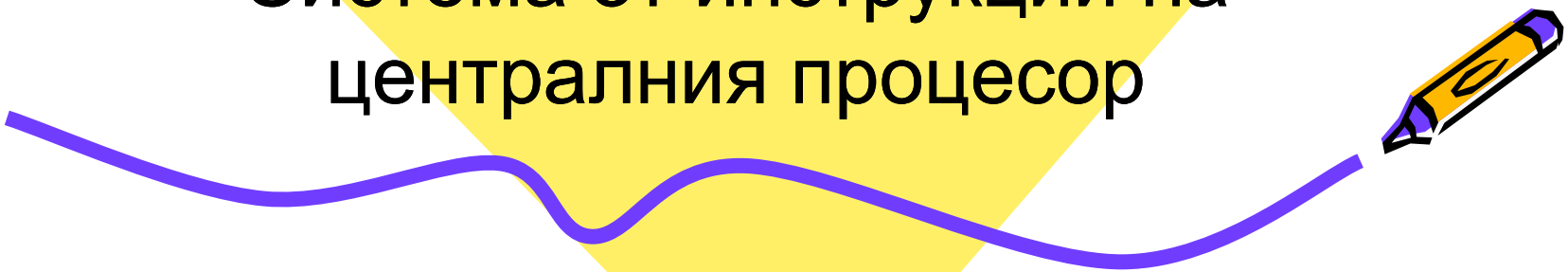
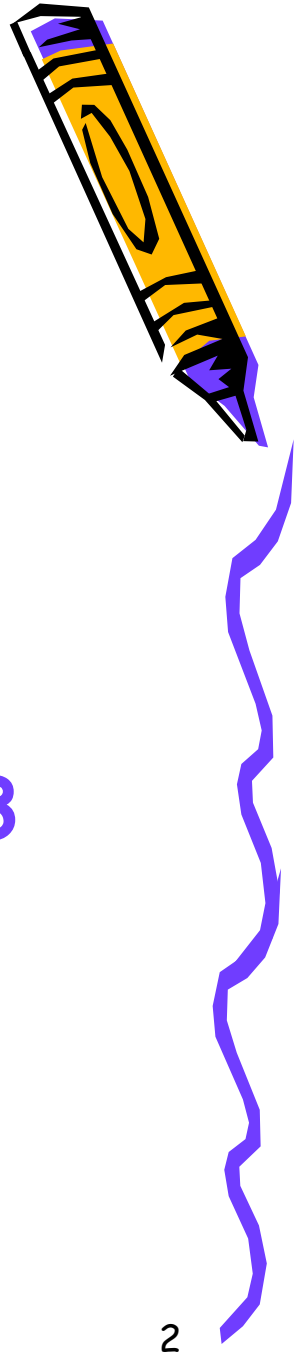
Компютърни архитектури - упражнения

Система от инструкции на централния процесор



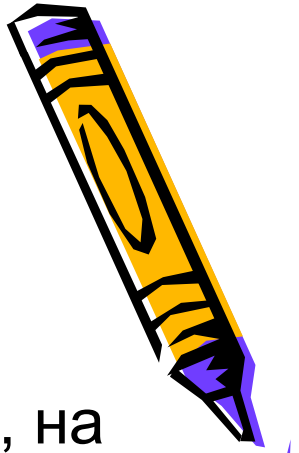
основен синтаксис на една инструкция на асемблер



етикет:

КОП операнд1, операнд2, операнд3
; коментар

Инструкции на Асемблер



- **Операнди** – обекти над които, или с помощта, на които се изпълняват действия, задавани от инструкции или директиви;
- **Машинните инструкции** могат да бъдат **без операнди**, да имат **един операнд. два операнда или три операнда;**
- Повечето инструкции имат **два операнда,** единият от които (**операнд2**) се нарича **източник,** а другият се нарича **приемник (операнд1).**

Инструкции на Асемблер

- При двуоперандните машинни инструкции са възможни следните съчетания на операнди:

➢ **Регистър – Регистър;**

➢ **Регистър – Памет;**

➢ **Памет – Регистър;**

➢ **Непосредствен операнд – Регистър;**

➢ **Непосредствен операнд – Памет;**

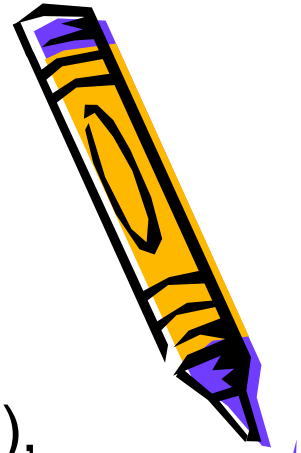
Инструкции на Асемблер

- Единият операнд може да се разполага в **регистър** или **памет**, но другият **задължително** трябва да бъде разположен в **регистър** или да бъде **непосредствен операнд**.
- **Непосредственият операнд** може да бъде само **източник**.

Изключения от тези правила:

- Инструкции за работа с паметта, които могат да прехвърлят данни от памет в памет;
- Инструкции за работа със ***стек***, които могат да прехвърлят данни от памет в стек;
- Инструкции за ***умножение***, които освен операнда указан в командата, използват неявно и друг операнд;

Операнди



- Операндите могат да бъдат **константи**(числа), **регистри, клетки памет** и **символни идентификатори**;
- Понякога се използват изрази, които са комбинация от числа, регистри, клетки памет, идентификатори с аритметични, логически, побитови и атрибутивни оператори

Целочислени машинни инструкции

- **Инструкции за прехвърляне на данни**
- **mov <оп.1 >, <оп.2 >**

☞ **Не може** да се осъществява пренос от една област на паметта в друга. При необходимост се използва като буфер произволен, достъпен в този момент регистър с общо предназначение;

```
mov al,prom1
mov prom2,al
```

Инструкция за прехвърляне на данни mov

- **Не може** да се зарежда в сегментния регистър стойност непосредствено от паметта. Използва се междинен РОП или стек;
- **Не може** да се прехвърля съдържанието на един сегментен регистър в друг сегментен регистър – в системата инструкции няма съответен код на операция. Използват се междинни РОП.

mov ax.ds

mov es.ax

Инструкция за прехвърляне на данни

- Съществува и друг начин за прехвърляне с използване на **стек** и команди:

  push ds ; прехвърля съдържанието на
  ; регистър ds в стек

  pop es ; записва в es число(стойността) от
  ; стека

☞ **Не може** да се използва сегментен регистър **CS** като операнд **приемник**; В архитектурата на процесора IA-32 двойката CS:IP съдържа адреса на следващата за изпълнение команда. Изменението на съдържанието на CS чрез команда MOV фактически ще бъде операция за преход, а не прехвърляне на данни;

Инструкция за прехвърляне на данни mov

- **mov ax , [bx]**
- **BX** съдържа **адрес** на някакво поле (използва се косвена базова адресация);
- **Какво адресира регистър BX в паметта – дума или байт**? Обикновено се приема размерът на **най-големият операнд**, но може и да се получи предупредително съобщение за възможно несъвпадение на типа на операндите.

Инструкция за прехвърляне на данни mov

- Да допуснем, че е необходимо да се инициализира поле, адресирано чрез регистър BX с 0.

  mov [bx] ,0

- Каква машинна команда се конструира: за инициализация на **байт**, на **дума** или **двойна дума**?
- В тези случаи е необходимо да се уточни типа на използваните операнди;
- За тази цел съществува специален оператор **PTR**

Инструкция за прехвърляне на данни mov

- **mov ax,word ptr[bx]** ; ако bx] адресира
  ; дума в паметта
- **mov ax,byte ptr[bx]** ; ако [bx] адресира
  ; байт в паметта
- **mov eax,dword ptr[ebx]** ; ако [ebx]
  ; адресира двойна дума

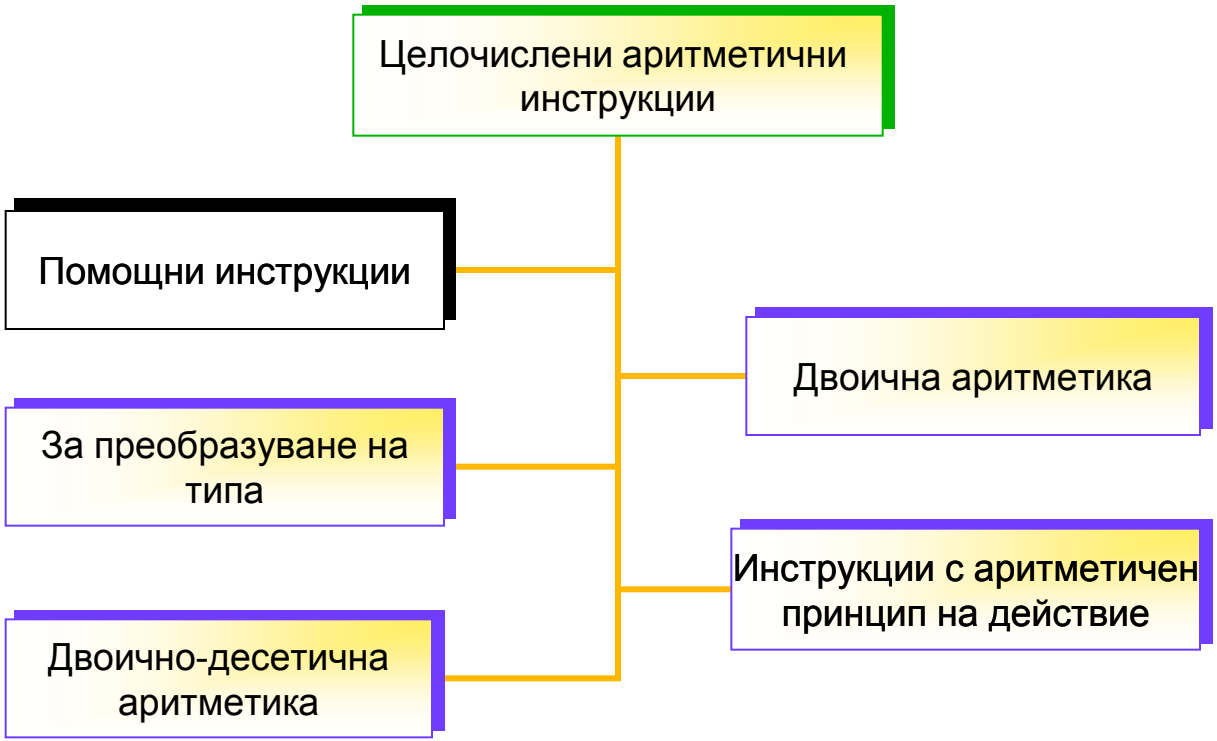
Инструкция за прехвърляне на данни xchg

- За двупосочно прехвърляне на данни се използва команда **xchg**
- **Операндите трябва да бъдат от един и същи тип;**
- **Не е допустимо да се разменя съдържанието на две клетки от паметта;**

xchg eax,eЬx ; разменя съдържанието на
; двата регистъра eax и eЬx

xchg ax,word ptr [si] ; разменя съдържанието на регистър ax и дума в паметта по адреса в [si];

Аритметични инструкции



Аритметични операции над цели двоични числа – Двоично събиране

- **Инкрементиране** (увеличаване на стойността на операнда с 1):

  inc операнд

- **Събиране** (оп.1=оп.1+оп.2):

  add операнд1,операнд2

- **Събиране с отчитане на флага за пренос CF** (оп.1=оп.1+оп.2+CF)

  adc операнд1,операнд2

Аритметични операции над цели двоични числа – Двоично изваждане

- **Декрементиране** (намаляване на стойността на операнда с 1):

  dec операнд

- **Изваждане** (оп.1=оп.1-оп.2):

  sub операнд1,операнд2

- **Изваждане с отчитане на заем от флага за CF** (оп.1=оп.1-оп.2-CF)

  sbb операнд1,операнд2

Аритметични операции – събиране и изваждане

- При операции събиране и изваждане, освен флаговете **CF** и **OF** на флаговия регистър EFLAGS, се използват:
- **ZF** — установява се в 1, ако резултата от операцията е равен на 0, и в 0, ако резултата не е равен на 0.
- **SF** — флаг за знак, стойността на който след аритметична операция(и не само) съвпада със стойността на старшия бит на резултата (т.е. с бит 7,15 или 31 (по този начин, този флаг може да се използва за операции над числа със знак).

- 1. Да се намери сумата на 2 цели числа.

```
program zad1;
var
a,b:integer;
function suma(a,b:integer):integer;
asm
mov eax,a
mov ebx,b
add eax,ebx
mov Result,eax
end;
```

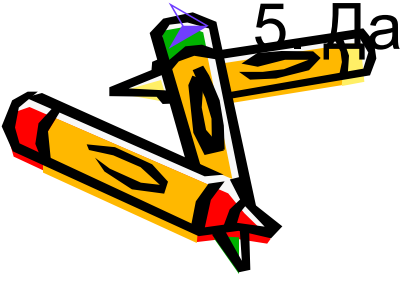
```
begin
write('a= ');read(a);
write('b= ');read(b);
write('suma= ',suma(a,b));
readln;
end.
```

Примери и задачи:

- 2. Да се намери разликата: x := y - z;

```
function razl (a,b:integer) :integer;
asm
mov eax,a
mov ebx,b
sub eax,ebx
mov Result,eax
end;
```

- 3. Да се пресметне израза x:= 20 + (a-b) + c;
- 4. Да се намери: w:= w - (x+z);
- 5. Да се намери: z:= (100+z) – (a+b);



Умножение на двоични числа без знак

- **mul** операнд

В тази команда има **само един** операнд, другият операнд е зададен **неявно**. Неговото местоположение е фиксирано и зависи от размера на другия множител.

В общият случай резултатът от умножението е число, по-голямо от всеки от множителите и неговият размер и местоположение трябва да са определени еднозначно.

Умножение на двоични числа без знак

Първи множител	Втори множител	Резултат
Байт	**AL**	**16**бита в **AX:AL**- младша част,**AH** – старша част на резултата
Дума	**AX**	**32**бита в двойката **DX:AX**- младша част, **DX** – старша част на резултата
Двойна дума	**EAX**	**64**бита в двойката **EDX:EAX** – младша част, **EDX** – старша част на резултата

Умножение на двоични числа без знак

- Произведението се състои от **две части** и в зависимост от размера на операндите се разполага на различни места;
- **Динамично**(по време на изпълнение на програмата) се разбира дали резултатът се събира в един регистър,или старшата част се разполага в друг регистър;
- Използват се флаговете за **пренос CF** и за **препълване OF** ;
- Ако старшата част на резултата е **0**, след завършване на операцията **CF=0** и **OF=0**;

Умножение на цели числа със знак

- imul операнд_1[,операнд_2,операнд_3]

- Изпълнява се по същият начин както mul;
- Особености при формиране на знака:
  - Ако резултатът е малък и се събира в един регистър (т.е. Ако **CF=OF=0**), то съдържанието на другия регистър (старша част) е разширението на знака – всички негови битове са равни на старшия бит(знаков разряд) на младшата част на резултата;
  - Ако **CF=OF=1**, знакът на резултата е знаков бит на старшата част на резултата, а знаковия бит на младшата част е значещ бит на двоичния код на резултата;

Примери и задачи:

- 1.Да се пресметне x:= 5*(a-b) +c;

```
asm
    mov eax,a      ; eax=a
    sub eax,b      ; eax=eax-b
    mov ebx,5      ; ebx=5
    imul ebx       ; eax=eax*ebx
    add eax,c      ; eax=eax+c
    mov x,eax      ; x=eax
end;
```

Примери и задачи:

- 2. Да се пресметне x:= (a*2) + (b*c);

```
asm
mov eax,2  ;eax=2
imul a     ;eax=eax*a
mov ecx,eax
mov eax,b
imul c     ;eax=eax*c
add ecx,eax
mov x,ecx
end;
```

Примери и задачи:

- 3. Да се пресметне w:=(a+b)*(y+z);

```
asm
    mov  eax,a
    add  eax,b
    mov ebx,y
    add  ebx,z
    imul eax, ebx
    mov w, eax
end;
```

Примери и задачи:

- 4. Да се пресметнат следните изрази:
  - X:=(a*8+2) – (b*c);
  - W:=(b-10*a) * (b+2*c);
  - X:=(a-b) * (b+c) * (c-d).

Деление на двоични числа без знак

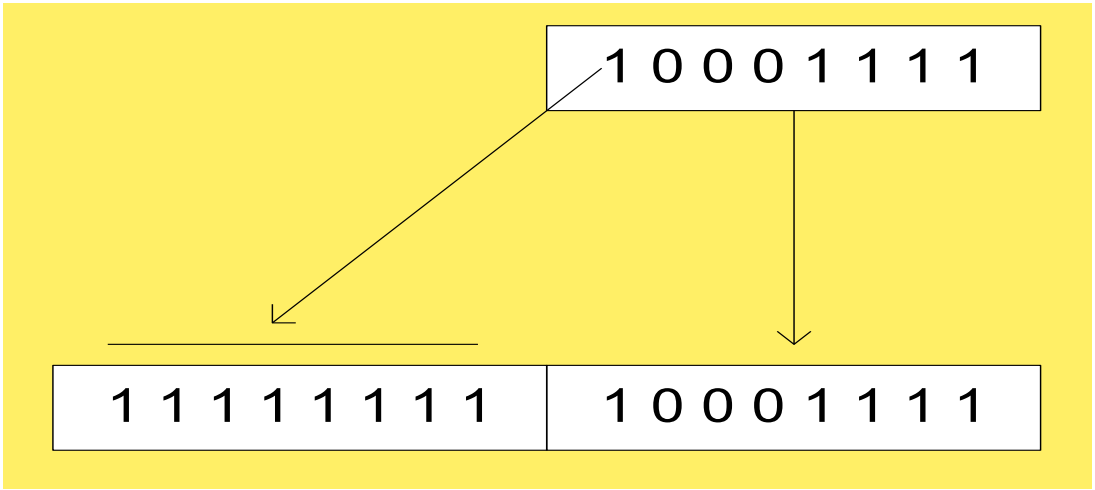
- **div** делител
- Делителят може да се намира в памет или в регистър и има размер 8,16 или 32 бита;
- Местоположението на делимото е фиксирано и зависи от размера на операнда;
- Резултатът от операцията се състои от частно и остатък;

Разположение на операнди и резултат при деление

делимо	делител	частно	остатък
Дума в регистър **AX**	Байт в регистър или в клетка от паметта	Байт в регистър **AL**	Байт в регистър **AH**
Двойна дума В **DX** старша ч. В **AH** младша ч.	Дума в регистър или клетка от паметта	Дума в регистър **AX**	Дума в регистър **DX**
Четворна дума **EDX** старша ч. **EAX** младша ч.	Двойна дума в регистър или в клетка от паметта	Двойна дума в регистър **EAX**	Двойна дума в регистър **EDX**

Деление на двоични числа със знак

- **idiv** делител
- Използва се по същият начин както **div;**
- Знаковите цели числа е необходимо да бъдат разширени предварително (знаковия бит);



Инструкции **CBD,CWD и CDD**

- **CBW, CWD, и CDQ** извършват знаково разширение на операндите:
  - **CBW** (convert byte to word) разширява старшият бит на **AL** във всички битове на **AH**
  - **CWD** (convert word to doubleword) разширява старшия бит на **AX** във всички битове на **DX**
  - **CDQ** (convert doubleword to quadword) разширява старшия бит на **EAX** във всички битове на **EDX**
- Например::

```
mov eax,0FFFFFF9Bh
cdq      ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```

Примери:

- Например:8-битово деление -48/5

```
mov al,-48
cbw                    //разширява AL в  AH
mov bl,5
idiv bl          //al=-9, ah=-3
```

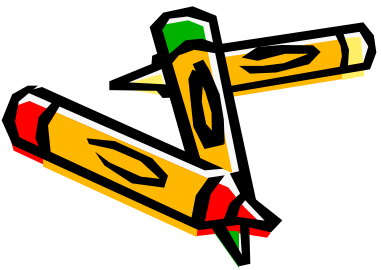
- 16 – битово деление

```
mov  ax,-48
cwd                // разширява AX в DX
mov  bx,5
idiv bx                // AX = -9,  DX = -3
```

Примери и задачи:

- 1. Да се пресметне z:=(a+b) / b;

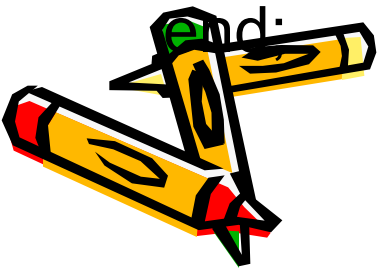
```
asm
    mov edx,0
    mov eax,a
    add  eax,b
    mov ecx,b
    idiv ecx
    mov z,eax
  end;
```



- 2. Да се пресметне:

var3 = (var1 * (-var2)) / (var3 – ebx)

```
asm
    mov eax,var1
    mov edx,var2
    neg edx
    mul edx
    mov ecx,var3
    sub ecx,ebx
    div ecx       ; eax = частно
    mov var3,eax
end;
```



Примери и задачи:

- 3. Да се пресметне x := (y+z) * (a-b) / 10;

Пресмятането се свежда до 4 последователни действия:

Temp1 := (y+z)

Temp2 := (a-b)

Temp1 := Temp1 * Temp2

X := Temp1 / 10

Примери и задачи:

```
asm
    mov edx,0
    mov eax,y
    add eax,z          // eax = y+z
    mov ebx,a
    sub ebx,b         // ebx = a-b
    imul eax, ebx    // eax =eax*ebx
    mov ecx,10
    cdq
    idiv   ecx          //eax=eax/ecx
    mov x, eax
end;
```