Архитектура ЭВМ

19 августа 2014 г.

1 Введение

Под архитектурой ЭВМ будем понимать основные принципы устройства компьютера: его структуру, взаимосвязи между отдельными эллементами, а так-же команды. Функциональную схему ПК можно представить:

ОП ЦП МОНИТОР КЛАВА

⇓

Системная шина

⇓

мышь дисковод и т.п

Система шина(магистраль)-путь по которому передается информация от источника к любому приемнику. Функционально в системной магистрали можно выделить 3 эллемента.

1. Шина адреса, по каторой передаються адреса оперативной памити
2. Шина данных, по которой передаються данные
3. Управляющая шина, по которой передаються команды управления

2 Оперативная память

Минимальная структурная единица ОП — это бит (bi*nary digi*t). Обычно обрабатывается не один бит, а последовательность, которую называют байт. Логически байт содержит 8 битов, а физически он содержит 9 битов. Девятый бит - это контроль на четность. Требования контроля на четность заключаеться в том, что количество еденичных битов в байте всегда должно быть не четно. Байт это минимально адресуемая еденица памяти, то есть каждый байт имеет свой адресс или номер. (7| | | | | | |0) - тип байт. Компы с архитектурой intel используют принцип адресности. Тоесть для доступа к любой информации, которая храниться в некотором байте необходимо указать адрес этого байта. ОП является энергозависимым устройством для хранения команд и данных. В памяти храняться не только данные, но и команды — это принцип програмного управления.

Определение. Поле ОП это совокупность байтов с последовательными адресами.

Выделяют поля фиксированной длины:

- Word 2 байта,
- DoubleWord 4 байта.

Сегмент поля памяти, адрес которого кратен 16. Максимальная длина сегмента 64кб. Это в частности означает, что адрес любого байта внутри сегмента можно задать 16-разрядным двоичным числом относительно начала этого сегмента, такие относительные адреса называют смешение. Таким образом адрес любого байта можно предаставить в виде адреса сегмента(база) и смещения внутри сегмента.

3 Схема ЦП

рисунок

- ОУ — опреационное устройство. Выполняет основную обработку данных.
- ШИ — шинный интерфейс. Обеспечивает ОУ данными и командами.
- АЛУ — арифметико-логическое устройство. Основной эллемент выполняющий преобразования данных.
- УУ — устройство управления. Управляет работой ЦП и остальных элементов компьютера.

3.1 Регистры

В состав ЦП входит локальная память, называемая так-же регистрвой. Это восемь 16-разрядных регистров общего назначения (РОН). Они предназначены для хранения данных, которые часто использует некоторая программа. Основным свойством локальной памяти являеться высокая скорость доступа к хранящийся к ней информации в сравнении со скоростью доступа к ОП. Особенность регистров общего назначения состоит в том, что возможна адресация как всего регистра, так и однобайтовой части.

3.2 Регистры общего назначения. РОН.

Основное использование регистров общего назначения, состоит в следующем:

- AX — основной сумматор и применяется для всех операций ввода вывода, некоторых операций над строками или некоторых арифметических операций. Некоторые команды генерируют более эффективный код если они имеют ссылки на регистр AX.
- BX — базовый регистр. Это единственный регистр общего назначения, который может использоваться в качестве “индекса” для расширенной адресации, используется так-же при вычислениях.
- CX — счетчик. Управляет количеством повторений циклов и числом битов для сдвигов. Так-же используется при вычислениях.

- DX — регистр данных. Применяется для некоторых операций ввода-вывода и для операций умножения и деления над большыми числами, которые используют регистровую пару (DX,AX).
- SP and BP — обеспечивают системе доступ к данным в сегменте стека. Реже они используются для операции сложения и вычитания.
- SP — Stack Pointer. Указатель стека позволяет временно хранить адреса и иногда данные. Связан с регистром SS для адресации стека.
- BP — Base Pointer. Указатель базы облегчает доступ к параметрам, данным и адресам передаваемым через stack.
- SI and DI — индексные регистры, используются для расширенной адресации.
- SI — индекс источника. Связан с регистром DS.
- DI — индекс назначения. Связан с регистром ES.

Замечание. Любые РОН могут использоваться для сложения и вычитания, как 8 так и 16 битных чисел.

3.3 Регистр командного указателя. IP.

IP - содержит смещение на команду, которая должна быть выполнена следующей.

4 Сегментные регистры

Каждый сегментный регистр обеспечивает адресацию в 64кб, такая область памати называется текущим сегментом, его адрес выровнен на границу параграфа и обязательно кратен 16. Регистр сегмента кода **CS,** содержит начальный адрес сегмента кода. Этот адрес + велечина смещения в командном указателе **IP** определяет адрес команды, которая будет выполняться следующая. Для обычной программы ссылка на **CS** не обязатьльна. Регистр сегмента данных **DS** содержит начальный адрес сегмента данных. Доступ к данным происходит относительно начала сегмента. Регистр сегмента стека **SS** содержит начальный адрес сегмента стека. Регистр **ES** дополнительный сегментный регистр для управления адресацией памяти, при выполнении некоторых операций со строками.

Регистр флагов:

				O	*D*	*I*	*T*	*S*	*Z*		*A*		*P*		*C*
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Девять из 16 флагового регистра являеються активными и определяют текущие состояние ПК и результат выполнения, многие арифметические команды и команды сравнения изменяют состояние флагов. Флаги:

- Флаги условий:

    1. **Carry-**содержит перенос из старшего бита, после арифметических операций, а так-же последний бит при сдвигах.

2. **Overflow-**указывает на переполнение старшего бита в арифметических операциях
3. **Sign-**содержит результирующий знак, после арфиметических операций. 0-"="; 1-"-"
4. **Zero-**Показывает результат арифметических операций и операций сравнения. 0-"≠". 1-"="
5. **A-**Дополнительный перенос. Содержит перенос из 3 бита для 8 битных данных.
6. **Parity-**контроль четности, показвыает четность младших 8 битовых данных. 1-четно, 0-нечетно

- Флаги состояния (устанавливаються программно):

1. **Derection-**указывает направление пересылки или сравнение строковых данных. 0-от младших к старшим битам. 1-наоборот.
2. **Interrapt-**прерывание. 1-указывает на возможность внешних прерываний. Прерывание - это специальная опаратная функция, выполнение которой заключаеться в передаче определенного сигнала процессору, после ЦП приостанавливает выполнение текущей программы и передает управление, некоторой другой программе называемой, обработчиком прерываний.
3. **Trap-**трассировка, обеспечивает возможность работу процессора в пошаговом режиме.

5 Внутреннее представление данных

Система команд ПК поддерживает работу с числами только размера байт и word, и частично doubleword. Более подрбоная информация в школьном курсе, который я обязуюсь повторить. В памяти байт полей, храняться в "обратном" порядке: -11 FFF5. Хранится будет как FF | F5. В регистрах храниться в нормальном виде.

5.1 О вещественных числах

В ПК, нет команд реализующих опреации над вещественными числами. Это связано с тем, что аппартаная реализация этих опреаций достаточно дорога. Возможно два решения данной проблемы.

1. На основе имеющихся команд, написать собственные процедуры, для реализцаии арифметических опреаций с вещественными числами.

2. Использование арифметического сопроцессора.

5.2 Представление симфольных данных

ASCII

6 Структура команд

Машинные команды, занимают от 1 до 6 байт. Коды операции занимают 1 или 2 первых байта команды. В пк достаточно много различных операций, так что для них не хватает 256 различных кодов, поэтому некоторые операции объеденяються в группу и им дается один и тот-же код операции, а во втором байте этот код уточнается. Кроме этого во втром байте, указываються типы операндов и способы их адресации. В остальных байтов команды указываються ее операнды. Команды могут иметь, до двух операндов(может быть и 0). Размер операндов, байт или слово, реже двойное слово. Операнд может быть указан "не успел написать" (непосредственный операнд), так и в регистре или в поле памяти. Результат операции помещаеться в один из операндов. Большинство команд с двумя операндами релизует op1=op1*op2, где *-это операция заданная кодом операции.

6.1 Форматы команд

1. RR-регистр регистр. Два байта.

КОП	d	w	1	1	$r1$	$r2$
7	1	0	7	6	543	210

   w-бит словности, если w=0 операция выполняеться над байтами, если w=1 над словами. d-направление

   r1=r1*r2 if d=1

   r2=r1*r2 if d=0

   3х битовые поля r1 and r2 учавствуют в операции по следующей схеме.

reg	w=1	w=0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	DH
110	SI	CH
11	DI	BH

2. RS-регистр памяти

   2-4 байта.

КОП	dw	mod	reg	mem	adr(0-2,)

   reg:=reg*adr; d=1

   adr:=ref*adr; d=0

   reg-ругистр

   adr-адрес памяти

d-направление

2х битовое поле mod определяет количество байтов, которое занимает операнд adr

00-0б

01-1б

10-2б

mem-3бита, указывает способ модификации адреса

mem\mod	00	01	10
000	[Bx]+[SI]	[BX]+[SI]+a8	...+a16
001	[BX]+[SI]	[BX]+[DI]+a8	...+a16
010	[BP]+[SI]	[BP]+[SI]+a8	...+a16
011	[BP]+[DI]	[BP]+[DI]+a8	...+a16
100	[SI]	[SI]+a8	...+a16
101	[DI]	[DI]+a8	...+a16
110	a16	[BP]+a8	...+a16
111	[BX]	[BX]+a8	...+a16

a8-адресс задаеться в байте

a16-адресс задаеться в слове

если в комнаде не задан адрес то он являеться нулевым. Случай когда mem=110 and mod=00, говорит об отсутсвиии регистр модификаторов

mod=11 соответствует формату RR

3. RI-регистр непосредсвенный операнд

   3-4 байта

КОП	sw	11	КОП'	reg	I(1-2б)

   reg=reg*I

   КОП-определяет группу операций

   КОП'-саму поерацию

   w-оперделяет длину непосредсвенного слова

   Для экономии памяти предусмотрен случа, когда в операциях над словами непосредственный операнд может быть задан байтом. S=1 при W=1.

   При этом при выполнении команд, байт автоматически расширается до слова

4. SI-память непосредсвенный операнд

   3-6 байтов

КОП	sw	11	КОП'	adr(0-2)	I(1-2б)

   adr=adr*I

   Смысл остальных полей остаеться прежним

Часть I
Язык Ассамблера

ЯА практически представляет собой символьную форму записи команд машинного языка, тоесь вместо цифровых кодов записывают мнемонические обозначения команд, вместо адресов имена, константы можно записывать в 10 системе исчесления

7 Индентификаторы

Нужны для обозначения различных частей программы, переменных, меток и т.п. В языке ассемблера, индентификатор-это последовательность латинских букв, цифр и знаков. На последовательность накладываеються следующие ограничения: последовательность может быть любой, но значишие только 31 первый символ, как обычно идентификатор не должен начинаться с цифры, ax=AX, запрешена кирилица.

Идентификаторы деляться на служебные слова и имена. Служебные слова имеют заранее определенный смысл и могут обозначать регистры, команды, дерективы и т.п. Все остальные идентификаторы называються именами и программист волен давать красивые имена объектом свеой прогрммы

8 Целые числа

Могут быть записанны в десетичной, двоичной, восьмеричной и шестандацитиричной системе исчесления, при записи числа в других системах исчесления кроме 10, в конце ставиться спецификатор(буква).

(2) 1011(b)
(8) 1230(o)(q)
(16) 123(h)
(10)123(d)( )

Замечание. Если 16ричное число начинаеться с “буквенной” цифры, то в начале числа должен быть записан хотябы одЫн ноль.

9 Символьные данные

Заключаються либо в одинарные либо в двойные ковычки ’A’=”A”

10 Структура программы

Программа на языке ассемблера, это последовательность инструкций каждый из которых записываеться в отдельной строке. Инструкции языка ассмблера деляться на три группы:

1. Коментарии
2. Директивы

3. Команды

10.1 Коментарии

Как обычно не влияют на смысл программы и предназначены для описания особенностей алгоритма. Коментарией являеться любая строка, начинающиеся с символа “;”. Все символы после ; являються коментариями. Допускается многострочный коментарией: COMMENT <маркер><текст>. В качества маркера выбираеться первый отличный от пробела символ следующий за словом COMMENT. Концом такого коментария будет строка с символом маркера, то есть. COMMENT *вы все гавно*. Такой вид коментария обычно используеться, когда нужно временно(например при отледаке) исключить из программы некоторый код.

10.2 Директивы

Кроме машинных команд программе нужно указывать, какие константы и переменные используются, какие имена мы им дали.

[<ИМЯ>]<Директивы>[<операнды>][;<коментарий>]

X DB 10,-3,02FAh;массив из 3 эллементов

Y DB 1

x	x+1	x+2	x+3	y
10	-3	FA	02	1

10.3 Команды

Это символьная мнемоническая форма записи машинных команд.

[<Метка>:]<Команда>[<операнды>][;<коментарий>]

L1: ADD AX,2 ; увеличить AX на 2

10.4 Директивы определения данных 3 штуки

Предназначены для описания переменных исползуеммых в программе.

- DB — описание байты

  define byte — определить байт

  [<имя>] DB <операнд>{,<операнд>}

  По этой дерективе ассемблер вычесляет операнды и записывает их значения в последовательные байты памяти. Первому из этих байтов даеться указанное имя, по которому можно ссылаться на этот адрес из других мест программы, есть два основных способоа задания операнда: ?(неопределенное значение) и констатное выражение со значением (-128..255).

  – A DB ?-описывает одЫн байт расположенный по адресу A, в который ничего не записывается. В этом байте будет, то что осталось от предыдушей программы. Адрес ячейки выделенной переменной с именем A принято называть

значением имени A(не путать с содержимым, которое хранится по этому адресу). По описанию переменной ассамблер запоминает, сколько байтов занимает переменная в памяти-это размер называеться типом имени переменной. Значение(адрес) и тип(размер) имени переменной одназачно определяет ячейку обазначаемую этим именем. Оператор типа TYPE <имя>. Значение этого оператора являеться размер в байтах ячейки выделенной под переменную с указанным именем. В ассемблере есть стандартная константа с красивым именем BYTE=1. И тогда TYPE A имеет размер BYTE.

– Константное выражение. Позволяет поисывать перменные с начальными значениями. При выполнении программы эти значения могут изменяться.
  * A DB 254 ; OFEh
  * B DB -2 ; OFEh(256-2=254)
  * C DB 15h; 21(10)

  Часто в качестве начально значения указываеться символ, задать его можно двумя способами:
  * D1 DB '*'
  * D1 DB 2Ah<-код символа

– Деректива с несколькими операндами.

```
M DB 2
DB -2
DB ?
DB '*'
```

  Сдесь описан массив состоящий из 4 элементов. Допускаеться упрощенная запись: M DB 2,-2,?,'*'. По этому описанию.

M	M+1	M+2	M+3
02	FE		2A

Имя M задает адрес первого байта. Для ссылок на остальные используется конструкция M+K. Записи вида <имя>±K это говорит о том что можно прибавить или отнять указанное количество байтов.

```
X DB 1,12,3
M DB 2,-2,?,'*'
Y DB 4,5,6
[M-2]=12
[Y-6]=12
[Y-7]=1
```

– Операнд строка

```
S1 DB 'a','b','c'
S2 DB 'abc'
S3 DB 'a'
   DB 'b'
   DB 'c'
S1 S2 S3 равны
```

– Конструктор повторения DUP

Часто в дерективе приходиться указывать одинаковые операнды

```
A DB 0,0,0,0,0,0,0,0,0,0,0,0,0
A DB 13 DUP(0)
```

Это конструкция имеет вид: n DUP (p1,p2,...,pk), где n константа или константное выражение, значение которого может быть вычесленно во время компиляции. p1, p2, ... pk любые допустимые эллементы дерективы DB в томчисле и конструкция повторения.

```
B DB 3 DUP(1,2); 1,2,1,2,1,2
C DB 10 DUP(20 DUP(?))
```

Про пример выше. В дерективе выделаяеться 200 байтов, значения которых неопределены, это место можно трактовать 10 строк по 20 эллементов в каждой строке. Фактически должны быть организованна, специальная обработка этих 200 байтов

- DW — описание слова

define word — описываються переменные размером в слово

```
A DW ?
TYPE A=2, ибо WORD
```

– Константное выражение

$-32768..65535 \quad [-2^{15}; 2^{16} - 1]$

```
B DW 1234h
C DW -2
```

Как и в случае дерективы DB неотрицательные числа записываются в память как числа без знака, а отрицательные в дополнительном коде. При размещении числовых констант, ассе

мблер автоматически меняет местами значения старшего и младшего байтов.

B		C	
34	12	FE	FF

Частным случаем может быть описание строки, состоящие из двух или одного символа

```
S1 DW '01'
S2 DW '1'
```

S1		S2	
31	30	31	00

– Адресное выражение

Определение. Адресное выражение — это выражение, значением которого является адрес.

```
A DB ?
B DW A
```

В слово выделенное под переменную B записывается адрес переменной A. Значение этого адреса определяется, как велечина смещения соответствующего поля, относительно начала того сегмента, в котором это поле памяти зарезервированно. В качестве адресного выражения может быть использованно специальная переменная $ — счетчик команд.

```
A DW $
```

Запись означает что в поле A адрес поля A

```
B DB 100DUP(?)
C DW $-B;=100-длина B
```

Все что работало с DB работает и с DW

- DD — двойное слово

  define double word — описывает двойное слово

  Имена таких переменных имеют тип DWORD=4байта

  – Целое число
    $[-2^{31}, 2^{32} - 1]$

    ```
    X DD 123456h
    ```

X			
56	34	12	00

  – Константное выражение
    $[-2^{15}; 2^{16} - 1]$

    В языке ассемблера результаты всех операций беруться по моделю велечены $2^{16} = 10000h$. Поэтому получить выражение значение, которого являлось бы 32битовое(даже 17 битовое) число не удаеться

    ```
    X DD 8000h+8002h;=2
    ```

  В DD работает все как в DB и DW

10.5 Директивы эквивалетности и присваивания

Константы описываються с использованием дерективы эквивалетности EQU:
<имя>EQU<операнд>.

Здесь обязательно должно быть указаноимя и один операнд. Это деректива эквивалента этому коду в паскале Const <имя>=<операнд>.

С её помощью программист информирует ассемблер о способе интерпретации некоторого имени. Возможно 3 способа задания операнда:

1. Операнд константное выражение

```
A EQU 10
X DB A DUP(?)
Y DB A*10+1; 101
```

2. Операнд имя

```
A EQU N
```

   В этом случаее именя A and N являються синонимами

3. Операнд произвольный текст не являющийся константым выражением или именем.

   В этом случае, любое вхождение имени, ассемблер заменяет на соответствующий текст. Отметим, что деректива EQU носит чисто информационый характер. По ней ассемблер ничего не записывает в программу, поэтому дерективу EQU можно ставить в любое место программы.

   <имя>=<конст.выражение>

```
K=1
N EQU K;N И K СИНОНИМЫ
A DB N; A:=1
K=2
A DB N; A:=2
```

   Если с помощью дерективы эквивалентности можно определить не только число но и другие конструкции, то присваивание можно определить только константу.

   Появление в языке констант, которые менять свои значения вносит некоторую неопределенность

```
K=1
N EQU K
A DW N; A:=1
K=2
B DW N; B:=2
//-------------
K=1
N EQU K+10
C DW N; C=11
K=2
D DW N; D=11
```

   Введем слудюшие уточнения в действие эквивалентност. Если в правой части дерективы указанно имя константы, то имя слево не константо, а как синоним имени справа. Если в правой части указанно любое другое констатное выражение, то имя слева становиться именем константы. По дерективе присваивания, правая часть вычесляеться сразу и становиться новым значением константы

11 Команды целочисленной арифметике

1. i8, i16, i32 — непосредственные операнды, те которые задаються в самой команде. Длиной соответсвтенно 8, 16, 32 бита.
2. r8, r16 — регистры общего назначения. r8 байтовые регистры: AH, BL, ... r16 регистры размером слово: AX, SI, CX.
3. sr — сегментные регистры.
4. m8, m16, m32 — адресс памяти соответсующей длины.

11.1 Команда пересылки. MOV

Имеет синтаксис:

MOV op1, op2// op1:=op2

В качестве операндов могут задаваться: регистры, непосредственный операнд и адреса.

Например

```
MOV AX,SI
MOV BL,CH
MOV AL,BX
//-----------
A DW ?
B DW ?

    MOV A,B//ERROR
    MOV A,10
    MOV AX,A
    MOV B,AX
```

В команде MOV нельзя одновременно использовать два сегментных регистра. С помощью MOV нельзя изменить регистр CS. Кроме явного задания адресов операндов(имена регистров, имена полей памяти), можно использовать так называемую косвенную адресацию, то есть адресс некоторого поля может формироваться с учетом содержимого некоторых регистров.

```
MOV AX, A[SI]
```

Адресс второго операнда вычесляеться как сумма адреса имени A и содержимого регистра SI. [ ]-содержимое по соответствующему адресу

```
A DW 100DUP(?)
ADR DW A

    ...
    MOV SI, ADR
    ...
    MOV AX, [SI]
```

```
...
SUB SI,SI
MOV AX,A[SI]
ADD SI,2
MOV AX,A[SI]
...
```

11.2 Оператор указания типа. PTR

Может возникать ситуация, когда ассемблер не может одназначно определить размер пересылаемой велечины.

```
mov [si],0
```

По этой команде нельзя одназначно определить, что следует сформировать по адресу который храниться в si. Нулевой байт или нулевое слово. В таких случаях программист, должен явно укзать ассемблеру такую информацию с помощью оператора указания типа <тип>PTR<выражение>, где тип — это byte, word, dword, а выражение может быть константым или адресным.

```
mov byte ptr[si],0
mov word ptr[si],0
```

Оператор PTR нужно не уточнить а нужно изменить тип непосредственного операнда, например:

```
z dw 1234h
mov z, byte ptr 0
//----------------------
```

11.3 Команда обмена. XCHG

XCHG op1,op2//swap(op1,op2)

12 Команды сложения и вычитания

```
add op1, op2;op1:=op1+op2;
sub op1, op2;op1:=op1-op2;
```

Программист должен сам учитывать, над каким типом целых чисел выполняеться операции. При выполнении операции результат может не помещаться в отведенное ему место, тогда для контроля коректности полученного результата, следует анализировать биты флагового регистра: OF - знаковый, CF - без знаковый. Если рузельтат операции с безнаковыми числами непомещаеться в соответствующий байт или слово, то CF устанавливаеться значение 1. Это означает, что результат сформирован по модулю 2^16 анлогичная ситуация у знаковых чисел будет OF=1.

Термины: установить флаг, означает, что флаг принимает значение 1, сбросить флаг, флаг принимает значение 0. Кроме флагов OF and CF, команда сложения и вычитания меняет флаги ZF and SF.

12.1 INC and DEC

```
inc op1; op1++
dec op1; op1--
neg op; op:=-op
```

Есть особые случаи op byte and op:=-128 Значение после Neg не измениться и т.п случаи.

op word and word:=-32768 в этом слуаче OF=1 при других операндах 0. Если операнд равен 0, то CF=0 в остальных случаях 1, при этом обычном способом устанавилваються ZF and SF.

12.2 ADC and SBB

```
adc op1, op2;op1:=op1+op2+CF; сложение с переносом
sbb op1, op2;op1:=op1-op2-CF; вычитание с учетом заёма
```

Эти команды предназначены для моделирования сложения и вычитания длинных чисел, которые имеют формат DD.

При выполнении adc:

xx...x 1x...x

xx...x 1x...x

———

x 0x...x

CF=1

Например:

```
x dd ?
y dd ?
z dd ?; z:=x+y; Напомним что ассемблер записывает в память значения в перевернтуо

mov ax, word ptr x
add ax, word ptr y
mov bx, word ptr x+2; CF=0|1
adc bx, word ptr y+2
mov word ptr z, ax
mov word ptr z+2, bx
```

Реализцаия сложения для большых чисел. Анологично можно реализовать вычитание длинных беззнаковых чисел.

13 Команды умножения и деления

13.1 Умножение

В отличии от сложения и вычитания, умножение и деление для знаковых и беззнакомых чисел, производиться по разным алгоритмам.

```
mul op; без знака
imul op; знак
```

Положение первого операнда фиксировано, второй операнд “op” задается в команде. op-регистр или поле памяти, размер op определяет размер результата.

```
op-db

    ax:=al*op

op-dw

    (dx,ax):=ax*op
```

Под результат отводиться в два раза больше места, чем под сомножители. Умножение n-значных чисел в общем случае дает произведение из 2n-знаков. Если результат не размещается в формате операндов, то CF and OF сбрасываються в 0. Если результат действительно требует удвоенного объема памяти, то CF and OF устанавливаються в 1.

13.2 Деление

```
div op; unsigned
idiv op; signed


op-db

    ah:=ax mod op
    al:=ax div op

op-dw

    dx:=(dx,ax) mod op
    ax:=(dx,ax) div op
```

В этих командах указывается только второй операнд, делитель, который может быть регистром или поле памяти, но не может быть непосредсвеннем операндом.

Обратим внимание на случай op-dw. Обычно делимое помещаетьсяв регистр ax при этом dx=0(до div). При выполнении деления возможно появления ошибок, деление на 0 или переполнение. Переполнение бывает тогда, когда неполное частное не помещаеться в отведенное ему место.

```
mov ax, 600
mov bh, 2
div bh; будет error!!


ex 2:
z:=x div y
x db ?
y db ?
z db ?
....
mov al, x
sub ah, ah
div y
mov z, al
```

14 Изменения размера числа. CBW

Пусть требуеться выполнить:

```
ax:=ax+bl;
add ax, bl
```

В этой ситуации bl необходимо расширить до bx. Если число трактуеть как беззнаковое : bl 20h -> bx 0020h. Для чисел со знаком, нужно учитывать знак числа, если число отрицательное, то левый байт нужно заполнять FF, то есть FF20h. Можно сказать, что расширение со знаком, заключается в дублировании слева знакового вида. Команда CBW всегда выбирает al, результат записывает в ax. Флаги не изменяються.

15 Переходы, циклы

15.1 Безусловный переход. JMP

Процессор выполняет команды машинной программы в том порядке, как они записаны в памяти. Выполнение любой команды начинаеться с того, что содержимое регистра IP увеличиваеться на длину текущей команды. И таким образом в регистре адресов команд, будет адресс следующей команды. Если команда во время своего выполнения изменяет содержимое IP, то в результате за данной командой необязательно будет выполнять следуюшая. Такие команды называються команды перехода или команды передачи управления. Команда перехода не изменяет регистор флагов. Смотри пример в мануале.

.....

Встречая символьную команду перехода с меткой, ассемблер вычесляет разность между адресом этой метки и адресом самой команды. Если она не большая и помещаеться в байт, то ассемблер формирует команды короткого перехода, которая занимает дав байта. Если разность большая, то формируеться длинный переход знамающий 3 байта. Однако такой переход ассемблер может сделать, если метка была описанна до команды перехода, то есть если эта метка являеться ссылкой назад. В случае, когда в переходе указана метка “вперед” то ассемблер не может знать какой сдесь переход, короткий или длинный и поэтому на всякий случай формирует команду длинного перехода. В ассемблера есть специальный оператор short. Если указан оператор short но при этом ошиблись, компилятор нам скажет.

15.2 Косвенный переход.

Это означает, что в команде перехода, указываеться не сам адресс перехода, а то место где находиться адресс. Это может быть регистр общего назначения или слово памяти. Если jmp ax, это всегда трактуеться как косвенный переход.

```
a dw l

    jmp a; goto l
    mov dx, a
    jmp dx; goto l
```

```
l: .
```

z-некоторое имя

```
jmp z
```

если z-имя метка то проблем нет. В случае, когда z сслыка вперед, то есть это имя описывается позже, то ассемблер не будет знать какой это переход. И поэтому формируеться команда прямого\длинного перехода, если вдруг обнаруживается что z не метка, то фиксируеться ошибка. И чтобы ассемблер думаел что это слово нужно, Jmp word ptr z.

16 Команды сравнения и условного перехода.

Условный переход обычно реализуеться в два шага. Сначала сравниваються две велечины, в результаты чего соответсвуюшим образом формируються флаги и затем выполняеться собственно переход в зависимости от значения флагов. Команда сравнения cmp анологична sub. Но результат вычитания никуда не помещаеться, поэтому единственный и главный эффект команды сравнения - это установка флагов, характеризующих разность. Команд условного перехода много и все они записываються единообразно:

```
J* <label>
```

Все команды условного перехода можно разедлить на три группы

1. Входят команды, которые ставяться после команд сравнения

   В их мнемокодов с помощью опредленных букв описываеться исход сравнения

   - E-equal
   - N-not
   - G-greater
   - L-less
   - A-above
   - B-below

   Две системы обазначений для условий больше, меньше объясняються тем, что после сравнения чисел со знаком и без знака, следует реагировать на разные значения флагов. JL передает только в том случае, когда SF!=OF. Если сравниваваемые числа тратуютьяс как знаковые, то возможны две комбинации флагов соответствующих условию: op1<op2(op1-op2<0).

   - OF = 0, SF = 1

     Если при вычитании не было переполнения, то SF фиксирует настоящий знак разности и поэтому.
   - OF = 1, SF = 0

     Если произошло переполнение, то результатом будет противоположное значение, чем у настоящей разности.

Допускается сочетание этих букв.

Приведем названия всех комманд условного перехода.

all		
JE	$op1 = op2$	ZF=1
JNE	$op1 \neq op2$	ZF=0
signed		
JL/JNGE	$op1 < op2$	SF!=OF
JLE/JMG	$op1 \leq op2$	SF!=OF or ZF=1
JG/JNLE	$op1 > op2$	SF=OF and ZF=0
JGE/JNL	$op1 \geq op2$	SF=OF
unsigned		
JB/JNAE	$op1 < op2$	CF=1
JBE/JNA	$op1 \leq op2$	CF=1 or ZF=1
JA/JNBE	$op1 > op2$	CF=0 and ZF=0
JAE/JNB	$op1 \geq op2$	CF=0

2. Содержит те команды, которые реагируют на значения определенного флага.

JZ	ZF=1
JNZ	ZF=0
JS	SF=1
JNS	SF=0
JC	CF=1
JNC	CF=0
JO	OF=1
JNO	OF=0
JP	PF=1
JNP	PF=0

3. JCXZ <label>

```
if cx=0 then goto <label>
```

Отметим общю особенность команд условного перехода: все они осуществляют только короткий переход, то есть с их помощью можно передать управление не более чем на 128 байтов вперед или назад. Это примерно 30-40 команд. Для реализации длинных условных переходов вынуждены привлекать команду длинного безусловного перехода. При далекой метки M, тогда:

```
if ax==bx then goto M
//делаем так
if ax<>bx then goto L
goto M
L:...
M:....
```

Отметим, что в командах условного перехода оператор short использовать не нужно.

Пример: Вводиться некотрая последовательность чиселя, признак завершения ввода 0. Сколько чисел кратно 3.

```
DATA SEGMENT

    a  dw ?; simbol
    k  dw 0; kol
    mod    db 5
    messOut    db  "Kol-vo %$"
    messOut2   db " = $"

DATA ENDS
Ваш код ============================
l1:

    inint a
    cmp a, 0
    je exit2
    jg l3
    neg a

l3:

    mov ax, a
    div mod
    cmp ah, 0
    jne l2
    inc k

l2:

    jmp l1

exit2:

    lea dx, messOut
    outstr
    mov ah, 0
    mov al, mod
    outint ax
    lea dx, messOut2
    outstr
    outint k
```

17 Команды управления цикла

С помощью команд перехода можно реализовать любые ответвления и циклы.

```
if x>0 then s1 else s2


    cmp x, 0
    jle L2
    s1
    jmp L1
```

```
L2:
    s2
//-----------------
while x>0 do S
beg:
    cmp x, 0
    jle fin
    S
    jmp beg
fin:
//-----------
repeat S until x>0
beg:
    S
    cmp x,0
    jle beg
```

Часто на практике встречаються циклы с заранее известным количеством повторений(for).

17.1 Команда Loop

```
Loop <метка>
```

Описать выполенние этой программы можно так:

```
cx:=N
L:
    тело цикла
    dec cx
    if cx<>0 then goto L;
```

Реализовать повторений некоторой группы команд N разможно так:

```
mov cx, N
L:
    тело цикла
    loop L
```

Особенности команды Loop:

1. Счетчик цикла обязательно регистр cx
2. Начальное значение cx, должно быть присвоено до цикла

3. Так Loop ставить в конце цикла, то тело выполниться хотябы одын раз, поэтому для N=0 это схема не потходит.

4. Если возможен вариант число повторений может быть равное 0, то при N=0 нужно сделать обход цикла

```
mov cx, N
JCXZ L1
L:
    тело цикла
    loop L
L1:
```

5. Как и команда условного перехода, Loop делает только короткий переход, поэтому цело цикла не должно првышать 128 байтов.

Пример(x^y):

```
x db ? ;0..6
y db ? ;0..6

mov ax, 1
mov cl, y
mov ch, 0
mov bx, 1
jcxz L1
mov bl, x
mov bh, 0
L:
    mul bx
    loop L
L1:
    outint bx
```

Пример(найти сумму положительных чисел последовательности):

```
a dw ?
s dw 0
n dw ?

inint n
mov cx, n
L1:
    inint a
    cmp a,0
    jle L2
    mov ax, a
    add s, ax
```

```
L2: loop L1
outint s
```

17.2 Команды Loope and Loopne

Эти команды очень похожи на команду Loop, то есть cx содержит количество повторений, однако они допускают и досрочный выход из цикла

Выполнение команды Loope можно поисать так:

```
cx:=n
L:

    тело цикла
    cx:=cx-1;
    if (cx<>0) and(zf=1) then goto L; когда результат равен 0 мы переходим на L
```

Выполнение команды LoopNe можно поисать так:

```
cx:=n
L:

    тело цикла
    cx:=cx-1;когда результат равен  мы переходим на L
    if (cx<>0) and(zf=0) then goto L;
```

Перед командой Loope помещаеться команда изменяющая флаг zf(Обычно cmp). Команда Loope посторяет цикл cx раз, но только если предыдушая команда фиксирует равенство сравниваемых велечин. По какой именно причене произошел выход из цикла, по zf=0 or cx=0 нужно проверять после цикла, причем проверять следует флаг z по команде jz/je or jnz/jne. Часто команда Loope используеться для поиска первого эллемента некоторой последовательности отличного от заданной велечины.

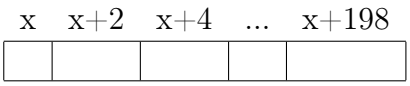
Пример

```
n db ?

inint dx
mov n, dl
mov cx, dx
sun cx, 2
mov bl, 1

dv:

    inc bl
    mov ax, dx
    div bl
    cmp ah, 0
    loopne dv
    je dv1
    mov bl, 0
```




```
dv1:

    mov bh, 0
    outint bx
```

Это не лучший авриант поиска простого числа.

18 Программирование вложенных циклов

Очевидно, что вложенные циклы типа while or repeat программируються на ассемблере без принципиальных сложностей в соответсвии с рассмотренными выше стандартными макеты. При использовани вложенных циклов с использовании команды loop необходимо перед входом во вложенный цикл позаботиться о сохранения переменной cx а после выходы из цикла востановить.

```
mov cx, n
l2:

    mov r, cx
    mov cx,m
    l1:
        ....
    loop l1

    mov cx, r
    loop l2
```

19 Циклы с переадрисацией

Циклы в теле которых используються команды с изменяющимеся исполнтелями адресами - это циклы с переадрисацией. Изменения исполнительного адреса возможно с использованием регистров модификаторов: BX,BP,SI,DI

Пусть имееться одномерный массив состоящий из 100 элементов размером слово:

```
x dw 100dup(?)
```

Доступ к эллементу массива формируеться из некоторой постоянной части x и “перменной части” — индекса.

```
mov ax, x[bx]
т.е это x+[bx]
```

Пример(получить сумму эллементов массив)

```
x dw 100dup(?)
s dw 0
....

    mov cx, 100
    mov bx, 0
    mov ax, 0

L1:

    add ax,x[bx]
    add bx,2
    loop L1

mov s, ax
```

В случае когда эллементы x байты то bx увеличиваем на 1 ибо байт

Пример(в массиве a состоящий из N эллеметов найти количесвто положительных с четными номерами)

```
a dw 100dup(?)
n dw ?
k dw 0
dva dw 2
....

    inint n
    mov cx, n
    mov si, 0

L1:

    inint a[si]
    add si,2
    loop l1

mov ax, n
div dva
mov cl, al
mov ch, 0
mov si, 2
L2:

    cmp a[si],0
    jl L3
    int k
    L3:
        add si, 4
        loop L2
```

Пример(дан массив из n эллементов, если массив не упорядочен по возрастанию своих эллементов то все эллементы кратные 3 заменить 0)

```
a dw 100dup(?)
n dw ?
f dw 0
tri dw 2
....
inint n
mov cx, n
mov si, 0
L1:

    inint a[si]
    add si,2
    loop l1

mov cx, n;проверка упорядоченности
dec cx
mov bx, 0
L2:

    mov ax, a[bx]
    cmp ax, a[bx+2]
    jl L3
    mov f, 0
    jmp L4

L3:

    add bx, 2
    loop L2

L4:

    cmp f, 1
    je L9

mov cx, n
mov bx, 0
L5:

    mov ax, a[bx]
    mov dx, 0
    div tri
    cmp dx, 0
    jne L6
    mov a[bx], 0

L6:

    add bx, 2
    loop L5

вывод
```

20 Обработка двумерных массивов

Обработка двумерных массивов в ассемблере можно использовать модификацию адреса по двум регистрам BX|BP or SI|DI. Причем один из них должен быть регистром. Модифицировать по парам BX|BP or SI|DI нельзя.

```
mov ax, a[bx][si]
Аисп=a+[bx]+[si]
```

Компиляторы большинства языков программирования размешает двумерные массивы по строкам

$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} = a_{11}a_{12}a_{13}a_{21}a_{22}a_{23}$

Способ размещения эллементов двурменого массива по строкам или по столбцам и соответсвующая их обработка определяеться программистом.

Пусть есть произвольная матрица NxM

$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & .. & a_{1n} \\ a_{21} & a_{22} & a_{23} & .. & a_{2n} \\ .. & .. & & & \\ a_{m1} & a_{m2} & a_{m3} & .. & a_{mn} \end{bmatrix} = |a_{11}\, a_{12}\, a_{13} \ldots a_{1n}|a_{22}a_{23} \ldots a_{2n}|a_{ij}^{K}| \ldots$

$k = (i-1) * N + j$

Обратную задачу по номеру k определить индексы i and j я сделаю самостоятельно.

Пример(дана матрица A размеров 10x20 каждый эллемент размером байт, найти сумму эллементов матрицы)

```
a db 10dup(20dup(?))
s dw 0


    mov ax, 0;sum
    mov cx, 200
    mov bx, 0

L1:

    add al, a[bx]
    inc bx
    loop L1
    mov s, ax
```

Пример(найти сумму диагональных эллементов матрицы A размером 10x10)

```
a db 10dup(10dup(?))
s dw 0


    mov ax, 0;sum
    mov cx, 10
    mov bx, 0

L1:
```

```
    add al, a[bx]
    add bx, 11
    loop L1
    mov s, ax
```

Отметим что решение этих двух задач практически не отличается от задач на обработку одномерных массивов, используеться один регистр модификатор.

Пример(дана матрица A 10x20, найти кол-во строк в которых первый эллемент строки встречаеться еще один раз)

```
a db 10dup(20dup(?))
s dw 0
comment @
al-кол-во строк
ah-первый эллемент строки
bx-индекс строки
si-индекс столбца
cx-счетчик цикла
dx-для хранения cx
@


    mov al,0
    mov bx, 0
    mov cx, 10

L:

    mov ah, a[bx];first ellement
    mov dx,cx

    mov cx, 19
    mov si, 0
    L1:
        add si, 1
        cmp a[bx][si], ah
        loopne L1; цикл пока a_ij <> ah <= 19
    jne L2
    inc al
    L2:
        mov cx, dx
        add bx, 20
        loop L
        mov ah, 0
        mov s, ax
```

Есть возможност организации доступа к эллементам матрицы по индексам, которые можно писать так

```
L:=1
for i:=1 ti M do
    fro j:=1 to N do
    begin
        r:=i*10+j;
        k:=(i-1)*n+j
        mov a[(k-1)*l],r
    end;h
```

Пример(Дана матрица A NxN каждый эллемент размер слово обнулить эллменты лежайшие ниже главной диагонали)

```
a dw 100dup(?)
n dw ?
i dw ?
;---------
    mov cx, n
    mov bx, 0
L5:
    push cx
    mov di,0
    mov cx, n
L6:
    inint a[bx][di]
    add di, 2
    loop L6
pop cx
add bx, n
add bx, n
loop L5
```

21 Множества

Практически ни в одном пк нет стандартного представления множеств, поэтому программистам самим приходится выбирать представление для множеств и реализовывать операции над ними.

```
Type
        Base = L..K;
        SetB = Set of Base;
Var
        M: SetB;
```

0	1	2	3	4	5	6	7
0	1	0	1	1	0	1	0

Таблица 1: Представаление множества

При моделировании множеств сдредствами любого языка, для представления каждой множественной переменной, резервируется так называемая множественная шкала. Если соответствующая позиция этой шкалы равна 1, то элемент базового типа принадлежит соответствующему множеству. Для переменных типа SetB резервируется пол одному байту (при условии, что L=0; K=7) тогда значение переменной M представлено шкалой(1).

Очевидно, чтол при таклом способе кодирования множеств, теоретико–множественные операции моделируются с помощью логических команд. В общем случае для представления множественного значения необходимо резервировать более 1 байта. При этом последний (самый правый) может использоваться не полностью. Для определения позиции некоторого произвольного элемента базового типа, соответствующего данной шкале, можно воспользоваться ...

Пример. Реализуем объединение двух множеств

```
L       EQU     10
K       EQU     200
N       EQU     (K-L)/8+1
M       DB      N DUP(?)
M1      DB      N DUP(?)
M2      DB      N DUP(?)
```

Необходимо использовать ... Надо в цикле логически сложить все байты M1 и M2

```
        SUB     BX,     BX
        MOV     CX,     N
L1:     MOV     AL,     M1[BX]
        OR      AL,     M1[BX]
        MOV     B[BX],  AL
        INC     BX
        LOOP
```

Операцию вычитания всами.

Пример. Пусть $L \leq P \leq K$, проверить принадлежность значения P базового типа множеству M.

```
        MOV     AL,     P
        SUB     AL,     L
        XOR     AH,     AH
        MOV     BL,     8
```

```
DIV     BL
MOV     BL,     AL
XOR     BH,     BH
MOV     AL,     M[BX]
MOV     CL,     AH
SHL     AL,     CL
```

22 Стек

22.1 Определение и описание.

Определение. Стек — специально отведенная область памяти для хранения промежуточных данных. Работа с ним ведется по принципу LIFO.

Эта обасть должна быть сегментом памяти (сегментом стека), тоесть ее размер не должен превышать 64K и начальный адрес должен быть кратен 16. На начало стека указывает регистр SS, на вершину SP. При иллюстрации работы со стеком отведенная область заполняется снизу вверх. При чтении из стека удаляется самый верхний элемент, поэтому низ стека фиксирован, а вершина все время сдвигается. В SP находится смещение относительно начала сегмента стека, и абсолютный адрес вершины задается парой SS:SP. Команды для работы со стеком используют только слова, обработка байтов или двойных слов требует некоторых ухищрений. Термин сегмент стека означает область памяти, которую потенциально могут занять данные. Собственно стек — данные, которые в настоящий момент хранятся, причем данные, адрес которых больше SP, считаются не относящимися к стеку. Чтобы зарезервировать M байтов под стек необходимо описать соответствующий сегмент.

```
ST    SEGMENT    STACK
      DB         N DUP (?)
ST    ENDS
```

Сегмент стека должен быть описан в программе, даже тогда, когда программа сама его не использует. Рекомендуемый размер стека — 128 байт.

22.2 Стековые команды

22.2.1 Запись и чтение

```
PUSH op ;r16, sr, m16
```

Выполнение этой команды можно описать как:

```
SP:= [SP] - 2;
[SP]:= op;
```

Следующая команда

```
POP op;
```

Извлекает слово из вершины стека, и помещает его по указанному операнду. Выполнение этой команды можно описать как:

```
op:=[SP];
SP:=[SP] + 2;
```

22.2.2 Запись и чтение регистра флагов.

```
PUSHF
```

По этой команде в стек записывается содержимое регистра флагов.

```
POPF
```

По этой команде извлекается из стека и помещается в регистр флагов.

Эти команды обычно используются для сохранения текущих состояний флагов и последующего их восстановления.

Пример. Записать в AX значение флага TF (8 бит) не изменяя другие биты.

```
PUSHF
PUSHF
POP     AX
MOV     CL,     8
SHR     AX,     CL
AND     AX,     1b
POPF
```

22.2.3 Запись и чтение РОН

Это команды:

```
PUSHA
POPA
```

Первая записывает в стек значение всех РОН: AX, CX, DX, BX, SP, BP, SI, DI в указанном порядке. В SP записывается значение до выполнения этой команды.

Команда POPA считывает из стека 8 слов и помещает их в регистры в обратной последовательности. После команды регистр SP указывает на состояние стека после этого считывания.

Команды PUSHA и POPA предназначены для работы с процедурами: для сохранения значений регистров при входе в процедуру и восстановления при выходе.

22.3 Некоторые приемы работы со стеком.

22.3.1 Сохранение регистров

Часто при организации вложенных циклов с использованием LOOP требуется сохранять значение регистра CX. Это можно делать следующим образом:

```
        PUSH    CX
        MOV     CX,     N
L:      ...
        LOOP    L
        POP CX
```

22.3.2 Пересылка данных через стек

Можно сделать так:

```
PUSH    Y
POP     X
```

22.3.3 Проверка на выход за пределы стека

Команды PUSH и POP не осуществляют проверку на выход за пределы сегмента стека. Если стек пуст, а мы читаем из стека ошибки не будет, но получим неопределенное значение. Аналогично при записи в стек, когда он уже исчерпан, мы можем получить фатальный результат. Проверку на границы стека можно сделать так:

```
SP = 0 ? - стек пуст?
SP = N ? - стек полон?
```

22.3.4 Очистка и восстановление стека

Очистка стека от N слов осуществляется так:

```
ADD     SP,     2*N
```

Часто стек используется для передачи данных (например в процедуры) или выполнение некоторых промежуточных операций. Тогда можно поступить так:

```
MOV     AX,     SP
... Работа со стеком
MOV     SP,     AX
```

23 Процедуры

23.1 Дальние переходы

Если программа состоит из нескольких сегментов кода и необходимо передать управление из одного сегмента в другой, то для этого нужно изменять не только IP (что делает JMP), но и CS.

Пример. Дальний переход:

```
C1      SEGMENT
        ASSUME  CS:C1
START:  MOV     AX,     0
        JMP     FAR     PTR L
C1      ENDS

C2      SEGMENT
        ASSUME  CS:C2
L:      INC     BX
C2      ENDS
        END     START
```

В начале каждого сегмента связывается CS с сегментом кода. Выполенеие команды перехода сводится к изменению адреса... Пара регистров CS:IP задает смещение данного сегмента. Изменение только IP означает переход внутри сегмента такие переходы называются близкими или внутрисегментными. При необходимости дальних переходов (из сегмента C1 на метку L из сегмента C2) должны изменяться и CS и IP. Дальние переходы только безусловные, прямые или косвенные. В языке ассемблера эти команды имеют тот-же мнемокод JMP, но испорльзуют другие типы операндов.

23.1.1 Дальний прямой переход

```
JMP     FAR     PTR <Метка>
```

Параметр FAR говорит о том, что метка находится в другом сегментекоманд. По этой команде в CS мпомещается начало того сегмента, где эта метка находится, а в IP — смещение этой метки внутри данного сегмента.

23.1.2 Дальний косвенный переход

Здесь указано поле памяти DD, в котором указан абсолютный адрес перехода в виде пары SEG:OFS, записанной в перевернутом виде. Если в команде указано имя X, которое описано до этой команды, то проблем не возникает, если же X описано позже, тодля правильной обработки ссылки вперед при дальнем косвенном переходе необходимо явно указать, что имя обозначает двойное слово:

```
JMP     DWORD   PTR X
```

1. Прямой близкий длинный:

```
JMP     X
```

2. Прямой близкий короткий:

```
JMP     SHORT   X
```

3. Дальний прямой:

```
JMP     FAR     PTR X
```

4. Близкий косвенный:

```
JMP     WORD    PTR X
```

5. Дальний косвенный:

```
JMP     DWORD   PTR X
```