Методи на адресация
в 64-разрядните процесори
на микропроцесорната фамилия x86-64

Кирил Иванов

Март 2024 година

1. Неявна адресация (или адресация по подразбиране)

Операндът се подразбира от самата инструкция. На асемблерно ниво названието на инструкцията, а на машинно ниво номерът на инструкцията, еднозначно определя кой точно е операндът.

> ***Пример***
>
> ***stc***
>
> *Тази команда записва единица във флага **CF**.*

Неявната адресация е удобна, когато се налага използването на операнди с точно фиксирани местоположения. Обикновено при тази адресация операндите са или флагове, или регистри с уникално предназначение (флаговете също може да се интерпретират като части на един и същ флагов регистър).

2. Адресация с автоувеличение или с автонамаление

Този вид адресация понякога се отбелязва като самостоятелен вид, защото по много удобен начин съчетава идеята за подразбиращо се местоположение на операндите с възможността за обработка на различни операнди в различни моменти на изпълнението на една и съща инструкция.

Характерното тук е, че в съответните машинни инструкции няма никакво указание за местоположението на операндите, защото техните адреси по подразбиране се вземат от регистрите ***rsi*** при четене и ***rdi*** при запис. Обаче по същата причина в различни моменти от изпълнението на такава инструкция е възможно да се обработват различни операнди (в различни моменти може да бъде различно съдържанието на ***rsi*** и ***rdi***).

Съответно термините **авто...** показват, че при изпълнението на такава инструкция използваните регистри ***rsi*** и ***rdi*** се променят автоматично за насочване към следващата или предхождащата данна (т. е. данна, имаща съседен адрес) със същата разрядност като обработената. Посоката за промяната се определя от флага **DF**. При **DF=0**, се минава към следващата данна (автоувеличение), а когато **DF=1**, се минава към предхождащата данна (автонамаление).

> ***Пример*** *(това е една инструкция)*
>
> ***repnz*** `movsq`
>
> *Тази команда първо проверява регистъра **rcx**. Когато **rcx** е нула, нищо не прави. Когато **rcx** е различен от нула, премества отново и отново 8-байтова стойност от адрес [**rsi**] на адрес [**rdi**] докато или се нулира регистърът **rcx**, или се получи стойност нула във флага **ZF**. След всяко преместване или увеличава, при **DF=0**, с 8 всеки от регистрите **rsi** и **rdi**, или намалява, при **DF=1**, с 8 всеки от регистрите **rsi** и **rdi**. Също след всяко преместване намалява с единица регистъра **rcx**. Това, че става дума точно за 8-байтови данни, се указва от буквата „q“ в инструкцията. Повторението в зависимост от регистъра **rcx** се указва от префикса „**rep**“, а зависимостта от флага **ZF** се указва с буквите „**nz**“. Така в примера префиксът **repnz** указва, че изпълнението на инструкцията ще се повтори многократно (в зависимост от стойностите на регистъра **rcx** и получаваните в процеса на изпълнението стойности на флага **ZF**). Съответно буквите „`movs`“ указват, че се извършва преместване с автоувеличение или автонамаление.*

Автоматичната промяна на адресни регистри позволява инструкциите с тази адресация да се изпълняват многократно, т. е. да работят като цикъл, състоящ се от една инструкция.

По този начин една инструкция може да обработва цяла редица от данни с еднаква разрядност, разположени една до друга в паметта. Такъв цикъл е съществено по-ефективен, отколкото съответния на него, организиран чрез няколко машинни инструкции.

Освен при нулиране на брояча ***rcx***, подобни зацикляния може да се прекратят и според стойността на флага **ZF** (флаг за нулев резултат). За указване на условието за прекратяване на зациклянето в асемблерния език се използват префикси ***rep***, ***repz*** (или ***repe***) и ***repnz*** (или ***repne***), а в машинния формат на инструкцията се добавя съответен префиксен байт. За брояч на повторенията се привлича регистърът ***rcx***. Това е част от специалното му предназначение.

3. Непосредствена адресация

Операндът е част от инструкцията.

Схема на непосредствена адресация

Пример (вторият операнд):

```
mov dword [Label21], -105
```

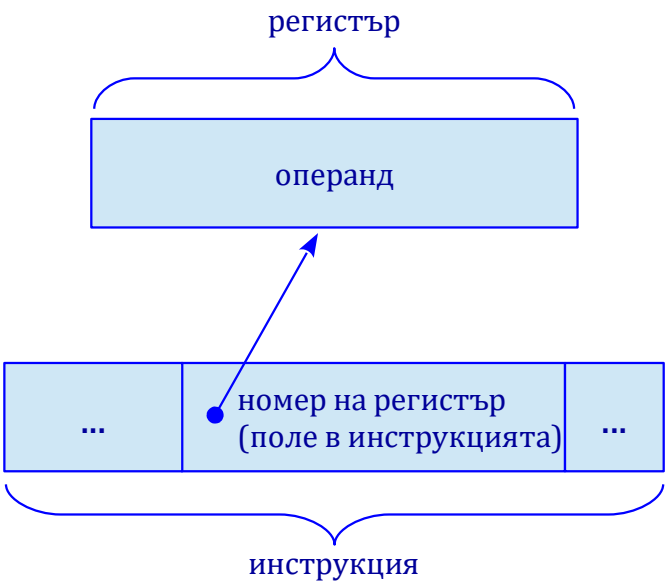
Тази инструкция записва `-105` *в паметта на адрес* `[Label21]` *във вид на 4-байтово цяло число в допълнителен код.*

Непосредствената адресация е удобна за операнди литерали.

4. Регистрова адресация

Операнд е съдържанието на регистър.

Схема на регистрова адресация



Пример *(първият операнд)*

```
mov rcx, -105
```

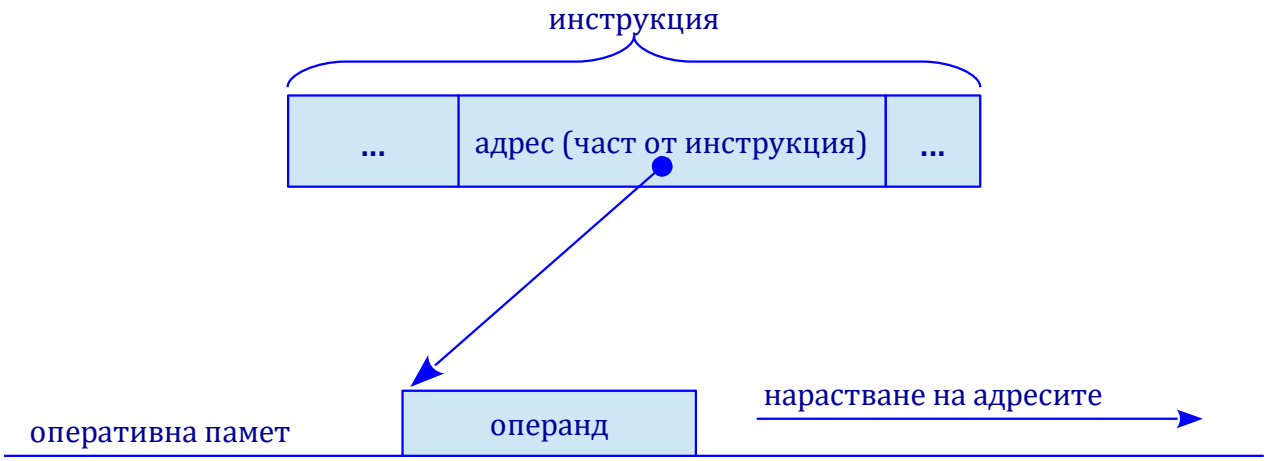
Тази команда записва `-105` *в регистъра* `rcx`*.*

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).

5. Пряка адресация

Адресът на операнда е пряко назован в инструкцията, той е част (поле) от инструкцията.

Схема на пряка адресация



> ***Пример*** *(първият операнд)*
>
> ```
> mov [Label23], rax
> ```
>
> *Тази команда записва стойността на регистъра* **`rax`** *в паметта на адрес* `[Label23]`*.*

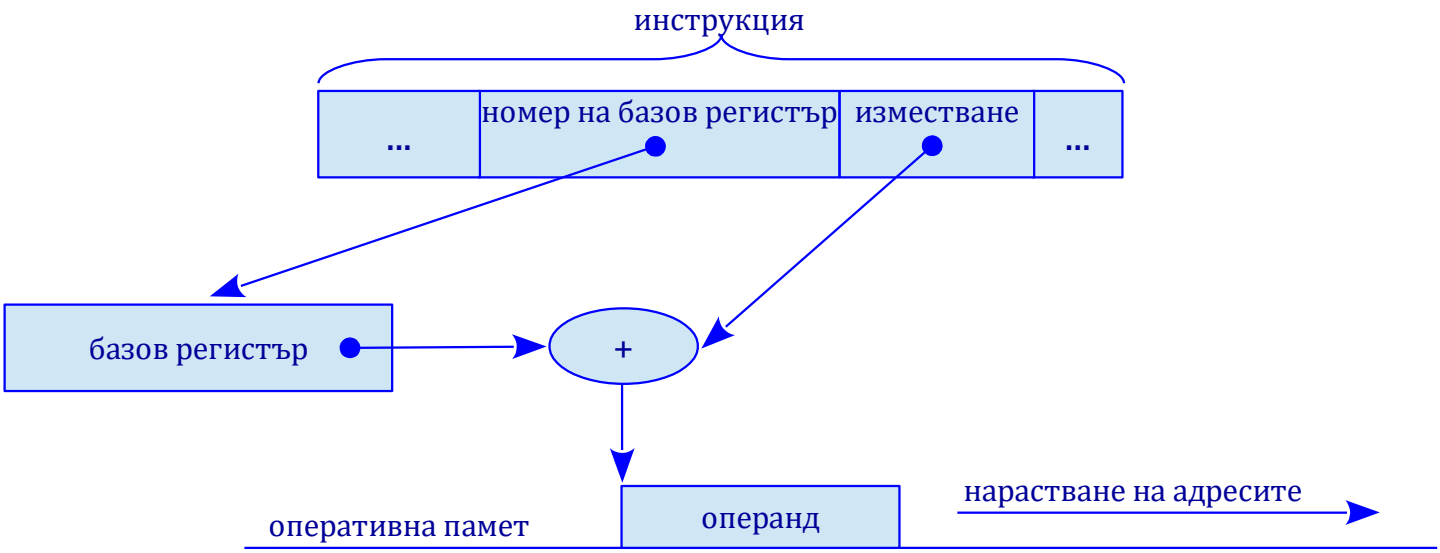
Пряката адресация е удобна за достъп до паметта чрез етикети или за достъп до системни константи, разполагани винаги на едни и същи адреси.

Обаче такава инструкция винаги работи с една и съща данна, което ограничава гъвкавостта на алгоритъма.

6. Базова адресация с изместване

Адресът на операнда е сума от съдържанието на базов регистър и изместване. В инструкцията се съдържат номерът на регистъра и изместването. Сумирането на базовия регистър и изместването става *по време на изпълнението* на инструкцията (и само тогава е възможно, защото зависи от моментната стойност на регистъра).

Схема на базова с изместване адресация



> ***Пример*** *(първият операнд)*
>
> ```
> mov [rsi + 8], rax
> ```
>
> *Тази команда записва стойността на регистъра* ***rax*** *в паметта на адрес* ***[rsi + 8]****, т. е. на адрес сумата от* ***8*** *и стойността на регистъра* ***rsi****.*

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

7. Индексна адресация с мащабиране и изместване

Мащабиране на индексен регистър означава умножение на регистъра с число, наричано мащаб. За да се ускори този процес се използват само множители (мащаби) 1, 2, 4 и 8, защото при тях умножаването се свежда до поразрядно изместване наляво на толкова позиции, колкото е степенният показател на степента на двойката, с дописване на нули в освобождаваните позиции. Такова умножение е пределно бързо действие.

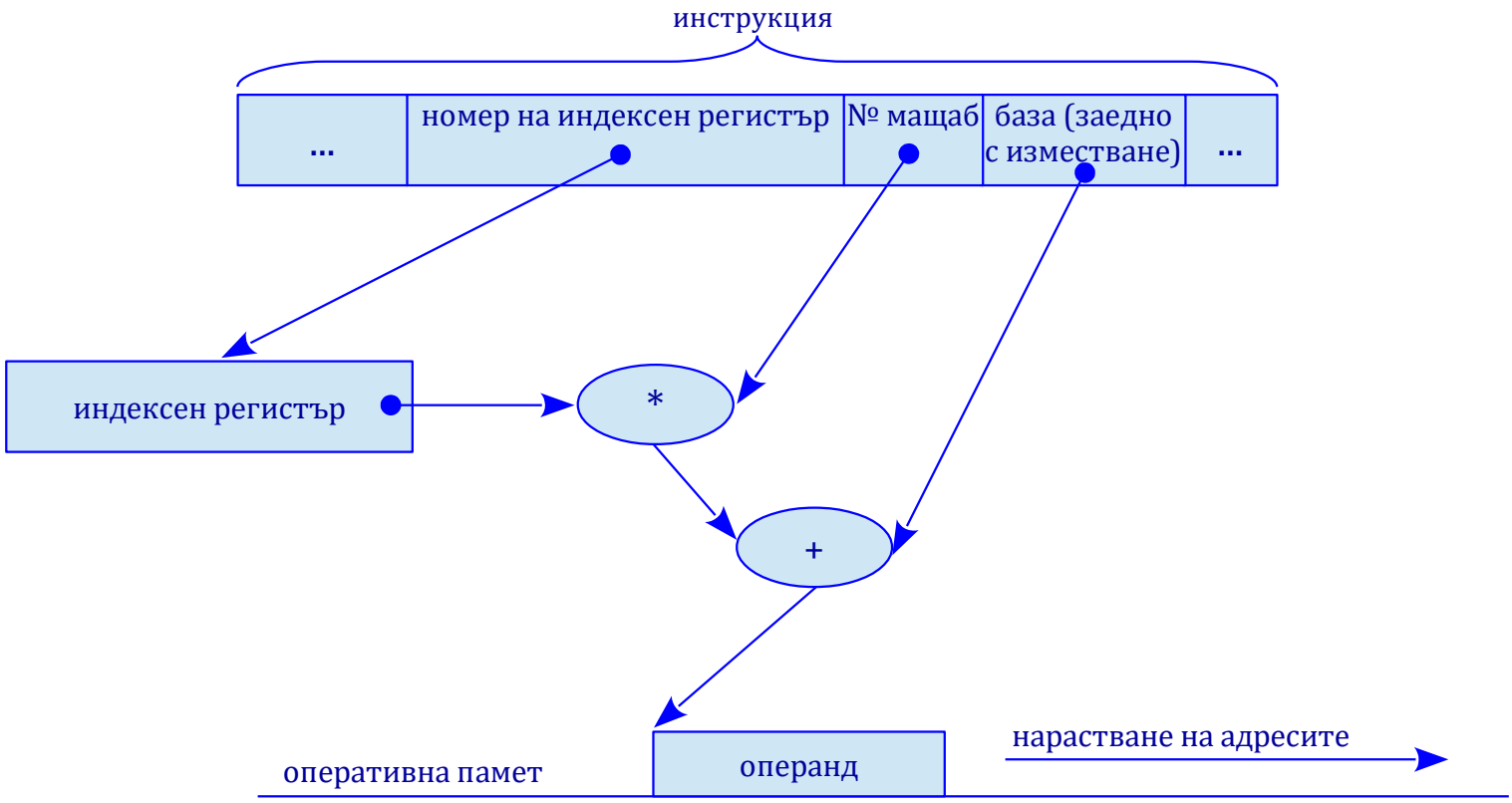
На ниво асемблерен език адресът на операнда е сума от базов адрес, изместване и произведение на индексен регистър с мащаб. Мащабирането (умножението на индексния регистър с мащаба) и сумирането на полученото произведение с базата стават *по време на изпълнението* на инструкцията (и само така би могло да бъде, защото резултатът зависи от моментното съдържание на регистъра).

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

Сумирането на базата и изместването, написани в асемблерния код, става *по време на транслация* и резултатът се интерпретира като изместване, записвано в инструкцията.

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

Схема на индексна с мащабиране и изместване адресация



Разрешените мащаби са 1, 2, 4 и 8, но при мащаб 1 тази адресация физически е идентична с базовата с изместване.

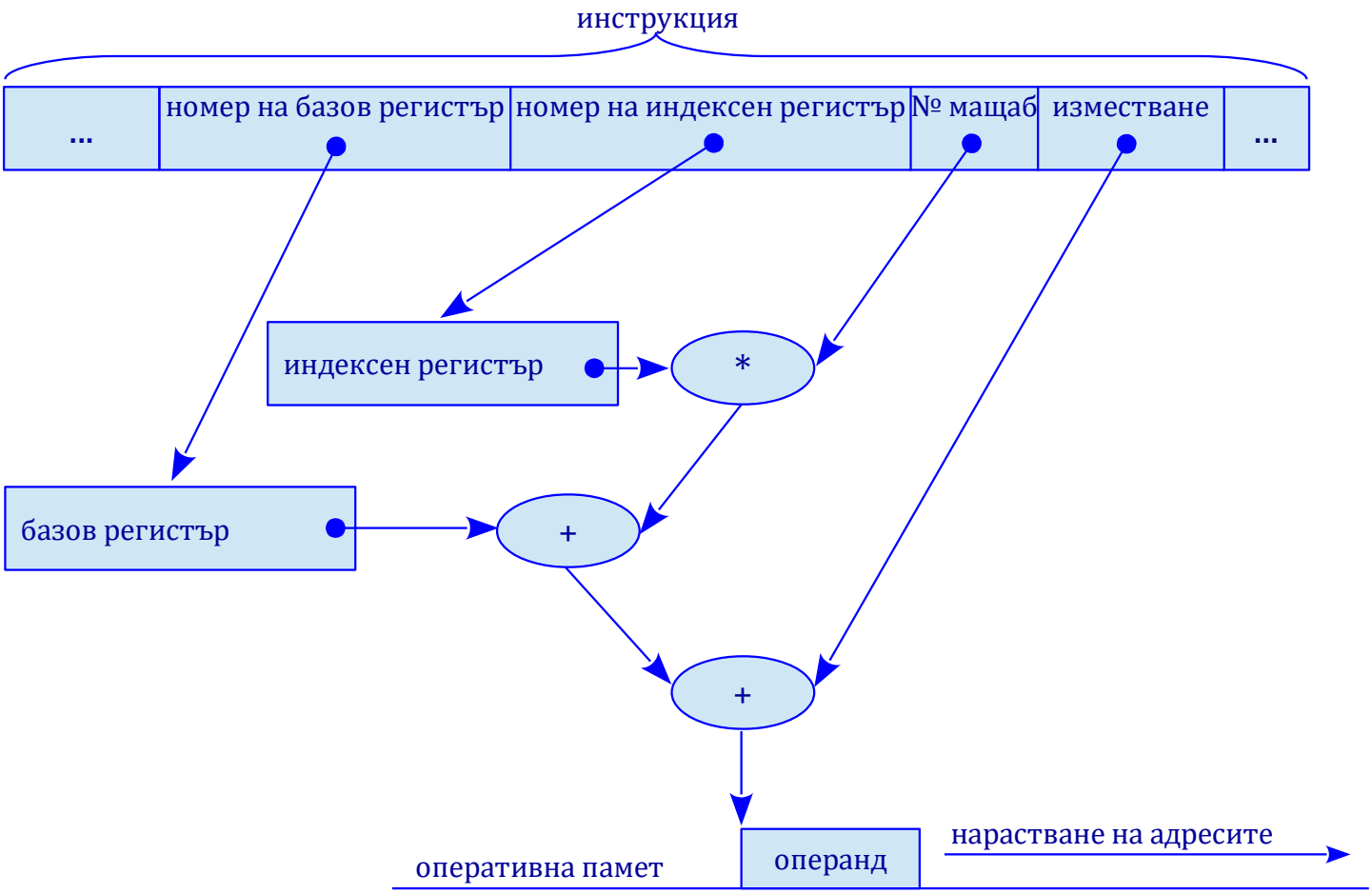
> ***Пример*** *(първият операнд)*
>
> ```
> mov arrLabel5[8*rsi - 8], rax
> ```
>
> *Тази команда записва стойността на регистъра* **`rax`** *в паметта на адрес* `arrLabel5+ 8 * rsi - 8`.

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

8. Базово-индексна адресация с мащабиране и изместване

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

Схема на базово-индексна с мащабиране и изместване адресация



Изчисляването на адреса на операнда може да става само *по време на изпълнението* на инструкцията.

В инструкцията се съдържат като самостоятелни полета номерът на базовия регистър, изместването, номерът на индексния регистър и мащабът. Възможните мащаби са 1, 2, 4 и 8, точно както и при индексната адресация с мащабиране и изместване и по същата причина за ефективност на умножението.

> ***Пример*** *(първият операнд)*
>
> ```
> mov [rdi + 4*rsi - 12], eax
> ```
>
> *Тази команда записва стойността на регистъра* ***eax*** *в паметта на адрес* ***rdi + 4 * rsi - 12****.*

Базово-индексната адресация с мащабиране и изместване 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 повечето числа).