Методи на адресация в микропроцесорната фамилия x86

Основни аспекти с примери

Кирил Иванов

Март 2018 година

В асемблерния език почти всички аритметични изрази се изчисляват от транслатора (и трябва да са константни), освен две изключения: мащабирането на индексен регистър и сумирането на базов или индексен регистър се извършват по време на изпълнение на съответната команда. Причината е, че в тези два случая в изчислението участвува стойността на регистър, която ще бъде известна само в момента на изпълнение на командата и може да бъде различна при различните изпълнения на командата.

1. Неявна (или по подразбиране) адресация

Адресът на операнда се подразбира от самата команда.

Примери за неявна адресация:

```
stc // записва 1 във флага CF;
    // неявно е зададен операнд CF
clc // записва 0 във флага CF;
    // неявно е зададен операнд CF
```

> ***Една особеност на терминологията:***
> *Понякога като самостоятелен вид адресация се разглежда подразбиращата се адресация при команди, подобни на movsb.*
> *(movsb премества един байт от адрес [esi] на адрес [edi] и после прибавя или изважда, според флага DF, единица към всеки от двата регистъра esi и edi.)*
> *В такива команди реално отсъствува адрес (в самата команда), което ги причислява към неявната адресация. Спецификата на такива команди е в това, че адресът на операнда или операндите* ***може да бъде различен*** *при различни изпълнения на командата, защото се взема от подразбиращи се регистри, а те могат да си променят стойностите между поредните изпълнения на командата.*

Неявната адресация е удобна за команди, чиято специфика налага използването на точно фиксирани местоположения (на самите операнди или на техните адреси). Обикновено тези местоположения са регистри или техни части с уникално предназначение.

2. Непосредствена адресация

Операндът е част от командата.

Схема на непосредствена адресация

Примери за непосредствена адресация:

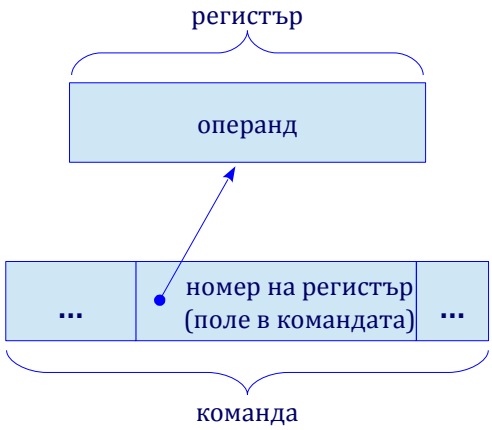
```
mov ecx, 10 // записва 10 е регистъра ecx;
            // непосредствен операнд е  10
add eax, 2*15-100 // записва в регистъра eax сумата на eax и 2*15-100;
                  // непосредствен операнд е  2*15-100;
                  // стойността на 2*15-100 изчислява транслаторът
```

Непосредствената адресация е удобна за операнди литерали числа.

3. Регистрова адресация

Операнд е съдържанието на регистър.

Схема на регистрова адресация



Примери за регистрова адресация:

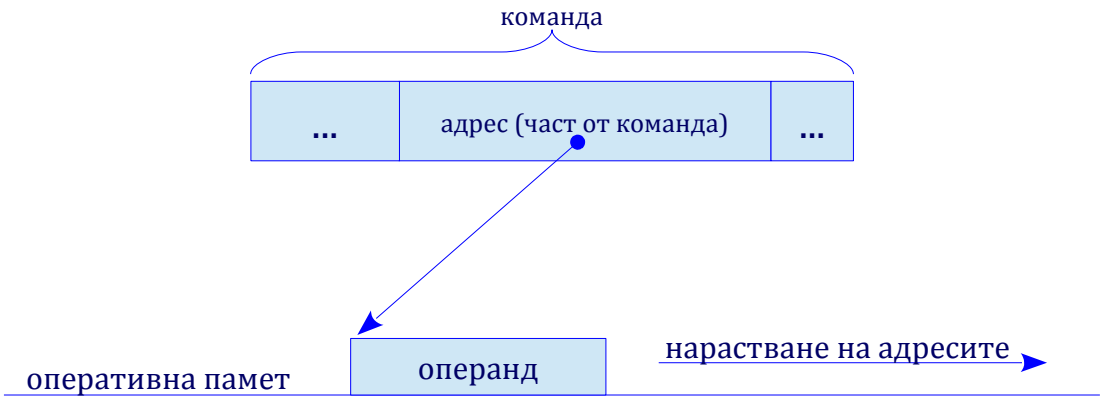
```
sub eax, esi // записва в регистъра eax разликата eax-esi;
             // и двата операнда са регистрово адресирани
inc ecx // увеличава с 1 (съдържанието на) регистъра ecx;
        // операнд регистър е ecx
```

Регистровата адресация е удобна за междинни стойности и за често използвани данни (регистровата памет е пределно бърза, по-бърза може да бъде само регистрова памет в още по-бърз процесор).

4. Пряка адресация

Адресът на операнда е пряко назован в командата и е част (поле) от командата.

Схема на пряка адресация



Примери за пряка адресация:

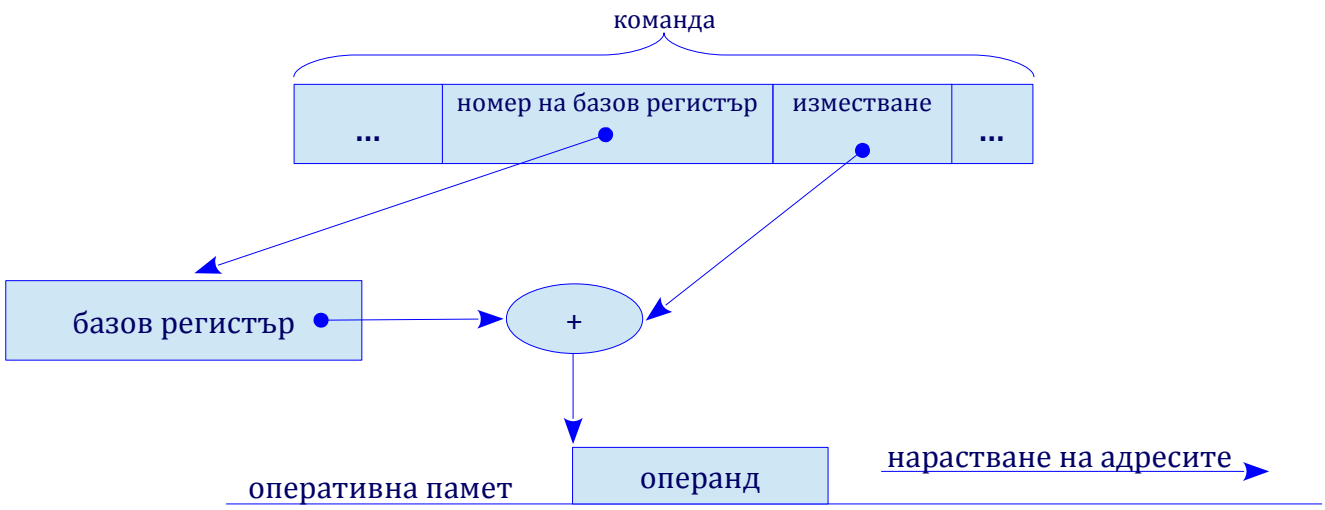
```
int ar[10];
__asm mov ar, 2 // записва 2 в ar[0];
                // пряко адресиран операнд е 1-я;
                // ar назовава адреса на ar[0] (точно както и в C++)
__asm mov [ar], 2 // има същия смисъл, като горната команда
__asm add eax, ar+4 // записва в eax сумата eax+ar[1];
                    // пряко адресиран операнд е 2-я;
                    // ar+4 назовава адреса на ar[1] (точно както и в C++)
__asm add eax, [ar+4] // има същия смисъл, като горната команда
```

Пряката адресация е удобна за достъп до именувани стойности или системни константи (например `[4]`), разполагани винаги на едни и същи адреси. Но командата винаги работи с една и съща данна.

5. Базова с изместване адресация

Адресът на операнда е сума от съдържанието на базов регистър и изместване. В командата се съдържат номерът на регистъра и изместването. Сумирането на базовия регистър и изместването става по време на изпълнението на командата.

Схема на базова с изместване адресация



Примери за базова с изместване адресация:

```
int ar[20];
__asm {
lea esi, ar // записва адреса (на) ar в регистъра esi;
            // 1-ят операнд е адресиран регистрово, а 2-ят е адресиран пряко
mov [esi],eax//записва стойността на eax на адрес (стойността на)esi,т.е.в ar[0];
             // 1-ят операнд е адресиран базово без изместване;
             // 2-ят операнд е адресиран регистрово
mov [esi+4], eax // записва стойността на eax на адрес esi+4
                 // (сумата от стойността на esi  и 4);
                 // 1-ят операнд е адресиран базово с изместване;
                 // 2-ят операнд е адресиран регистрово
}
```

Базовата с изместване адресация е удобна за достъп до полетата на структури.

6. Индексна с мащабиране и изместване адресация

На ниво асемблерен език адресът на операнда е сума от базов адрес, изместване и произведението на индексен регистър с мащаб.

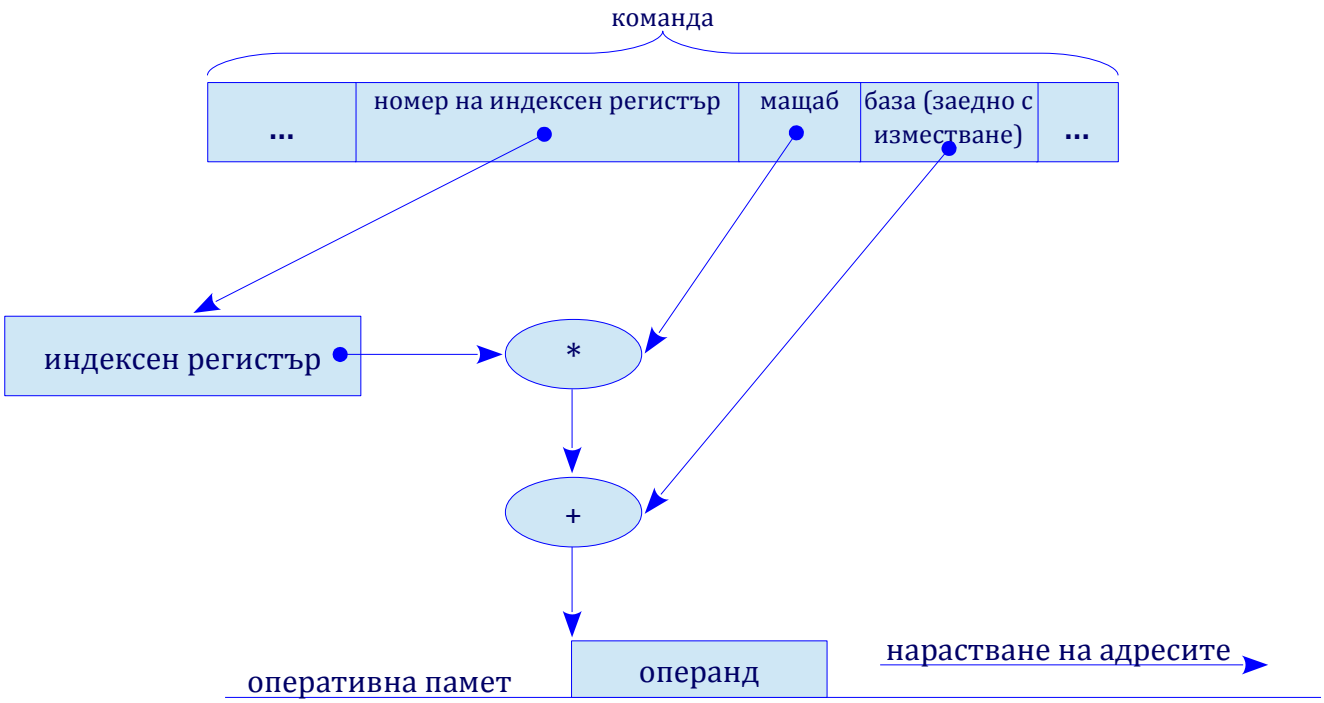
На физическо ниво адресът е сума от произведението на индексен регистър с мащаб и изместване.

В машинната команда се съдържат (като самостоятелни полета) номерът на индексния регистър, мащабът и изместването.

Изчисляването на адреса на операнда става при изпълнение на командата. Обаче, когато в асемблерната команда има базов адрес и изместване, още по време на транслация се сумират базовият адрес и изместването и резултатът се записва в машинната команда като изместване.

Мащабирането, т. е. умножаването по мащаб, задължително става по време на изпълнение на съответната команда. Възможните мащаби са 1, 2, 4 и 8. При мащаб 1 тази адресация физически е идентична с базовата с изместване.

Схема на индексна с мащабиране и изместване адресация



Примери за индексна с мащабиране и изместване адресация:

```
int ar[20];
__asm {
mov esi, 8 // записва 8 в регистъра esi;
           // 1-и операнд – регистрова адресация;
           // 2-и операнд – непосредствена адресация
mov ar[esi], -30 // записва -30 в ar[2];
                 // 1-и операнд – индексна адресация без изместване;
                 // 2-и операнд – непосредствена адресация
```

```
mov [ar+esi], -30 // еквивалентно на горното
mov esi, 3 // записва 3 в регистъра esi;
           // 1-и операнд – регистрова адресация;
           // 2-и операнд – непосредствена адресация
mov ar[esi*4], -50 // записва -50 в ar[3];
                   // 1-и операнд – индексна адресация без изместване;
                   // 2-и операнд – непосредствена адресация
mov ar[esi*4+ -2-2], -40 // записва -40 в ar[2];
    // 2-и операнд – непосредствена адресация;
    // 1-и операнд – индексна адресация с изместване
    // (ar – база; esi – индексен регистър; 4 – мащаб; -2-2 – изместване);
    // в съответната на тази машинна команда ще има изместване ar-4
mov ar[esi*4-4], -40  // има същия смисъл, като горната команда
}
```

Индексната с мащабиране и изместване адресация е удобна за достъп до масиви, включително и от структури, но е най-подходяща, когато елементите на масива заемат по 1, 2, 4 или 8 байта, а такива са най-често използваните масиви – от знакове, числа и адреси (указатели или референции).

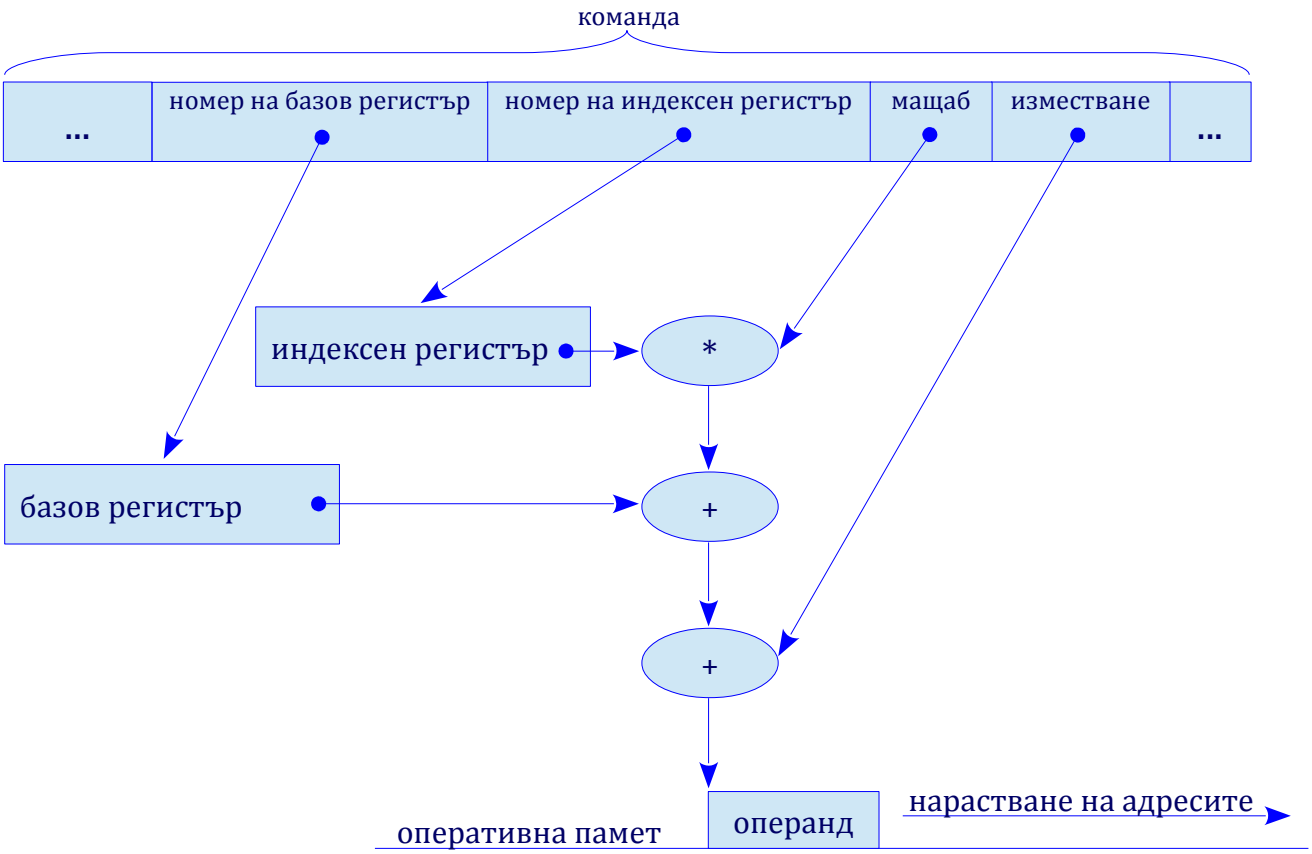
7. Базово-индексна с мащабиране и изместване адресация

Адресът на операнда е сума от базов регистър, изместване и произведението на индексен регистър с мащаб.

Изчисляването на адреса на операнда става при изпълнение на командата.

В командата се съдържат (като самостоятелни полета) номерът на базовия регистър, изместването, номерът на индексния регистър и мащабът. Възможните мащаби са 1, 2, 4 и 8.

Схема на базово-индексна с мащабиране и изместване адресация



Примери за базово-индексна с мащабиране и изместване адресация:

```
int ar[20];
__asm {
lea ecx, ar // записва адреса на ar[0] (адреса на масива) в регистъра ecx
mov esi, 6 // записва 6 в регистъра esi
mov [ecx + esi*4 + -4*1+0*20], eax // записва eax в ar[5]
     // 1-ят операнд е адресиран базово-индексно с мащабиране и изместване;
     // съответно: ecx – база (още базов регистър);
     //            esi – индекс (още индексен регистър);
     //            4 – мащаб;
     //            -4*1+0*20 – изместване (изчислява се при транслация)
}
```

Базово-индексната с мащабиране и изместване адресация е удобна за достъп до двумерни масиви, включително и от структури, но пак като предишната е най-подходяща за масиви, чиито елементи заемат по 1, 2, 4 или 8 байта (каквито са адресите, знаковете и почти всички числа).