
```

Способы адресации данных

7. *Относительный базовый индексный*.

Используется для адресации элементов в указываемом массиве записей. Базовый адрес массива задаётся указателем базы, номер записи (т.е., элемента массива) определяется содержимым индексного регистра, а смещение в команде указывает расстояние до записи.

Таким образом, чтобы получить доступ к конкретному полю массива записей, сначала необходимо определить начало массива, в нём найти нужную запись, а уже в ней – требуемое поле.

$$EA = \begin{Bmatrix} BP \\ BX \end{Bmatrix} + \begin{Bmatrix} SI \\ DI \end{Bmatrix} + disp.$$

Вычисление смещения операнда

при относительном базовом индексном режиме адресации

Способы адресации данных

```
…
d_s segment
 ;опишем массив из 5 сотрудников со значениями по
 ;умолчанию
 mas_sotr worker 5 dup (<>)
d_s ends
c_s segment
 assume ds:d_s, cs:c_s
 begin:
 …
 ;в bx – адрес начала
 ;массива сотрудников
 lea bx, mas_sotr
 ;в si – смещение второй
 ;(начиная с нуля) записи
 mov si, (type worker)*2
 ; в ax – стаж второго сотрудника
 mov ax,[bx][si].standing
 …
c_s ends
end begin
```

Чтобы узнать стаж работы третьего сотрудника, сначала нужно в базовый регистр загрузить адрес начала массива (в примере *mas_sotr*), затем определить смещение (номер ячейки) в массиве, с которого начинается запись о третьем сотруднике (в примере – это вторая запись, поскольку индексация начинается с нуля), и записать его в индексный регистр.

Для определения номера нужной ячейки требуется размер одной записи (*woker*) в байтах (директива *type*) умножить на индекс записи в массиве.

И, наконец, в найденной записи следует найти нужное поле (в примере *standing*).

Способы адресации команд

В зависимости от того, в каком сегменте кода находится требуемая команда и явно или нет указывается её адрес, выделяют следующие режимы адресации команд:

1. *Внутрисегментный прямой.*

Команда, к которой осуществляется переход, находится в том же сегменте кода, что и текущая команда перехода.

Эффективный адрес перехода (смещение команды в сегменте кода) вычисляется как сумма текущего содержимого указателя команд IP и 8- или 16- битного относительного смещения. Данный режим допустим в условных и безусловных переходах.

```
…
cmp ah, al; Сравнивается содержимое регистров ah и al
jne met; Если содержимое регистров не равно,
                    ; выполняется переход на команду с адресом met
inc al; Увеличение содержимого регистра al на 1
met: inc ah; Увеличение содержимого регистра ah на 1
…
```

Если содержимое регистров *ah* и *al* не равно (команда *jne*), то осуществляется переход к команде с меткой *met* путём добавления к текущему содержимому регистра *IP* длины пропускаемой команды (*inc al*).

Способы адресации команд

2. *Межсегментный прямой*.

Команда, к которой осуществляется переход, находится в другом сегменте кода по отношению к текущей команде перехода, т.е. при выполнении перехода изменяется содержимое регистра *CS* и регистра *IP*.

В команде указывается пара: сегмент и смещение. Начальный адрес нового сегмента кода загружается в сегментный регистр CS, а смещение – в регистр IP. Данный режим допустим только в командах безусловного перехода.

```
c_s1 segment
assume cs:c_s1
m1: mov ah,4
add ah,7
jmp c_s:m2
c_s1 ends
```

```
c_s segment
assume cs:c_s
begin:
jmp c_s1:m1
m2: mov ah,4ch
int 21h
c_s ends
end begin
```

Сегмент кода *c_s* является главным (в нём содержится начало и конец программы) и с него начинается выполнение программы.
Адрес перехода, указанный в выделенных командах, является четырёхбайтовым (два старшие байта указывают начальный адрес сегмента, два младшие байта – смещение команды в этом сегменте).
При выполнении первого перехода в регистр *CS* записывается начальный адрес сегмента *c_s1*, а в регистр *IP* – адрес команды в этом сегменте кода (*m1*). Второй межсегментный переход выполняется аналогично.

Способы адресации команд

3. Внутрисегментный косвенный.

В этом случае двухбайтовый адрес перехода размещается в ячейках памяти по некоторому адресу (смещению) в сегменте данных. В команде перехода это смещение указывается в регистре процессора или ячейке памяти с помощью любого режима адресации данных, кроме непосредственного.

Содержимое указателя команд *IP* заменяется соответствующим содержимым регистра или ячейки памяти.

Данный способ допустим только в командах безусловного перехода.

```
d_s segment
 adr dw met; По адресу adr указан адрес
                ; команды, на которую должен
                ; быть выполнен переход
d_s ends
c_s segment
assume ds:d_s, cs:c_s
begin:
mov ax, d_s
mov ds, ax
add ah, al; Складывается содержимое
             ; регистров ah и al
jmp adr; Выполняется переход на команду,
           ; адрес которой указан по смещению
           ; adr в сегменте данных
inc al; Увеличение содержимого
       ; регистра al на 1
met: inc ah; Увеличение содержимого
              ; регистра ah на 1
mov ah, 4ch
int 21h
c_s ens
end begin
```

В команде безусловного перехода используется прямой режим адресации данных для указания местоположения адреса перехода, т.е. адрес адреса.

Способы адресации команд

4. Межсегментный косвенный.

В этом режиме четырёхбайтовый адрес перехода размещается в смежных ячейках памяти по некоторому адресу в сегменте данных. В команде перехода это смещение указывается в регистре процессора или ячейке памяти с помощью любого режима адресации данных, кроме непосредственного и регистрового.

Содержимое регистров *CS* и *IP* заменяется содержимым двух смежных слов памяти. Младшее слово загружается в регистр *IP*, старшее – в регистр *CS*.

Данный режим допустим только в командах безусловного перехода.

```
d_s segment
a dd c_s1:m1
d_s ends
c_s1 segment
assume cs:c_s1
m1: mov ah,4
add ah,7
jmp c_s:m2
c_s1 ends
c_s segment
assume ds:d_s, cs:c_s
begin:
lea bp, a
jmp dword ptr [bp]
m2: mov ah,4ch
int 21h
c_s ends
end begin
```

Перед выполнением команды перехода в регистр *BP* загружается адрес ячейки *a*, по которому хранится адрес перехода в сегмент *c_s1* на команду с адресом *m1*. В команде перехода с помощью регистра *BP* указывается местонахождение адреса перехода в виде двойного машинного слова с использованием косвенной регистровой адресации данных.

Способы адресации команд

Разрядность внутренних регистров процессора, а также внешних шин адресов и данных – одна из важных характеристик процессора.

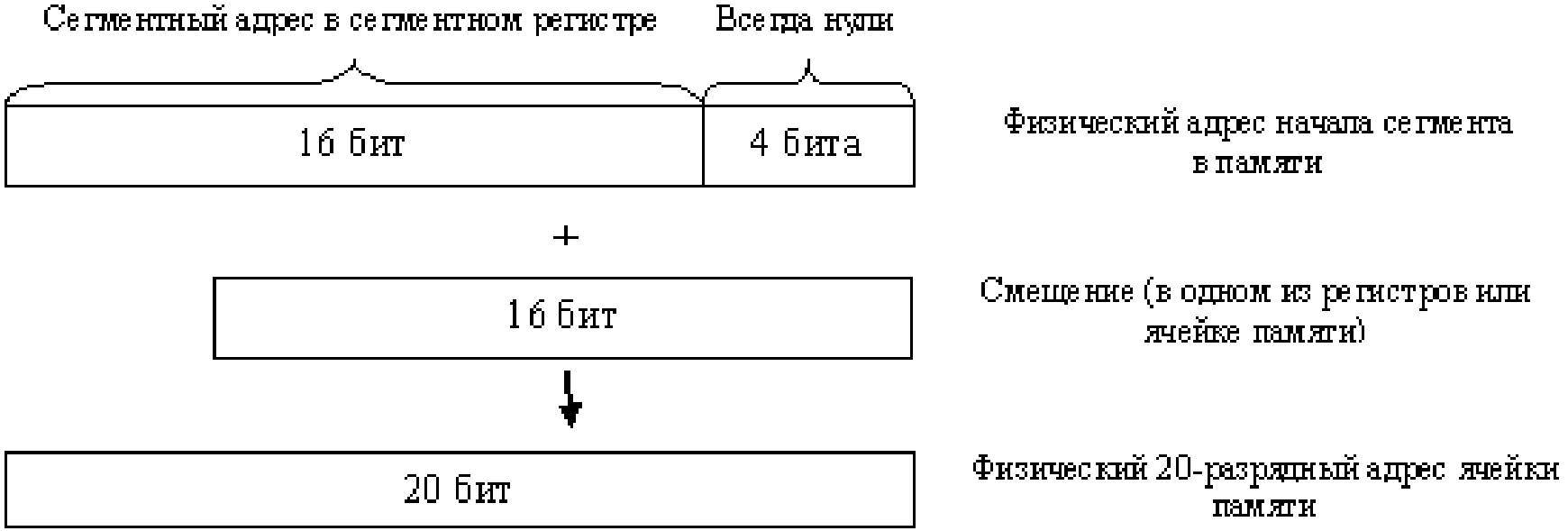
Адресная шина процессора Intel 8086 содержит 20 линий, что соответствует адресному пространству Мбайт. Для получения 20-разрядного физического адреса ячейки памяти требуется сложить начальный адрес сегмента памяти, в котором располагается эта ячейка, и смещение этой ячейки относительно начала сегмента (см. рис. 41).

Адрес начала сегмента всегда начинается с параграфа, т.е. выбирается таким образом, чтобы 20-разрядный адрес ячейки был кратен 16 (содержал четыре последних нуля).

Это позволяет сегментный адрес без четырёх младших битов (т.е., делённый на 16) хранить в одном из сегментных двухбайтовых регистров (*SS*, *DS*, *CS*, *ES*).

Способы адресации команд

При вычислении физического адреса процессор умножает содержимое сегментного регистра на 16 и добавляет к полученному 20-разрядному адресу двухбайтовое смещение.



Формирование физического адреса ячейки памяти в процессоре Intel 8086

Способы адресации команд

Современные 32-разрядные процессоры имеют 32-разрядную адресную шину, что соответствует адресному пространству 2^{32} = 4 Гбайта.

Однако описанный ранее способ формирования физического адреса не позволяет выйти за пределы 1 Мбайта.

Для преодоления этого ограничения в 32-разрядных процессорах используются два режима работы: *реальный* и *защищённый*.

В *реальном режиме* процессор функционирует фактически также, как Intel 8086 с повышенным быстродействием и может обращаться только к 1 Мбайту адресного пространства. Оставшаяся память, даже если она установлена на компьютере, использоваться не может.

В *защищённом режиме* также используются сегменты и смещения, но физические начальные адреса сегментов извлекаются из таблиц сегментных дескрипторов, а сегментные регистры хранят указатели (адреса) на строки в этих таблицах (дескрипторы). Каждый сегментный дескриптор занимает 8 байт, из которых 4 байта (32 бита) отводятся под сегментный адрес. Такой механизм позволяет обеспечить полное использование 32-разрядного адресного пространства.

Способы адресации команд

В 64-разрядных процессорах под физический адрес отводится 40, 44, 48, 64 разряда. Таким образом, объём адресного пространства в 64-разрядных микропроцессорах может составлять от 1Тбайта (1Терабайт – 2^{40} байт) до нескольких Эбайтов (1 эксабайт = 2^{60} байт).

Применяется сегментная организация памяти и может использоваться сегментно-страничная организация памяти (о логической организации памяти и её видах подробнее будет рассказано в следующих разделах).

Формирование физического адреса также выполняется с использованием дескрипторных таблиц .

Поток управления и механизм прерываний

Поток управления – это последовательность, в которой команды выполняются динамически (во время выполнения программы).

Большинство команд не меняют поток управления: после выполнения одной команды выполняется команда, расположенная следом за ней в памяти. Счётчик команд (*IP*) после выполнения каждой команды увеличивается на число, соответствующее длине команды.

Изменение потока управления происходит при наличии команд переходов (условных и безусловного), вызова процедур, сопрограмм, а также при возникновении исключений и прерываний.

1. *Команды перехода*.

При выполнении команд перехода в счётчик команд IP принудительно записывается новое значение – новый адрес в памяти, начиная с которого будут выполняться команды.

- Команда *безусловного перехода* обеспечивает переход по заданному адресу без проверки каких-либо условий. Например,

jmp met,

означает переход к команде, которая начинается с адреса *met* в тексте программы. При этом все предшествующие ей команды пропускаются.

Поток управления и механизм прерываний

- *Ветвление* (условный переход) происходит только при соблюдении определённого условия, в противном случае выполняется следующая по порядку команда программы.

Условием, на основании которого осуществляется переход, чаще всего выступают признаки результата выполнения предшествующей арифметической или логической команды. Каждый из признаков фиксируется в своём разряде регистра флагов PSW.

Возможен и такой подход, когда решение о переходе принимается в зависимости от состояния одного из регистров общего назначения, куда предварительно помещается результат операции сравнения.

Поток управления и механизм прерываний

Рассмотрим примеры:

```
sub ah,al
jz m1
add ah,3
jmp m2
m1: add al,2
m2: mov ah,4
```

```
cmp ah,al
je m1
add ah,3
jmp m2
m1: add al,2
m2: mov ah,4
```

Левый фрагмент иллюстрирует проверку содержимого регистров *AH* и *AL* на равенство. При этом используются флаги, в частности, флаг нуля *ZF*. Предварительно выполняется вычитание содержимого регистров: если их значения равны, то в результате образуется ноль и изменяется значение флага *ZF*. Команда *jz* проверяет условие: если флаг *ZF* равен 1, то выполняется переход на команду с адресом *m1*, иначе выполняется команда сложения, следующая за командой условного перехода. Команда с адресом *m2* выполняется в любом случае.

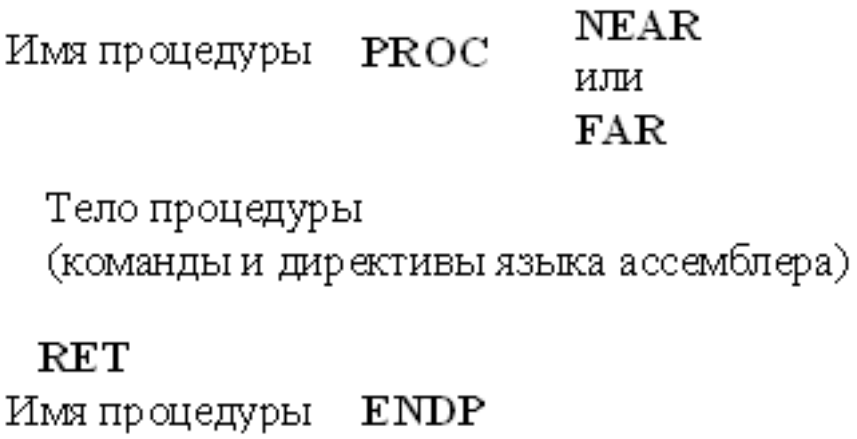
Правый фрагмент выполняет ту же проверку, но с использованием команды сравнения *cmp* и команды перехода по равенству содержимого регистров *ah* и *al* (*je m1*).

Поток управления и механизм прерываний

2. *Процедуры*.

Важным способом структурирования программ является процедура. Она может быть вызвана в любой точке программы. Но, в отличие от команд перехода, после выполнения процедуры управление возвращается к команде, следующей за командой вызова процедуры.

Процедурный механизм базируется на командах вызова процедуры, обеспечивающих переход из текущей точки программы к начальной команде процедуры, и командах возврата из процедуры для возврата в точку (на команду), непосредственно расположенную за командой вызова.



Общее описание процедуры

Поток управления и механизм прерываний

Вызов процедуры осуществляется командой *CALL*, за которой следует имя процедуры.

Для работы с процедурами используется стек (дополнительная память, организованная в виде очереди), в который команда вызова помещает текущее значение счётчика команд (*IP*) при внутрисегментных переходах (или значения регистров *IP* и *CS* при межсегментных переходах) – адрес точки возврата. При выходе из процедуры старые значения соответствующих регистров восстанавливаются из стека.

Особый интерес представляет ***рекурсивная процедура*** – процедура, которая вызывает сама себя либо непосредственно, либо через цепочку других процедур. Для таких процедур также используется стек, в котором помимо адреса возврата сохраняются параметры и локальные переменные для каждого вызова.

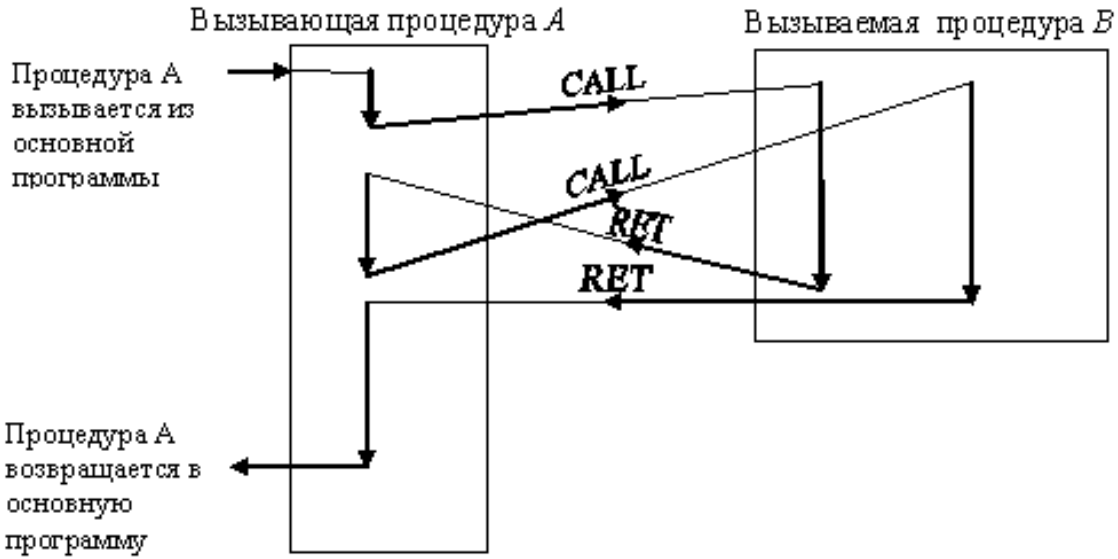
Обычно, если в программе используются процедуры, то в ней описывается сегмент стека для резервирования ячеек под дополнительную память. Если сегмент стека в программе отсутствует, то в этом случае операционная система формирует стек самостоятельно.

Поток управления и механизм прерываний

```
s_s segment stack “stack”
dw 12 dup(?)
s_s ends
d_s segment
 aa dw 10
d_s ends
c_s segment
 assume ss:s_s,ds:d_s,cs:c_s
begin:
 mov ax,d_s
 mov ds,ax
 call pr1              ;вызов подпрограммы
 mov ah,4ch            ; точка возврата
 int 21h
pr1 proc near    ;начало подпрограммы (ближний вызов)
 push ax         ;записать в стек содержимое регистра ax
 mov ax, aa
 pop ax          ;выбрать из стека содержимое регистра ax
 ret             ;команда возврата на следующую команду после
                 ;вызова процедуры
pr1 endp         ;конец подпрограммы
c_s ends
end begin
```

При выполнении вызова процедуры *pr1* в стек помещается адрес возврата – значение счётчика команд *IP*, содержащего на данный момент адрес команды, которая должна будет выполняться после текущей (*mov ah,4ch*). Значение регистра *IP* замещается новым значением – адресом первой команды процедуры. При достижении команды возврата из процедуры (*ret*) из стека в регистр *IP* записывается старое значение, что обеспечивает возврат в основную программу на команду, которая непосредственно следует за командой вызова процедуры.

Поток управления и механизм прерываний



Взаимодействие вызывающей и вызываемой процедур

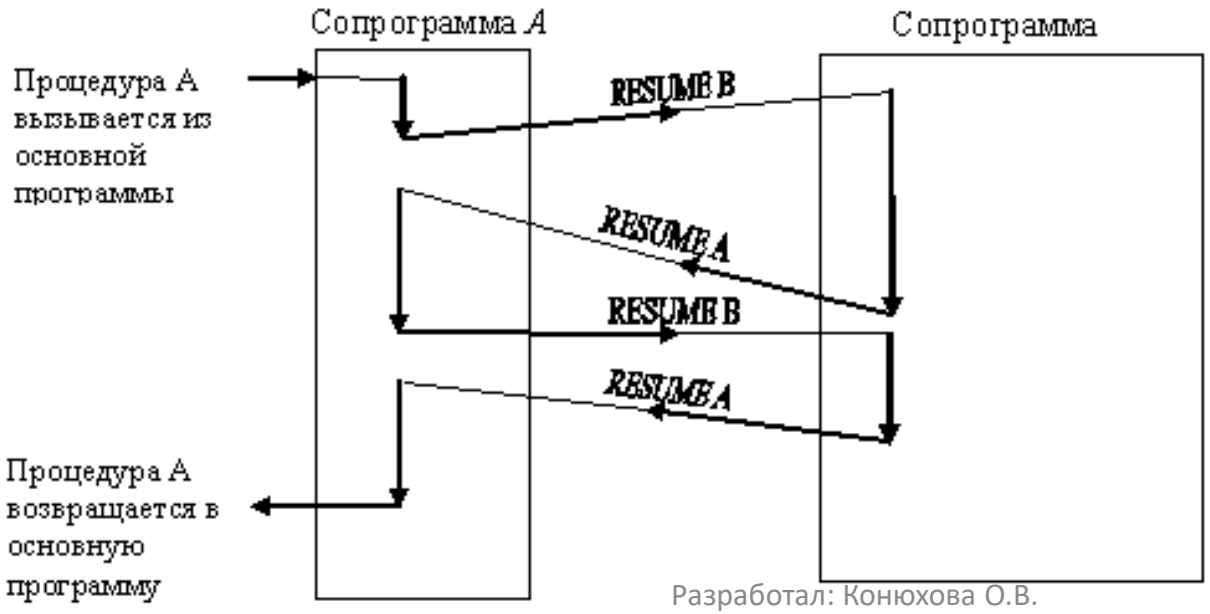
В обычной последовательности вызовов существует чёткое различие между вызываемой и вызывающей процедурами. Вызываемая процедура каждый раз начинается сначала, сколько бы раз не происходило обращение к ней. Для выхода из вызываемой процедуры используется команда возврата *RET*.

Для обеспечения такого механизма взаимодействия вызывающей и вызываемой процедур достаточно сохранять в стеке только адрес возврата в вызывающую процедуру.

Поток управления и механизм прерываний

3. *Сопрограммы*.

Пусть имеются две процедуры: *A* и *B*, каждая из которых вызывает другую в качестве процедуры. При возврате из *B* к *A* процедура *B* совершает переход к тому оператору, следующему за командой вызова процедуры *B*. Когда процедура *A* передаёт управление процедуре *B*, она возвращается не к самому началу *B* (за исключением первого раза), а к тому месту, где произошёл предыдущий вызов A. Процедуры, работающие подобным образом, называются ***сопрограммами***.



Взаимодействие сопрограмм

Поток управления и механизм прерываний

Обычно сопрограммы используются для того, чтобы производить параллельную обработку данных на одном процессоре.

Обычные команды *CALL* и *RET* не подходят для вызова сопрограмм, поскольку адрес для перехода берётся из стека, как и при возврате, но, в отличие от возврата, при вызове сопрограммы адрес возврата помещается в определённом месте, чтобы в последующем к нему вернуться.

Для этого сначала необходимо:

1. вытолкнуть старый адрес возврата из стека и поместить его во внутренний регистр;
2. поместить содержимое счётчика команд *IP* в стек;
3. скопировать содержимое внутреннего регистра в счётчик команд *IP*.

Поскольку одно слово выталкивается из стека, а другое помещается в него, состояние указателя стека не меняется.

Поток управления и механизм прерываний

4. *Исключения и прерывания* обеспечивают принудительную передачу управления специальной процедуре – ***обработчику системных прерываний***, который выполняет определённые действия.

После завершения действий обработчик прерывания возвращает управление прерванной программе. Она должна продолжить выполнение прерванного процесса в том же самом состоянии, в котором находилась в момент прерывания.

Обработчик прерываний отличается от обычной процедуры тем, что вместо связывания с конкретной программой, он размещается в фиксированной области памяти.

Поток управления и механизм прерываний

Исключение – это особый тип вызова процедуры, который происходит при определённом условии.

Наиболее распространенными условиями, которые могут вызвать исключения, являются переполнение и исчезновение значащих разрядов при операциях с плавающей точкой, а также переполнение при операциях с целыми числами, нарушение защиты, неопределяемый код операции, переполнение стека, попытка запустить несуществующее УВВ, попытка вызвать слово из ячейки с нечётным адресом, деление на ноль.

Если результат находится в пределах допустимого, исключение не возникает. Важно отметить, что этот вид прерывания вызывается каким-то исключительным условием, вызванным самой программой и обнаруженным аппаратным обеспечением или микропрограммой.

Исключения, в свою очередь, подразделяются на *сбои*, *ловушки* и *аварийные завершения*.

Поток управления и механизм прерываний

Сбой – это исключение, сообщение о котором выдаётся на границе команды, предшествующей команде, вызвавшей это исключение. После сообщения о сбое состояние ВМ восстанавливается в ситуацию, когда можно выполнить рестарт команды. Примером сбоя может служить обращение к несуществующему УВВ.

Ловушка – это исключение, сообщение о котором выдаётся на границе команды, непосредственно расположенной после команды, для которой было обнаружено данное исключение. Например, переполнение при обработке целых чисел.

Аварийное завершение – это исключение, не позволяющее ни точно определить команду, вызвавшую его, ни перезапустить программу, в которой произошло данное исключение. Аварийные завершения используются для сообщения о серьёзных ошибках: аппаратных ошибках, противоречивых или недопустимых значениях в системных таблицах.

Поток управления и механизм прерываний

Прерывания - это изменения в потоке управления, вызванные не самой программой, а какими-то асинхронными событиями, и связанные обычно с процессом ввода – вывода.

Прерывания дают возможность осуществлять операции ввода – вывода независимо от процессора. Поскольку быстродействие микропроцессора выше, чем УВВ, то процессор имеет возможность выполнять другие программы или осуществлять другие функции вместо постоянного контроля состояния присоединённых к нему периферийных устройств.

Когда УВВ требует обслуживания со стороны процессора, оно сообщает ему об этом путём формирования соответствующего запроса (сигнала), по которому процессором может быть прервано выполнение текущей программы. Управление прерываниями от УВВ осуществляется контроллером прерываний, который подключён к процессору и структурно входит в его состав.

Поток управления и механизм прерываний

Различие между исключениями и прерываниями заключается в том, что исключения синхронны по отношению к программе, а прерывания асинхронны.

Исключения вызываются программой непосредственно, а прерывания – опосредованно.

Если многократно перезапускать программу с одними и теми же входными данными, то исключения будут возникать всякий раз в одних и тех же местах программы, а прерывания – нет.

Поток управления и механизм прерываний

Наличие сигнала прерывания не обязательно должно вызывать прерывание исполняющейся программы, процессор может иметь систему защиты от прерываний: отключение системы прерываний, запрет или маскирование отдельных сигналов прерываний.

Программное управление этими средствами позволяет операционной системе регулировать обработку сигналов прерываний. Процессор может либо обрабатывать прерывания сразу по их приходу, либо откладывать их обработку на некоторое время, либо полностью игнорировать.

Например, если установлен в единицу флажок трассировки *TF*, то процессор выполняет одну команду программы, а затем генерирует прерывание типа 1, т.е., программа выполняется по шагам.

Если сброшен флаг прерываний *IF*, то процессор не реагирует на внешние прерывания (за исключением немаскируемых). Для маскирования отдельных прерываний используется регистр масок. Управляется командами *CLI* (запретить прерывания) и *STI* (разрешить прерывания).

Поток управления и механизм прерываний

С каждым отдельным типом прерывания или исключением связан идентифицирующий его номер в диапазоне от 0 до 255 и соответствующий обработчик.

Исключениям и немаскируемым прерываниям присвоены номера из интервала от 0 до 31, а маскируемым прерываниям – от 32 до 255. Не все из этих значений используются процессорами в настоящее время; неназначенные номера зарезервированы для использования в будущем.

Номера прерываний, исключений и адреса (векторы) соответствующих обработчиков хранятся в специальной таблице – ***таблице векторов прерываний***, расположенной в памяти. При возникновении прерывания или исключения по его номеру в таблице векторов прерываний определяется адрес соответствующей процедуры обработки прерывания или исключения, к которой осуществляется переход. Рассмотрим более подробно, каким образом выполняется вызов и возврат из обработчика прерываний или исключений.

Поток управления и механизм прерываний

Механизм обработки прерываний включает в себя следующие этапы:

1. Установление факта прерывания.

2. Запоминание в стеке состояния прерванного процесса, которое определяется содержимым регистра флагов *PSW*, счётчика команд *IP*, сегментного регистра *CS*. При необходимости, также запоминается содержимое регистров, которые будут использоваться процедурой прерывания и, следовательно, изменяться. Некоторые типы исключений и прерываний также помещают в стек код ошибки для того, чтобы диагностировать причину, вызвавшую исключение.

3. Определение адреса процедуры обработки прерывания по номеру прерывания в таблице векторов прерываний и осуществление перехода к этому обработчику путём загрузки адреса в регистры *CS* и *IP*.

4. Обработка прерывания. Процедура обработки прерывания выполняет свои команды.

5. Восстановление состояния прерванной программы. После успешного выполнения процедуры обработки прерывания при достижении команды *IRET* (этой командой завершаются обработчики прерываний) из стека восстанавливается старое содержимое сохраненных в нём регистров (старое состояние), в т.ч., и ***адрес возврата*** – значения регистров *CS* и *I*P.

6. Возврат в прерванную программу. На основании адреса возврата осуществляется переход к прерванному процессу. Возврат должен осуществляться на команду, следующую за командой, выполненной до возникновения прерывания. Процедура обработки прерываний, обладающая таким свойством, называется ***прозрачной***.

Поток управления и механизм прерываний

При возникновении сбоя адрес возврата является адресом команды, вызвавшей сбой, поэтому возврат осуществляется снова к этой команде для попытки её повторного выполнения (рестарта). Обработка ловушек аналогична обработке прерываний. При аварийных завершениях вычислительный процесс завершается без возможности восстановления исходного состояния программы, в которой произошло данное исключение.

Поскольку сигналы прерываний возникают в произвольные моменты времени, то на момент прерывания может существовать несколько сигналов прерываний, которые могут быть обработаны только последовательно. Для этого прерываниям присваиваются приоритеты. Существуют две дисциплины обслуживания приоритетных прерываний:

1) прерывание с более высоким приоритетом может прервать обработку прерывания с более низким приоритетом;

2) прерывание с более низким приоритетом обслуживается до конца, после чего обрабатывается прерывание с более высоким приоритетом.