[P220]

Увод у архитектуру рачунара

11

Саша Малков
Универзитет у Београду
Математички факултет
2013/2014

[P271]

Увод у архитектуру рачунара

Саша Малков

Тема 12

Систем прекида (наставак)

Обрада прекида (*Intel x86, Protected mode*)

- Прекиди се означавају бројевима 0-255
- Адресе опслужилаца се налазе у таблици дескиптора прекида (*interrupt descriptor table – IDT*)
  - Ставке таблице су величине по 8 бајтова и представљају 64-битну адресу опслужиоца
  - Таблица се може налазити било где у меморији, а њена адреса мора бити уписана у регистар процесора *IDTR*
    - 48-битни регистар
    - 32-битна адреса положаја таблице
    - 16-битна величина таблице у бајтовима
      - има смисла само ако је таблица мања од 256 ставки
  - Постоје посебне инструкције за пуњење и чување таблице
- Број прекида се користи као индекс таблице

Обрада прекида (*Intel x86, Real mode*)

- *IDT* је увек на адреси 0
- Ставке таблице су величите 32 бита
  - 16 бита за адресу у оквиру сегмента
  - 16 бита за сегмент кода (*CS*)

Обрада прекида (*Intel x86, Real mode*)

- При наступању прекида
  - регистар заставица се ставља на стек
  - поништавају се заставице омогућавања прекида и замки
  - регистри *CS* и *IP* (или *EIP*) се стављају на стек
    - они описују адресу инструкције од које ће се касније наставити извршавање
    - нешто је другачије понашање у случају грешака (*fault*)
  - регистри *CS* и *IP* (или *EIP*) се постављају на основу вредности из *IDT*
    - *CS* = [*IDT* + *intType* * 4 + 2]
    - *IP* = [*IDT* + *intType* * 4]

Универзитет у Београду - Математички факултет

Обрада прекида (*Intel x86, Real mode*)

- Потпрограми који опслужују прекиде
  - ако је потребно омогућити опслуживање других прекида током обраде
    - постављају заставицу омогућавања прекида инструкцијом *sti*
  - чувају стање регистара
  - раде одговарајући посао
  - рестаурирају стање регистара
  - заврше рад инструкцијом *iret*
    - рестаурира са стека вредности регистара CS и IP (или EIP)
    - рестаурира заставице са стека

Универзитет у Београду - Математички факултет

Пример софтверског прекида

- Софтверкси прекиди се праве инструкцијом
  int N
- На пример, инструкција *int 21H* прави прекид који позива одговарајућу функцију ОС-а *DOS*,
  - регистар А (AL, AH) обично описује операцију
- На пример, *AL*=06*H* означава непосредан конзолни У/И
  - ако је *DL = FFH*
    - проверава се да ли на улазу постоји неки карактер
    - ако постоји, уписује се у регистар *AL*, поставља се *ZF=0*
    - ако не постоји, онда је *ZF = 1*
  - иначе
    - на екрану се исписује карактер са кодом *DL*

Изузеци (*Intel x86*)

- Деле се на
  - грешке (*fault*)
    - грешке се јављају при стању које је претходило извршавању инструкције која је произвела грешку
    - након обраде прекида, инструкција ће бити поновљена
    - пример: промашај странице или сегмента
  - замке (*trap*)
    - замке се јављају са условима након извршавања инструкције која је произвела грешку
    - након обраде прекида, биће извршена наредна инструкција
    - пример: прекорачење, кориснички прекиди
  - заустављања (*abort*)
    - пријављују озбиљне проблеме, могуће је да не може ни бити препозната инструкција која је произвела прекид
    - једино што опслужилац може да уради јесте да прекине процес
    - пример: хардверске неисправности, неконзистентне вредности системских таблица и сл.

Примери изузетака

- Неки изузеци су унапред дефинисани
  - 0 – грешка дељења
  - 1 – један-корак
    - ако је постављена заставица замке (*TF=1*), после сваке извршене инструкције се прави ова замка
    - користи се за дебаговање
  - 2 – тачка прекида
    - тачка прекида у коду, за дебаговање
  - 3 – прекорачење

Хардверски прекиди (*Intel x86*)

- Немаскирајући прекид се изазива довођењем сигнала на ножицу *NMI* процесора
  - процесор увек одмах реагује на овај прекид
  - он се не може софтверски маскирати
  - његов тип (број) је 2
- Сви остали прекиди су маскирајући
  - изазивају се довођењем сигнала на ножицу *INTR* процесора
  - процесор реагује само ако је заставица *IF* = 1 (*interrupt enable flag*)
  - значи, може се софтверски маскирати

Препознавање типа прекида

- Уређај поставља сигнал *INTR*
- Процесор реагује започињањем секвенце прихватања прекида
  - као део те секвенце, шаље сигнал *INTA* уређају
- Уређај реагује на сигнал *INTA* постављањем идентификације (броја, типа) прекида на магистралу података
  - у случају фамилије *x86* тип прекида је 8-битни број

Универзитет у Београду - Математички факултет

Контролери прекида

- Ако је више уређаја у прилици да изазива прекиде, постоји могућност да више прекида настане у исто време
- Проблем је у томе што се сви прекиди пријављују на исти начин и путем истих сигнала
- То се решава помоћу додатног уређаја – контролера прекида
- Контролер прекида служи као посредник између уређаја и процесора при изазивању прекида

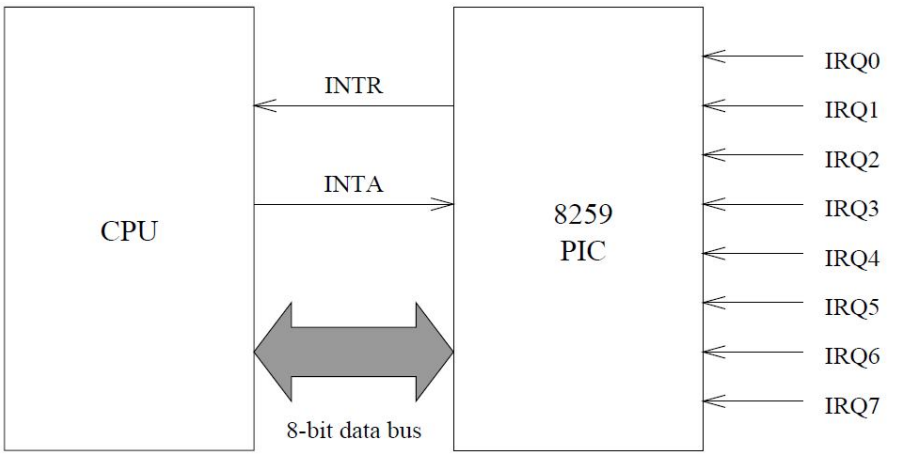
Универзитет у Београду - Математички факултет

Програмибилни контролер прекида *PIC* 8259

- Пример контролера прекида је чип *Intel PIC 8259*
- Може да опслужује до 8 уређаја
  - сваки уређај има по једну линију према контролеру, којом покушава да изазове прекид
  - са процесором се везује преко линија *INTR* и *INTA* и магистрале података
- У случају да више уређаја истовремено затражи прекид, контролер серијализује те прекиде на основу приоритета

Универзитет у Београду - Математички факултет

PIC 8259



Универзитет у Београду - Математички факултет

Програмирање *PIC 8259*

- *PIC* 8259 има два регистра
  - 8-битни командни регистар (*ICR*)
    - користи се за програмирање контролера
    - подразумевани приоритети су од навишег: 0,1,...,7
  - 8-битни регистар маске (*IMR*)
    - користи се за допуштање и онемогућавање појединачних прекида
- Програмирање се остварује комуникацијом процесора са контролером
  - путем магистрале података и адреса:
    - 20H – за регистар *ICR*
    - 21H – за регистар *IMR*
- Типови прекида се додељују одређивањем најнижег типа
  - то је тип прекида 0
  - остали типови су у распону +1 до +7

Универзитет у Београду - Математички факултет

Маскирање прекида и *PIC 8259*

- Блокирање свих прекида се остварује постављањем заставице процесора *IF=0*
- Блокирање појединачних прекида се остварује постављањем регистра *IMR* контролера
  - бит 0 – прекид је омогућен
  - бит 1 – прекид је онемогућен
- На пример, омогућавање само прекида 0 се постиже инструкцијама:

```
mov   AL, 0FEH
out   21H, AL
```

Универзитет у Београду - Математички факултет

Опслуживање прекида и *PIC 8259*

- Сваки пут при завршетку обраде прекида који су изазвани посредством контролера, неопходно је јавити контролеру да је обрада завршена
- То се ради посредством командног регистра, слањем кода *20H* (назива се *EOI – end of interrupt*), који означава крај обраде
- Након тога контролер може да настави са изазивањем нових прекида
- Пример:

```
mov   AL, 20H
out   20H, AL
iret
```


[P271]

Увод у архитектуру рачунара

Саша Малков

Тема 13

Процесори
-
Архитектура инструкција

Архитектура скупа инструкција

- Један од основних аспеката архитектуре процесора је скуп инструкција
- Архитектура скупа инструкција има неколико основних аспеката:
  - пројектовање скупа инструкција
  - имплементација
  - перформансе

Универзитет у Београду - Математички факултет

Број и сложеност инструкција

- По броју и сложености инструкција процесори се деле у три групе:
  - *CISC* – процесори са сложеним скупом инструкција (енгл. *complex instruction set computing*)
  - *RISC* – процесори са редукованим скупом инструкција (енгл. *reduced instruction set computing*)
  - векторски процесори

Универзитет у Београду - Математички факултет

CISC процесори

- Циљеви:
  - сложена архитектура скупа инструкција (*ISA*)
    - кодирање што сложенијих инструкција у што мање меморије
  - разноврсност операција
  - разноврсност начина адресирања

Универзитет у Београду - Математички факултет

CISC процесори

- Последице:
  - нови модели процесора уводили су све више и више нових начина адресирања и нових инструкција
  - подржан превелики број сложених инструкција
    - чак се додају неке иснтрукције које се *никада* не користе при програмирању на асемблеру, али омогућавају ефикасније превођење програма писаних на вишим програмским језицима
  - отежано декодирање инструкција

Универзитет у Београду - Математички факултет

RISC процесори

- Циљеви:
  - једноставна архитектура скупа инструкција
  - обезбеђивање минималног скупа инструкција и начина адресирања
  - повећан број регистара који се могу користити за рачунање

RISC процесори

- Последице:
  - сталнији скуп инструкција
  - једноставно декодирање инструкција
  - скраћивање трајања извршавања операција
    - већина операција у једном или два циклуса
  - једноставнија имплементација процесора

RISC процесори

- Основни принципи дизајна:
  - једноставне операције
  - операције регистар-у-регистар
  - једноставни начини адресирања
    - углавном се користи регистарско адресирање
  - већи број регистара
  - фиксна дужина и једноставан формат инструкција
    - често су инструкције фиксне дужине, поравнате на дужину речи

Универзитет у Београду - Математички факултет

RISC процесори

- Друге одлике:
  - удвајање магистрале – посебно за податке и инструкције
    - тзв. Харвард архитектура
    - најчешће само на нивоу кеша
  - сви регистри су равноправни по могућностима и перформансама
  - преклапање извршавања инструкција
    - извршавање бар једне инструкције по циклусу
  - висока пропусност системске магистрале

Универзитет у Београду - Математички факултет

Однос *RISC* и *CISC* процесора

- *RISC* концепти се развијају од 1975. године
- У актуелном стању технологије производње процесора (*CISC*):
  - било је скупо подржавати сложене операције на већем броју регистара, па је зато постојало мало регистара
    - *RISC* приступ омогућава повећавање броја регистара
  - већина инструкција је захтевала сложену имплементацију, па и дугачке циклусе
    - *RISC* приступ поједностављује имплементације и скраћује извршавање инструкција у циклусима
  - сложена имплементација отежава подизање радне фреквенције
    - *RISC* омогућава значајно подизање радне фреквенције

Однос *RISC* и *CISC* процесора

- Анализе програма показују да већина инструкција обавља преношење података између процесора и меморије
- Имплементација сложених инструкција које се ретко извршавају значајно усложњава имплементацију процесора а доноси скромне добитке у перформансама

Однос *RISC* и *CISC* процесора

- Од 1975. па до средине 1990. постоји тенденција да се произвођачи процесора приклањају *RISC* концептима
- Касније постепено усавршавање производних технологија смањује значај *RISC* архитектуре
- Долази до спајања елемената архитектура

Однос *RISC* и *CISC* процесора

- Данас су уобичајене архитектуре које се одликују *CISC* односом према скупу инструкција, а имају већину осталих одлика *RISC* архитектура:
  - велики број регистара, који су практично равноправни
  - преклапање извршавања инструкција
  - напредне архитектуре кеш меморија

Однос *RISC* и *CISC* процесора

Characteristic	CISC		RISC
	VAX 11/780	Intel 486	MIPS R4000
Number of instructions	303	235	94
Addressing modes	22	11	1
Instructions size (bytes)	2–57	1–12	4
Number of general-purpose registers	16	8	32

Пример кода

- У случају *VAX*-а, једна инструкција је могла да
  - прочита податак из меморије, сабере га са вредношћу регистра, упише назад у меморију и повећа вредност показивача:

    (R2) = (R2) + R3; R2 = R2 + 1
- У случају *RISC* процесора, за то су потребне 4 инструкције:

  R4 = (R2)
  R4 = R4 + R3
  (R2) = R4
  R2 = R2 + 1

Векторски процесори

- Векторски процесори су процесори који оперишу на низовима података
  - инструкције су пројектоване тако да раде са низовима података
  - може се рећи да су низови података елементарни облик података векторских процесора
  - представљају контраст тзв. скаларним процесорима, који раде са појединачним подацима

Универзитет у Београду - Математички факултет

Број адреса у инструкцијама

- Бинарне операције захтевају два, а унарне један аргумент
- Операције најчешће имају један излаз, али их може бити и више
  - на пример, дељење даје количник и остатак
- Уобичајена бинарна операција захтева три адресе:
  - две адресе аргумената
  - једну адресу резултата

Универзитет у Београду - Математички факултет

Број адреса у инструкцијама

- Постоје процесори:
  - са 3 адресе
  - са 2 адресе
  - са 1 адресом
  - без адреса
- Процесор који подржава неки број адреса, обично може да подржи и инструкције са мањим бројем адреса

Универзитет у Београду - Математички факултет

Процесори са 3 адресе

- “Троадресни” процесори експлицитно адресирају два аргумента и резултат операције
- Већина савремених процесора је троадресна

Универзитет у Београду - Математички факултет

Примери троадресних инструкција

add dest, src1, src2	Сабирају се вредности адресиране са *src1* и *src2* и резултат се уписује у *dest*
sub dest, src1, src2	Одузимају се вредности адресиране са *src1* и *src2* и резултат се уписује у *dest*
mult dest, src1, src2	Множе се вредности адресиране са *src1* и *src2* и резултат се уписује у *dest*

Универзитет у Београду - Математички факултет

Пример кода

- На троадресном процесору израз:

  A = B + C * D – E + F + A

- може да се имплементира као:

```
mult   T,C,D      ; T = C*D
add    T,T,B      ; T = B + C*D
sub    T,T,E      ; T = B + C*D – E
add    T,T,F      ; T = B + C*D – E + F
add    A,T,A      ; A = B + C*D – E + F + A
```

Универзитет у Београду - Математички факултет

Карактеристике

- Из примера се виде неке карактеристике:
  - у пракси већина инструкција садржи поновљену једну од адреса аргумената као адресу резултата
  - скуп инструкција одговара операцијама које процесор може да извршава

Универзитет у Београду - Математички факултет

Процесори са 2 адресе

- "Двоадресни" процесори експлицитно адресирају један аргумент и резултат операције
- Мотивација потиче из чињенице да се релативно ретко употребљавају три различите адресе
- Процесори фамилије *Intel x86* су двоадресни

Универзитет у Београду - Математички факултет

Примери двоадресних инструкција

load dest, src	Садржај податка адресираног са *src* се преписује у *dest*
add dest, src	Сабирају се вредности адресиране са *dest* и *src* и резултат се уписује у *dest*
sub dest, src	Одузима се вредност адресирана са *src* од вредности адресиране са *dest* и резултат се уписује у *dest*
mult dest, src	Множе се вредности адресиране са *src* и *dest* и резултат се уписује у *dest*

Универзитет у Београду - Математички факултет

Пример кода

- На двоадресном процесору израз:
  A = B + C * D – E + F + A
- може да се имплементира као:

```
load   T,C        ; T = C
mult   T,D        ; T = C*D
add    T,B        ; T = B + C*D
sub    T,E        ; T = B + C*D – E
add    T,F                ; T = B + C*D – E + F
add    A,T        ; A = B + C*D – E + F + A
```

Универзитет у Београду - Математички факултет

Карактеристике

- Из примера се виде неке карактеристике:
  - у пракси већина инструкција понавља једну исту циљну адресу
  - скуп инструкција одговара операцијама које процесор може да извршава
  - додаје се инструкција за преписивање података

Универзитет у Београду - Математички факултет

Процесори са 1 адресом

- "Једноадресни" процесори експлицитно адресирају један аргумент
- Резултат се увек уписује у *акумулатор*
  - акумулатор је (углавном) једини регистар на коме могу да се извршавају операције
- Мотивација потиче из чињенице да се за већину операција употребљава понављање адресе циља

Универзитет у Београду - Математички факултет

Примери једноадресних инструкција

load addr	Садржај податка адресираног са *addr* се преписује у акумулатор
add addr	Сабирају се вредност акумулатора и вредност адресирана са *addr* и резултат се уписује у акумулатор
sub addr	Одузима се од вредности акумулатора вредност адресирана са *addr* и резултат се уписује у акумулатор
mult addr	Множе се вредност акумулатора и вредност адресирана са *addr* и резултат се уписује у акумулатор
store addr	Вредност акумулатора се преписује на адресу *addr*

Универзитет у Београду - Математички факултет

Пример кода

- На једноадресном процесору израз:
  A = B + C * D – E + F + A
- може да се имплементира као:

```
load   C      ; acc = C
mult   D      ; acc = C*D
add    B      ; acc = B + C*D
sub    E      ; acc = B + C*D – E
add    F      ; acc = B + C*D – E + F
add    A      ; acc = B + C*D – E + F + A
store  A      ; A = B + C*D – E + F + A
```

Универзитет у Београду - Математички факултет

Карактеристике

- Из примера се виде неке карактеристике:
  - редукован број регистара
    - само један има пуну оперативну функционалност
  - скуп инструкција одговара операцијама које процесор може да извршава
  - додају се инструкције за преписивање података у акумулатор и из акумулатора

Универзитет у Београду - Математички факултет

Процесори са 0 адреса

- "Безадресни" процесори не адресирају ниједан аргумент експлицитно
  - осим у посебним инструкцијама које стављају податке на стек и узимају податке са стека
- И аргументи и резултат се увек налазе на стеку
- Мотивација потиче из чињенице да је за већину операција потребно релативно мало података

Универзитет у Београду - Математички факултет

Примери безадресних инструкција

push addr	Садржај података адресираног са *addr* се ставља на врх стека
pop addr	Податак са врха стека се склања са стека и уписује на адесу *addr*
add	Два податка са врха стека се склањају са стека и сабирају. Резултат се ставља на врх стека.
sub	Два податка са врха стека се склањају са стека и одузимају. Резултат се ставља на врх стека.
mult	Два податка са врха стека се склањају са стека и множе. Резултат се ставља на врх стека.

Универзитет у Београду - Математички факултет

Пример кода

- На безадресном процесору израз:

  A = B + C * D – E + F + A

- може да се имплементира као:

```
push   E        ; <E>
push   C        ; <C, E>
push   D        ; <D, C, E>
mult            ; <D*C, E>
push   B        ; <B, D*C, E>
add             ; <B + D*C, E>
sub             ; <B + D*C – E>
push   F        ; <F, B + D*C – E>
add             ; <B + D*C – E + F>
push   A        ; <A, B + D*C – E + F>
add             ; <B + D*C – E + F + A>
pop    A        ; <>
```

Универзитет у Београду - Математички факултет

Карактеристике

- Из примера се виде неке карактеристике:
  - не постоје именовани регистри
  - скуп инструкција одговара операцијама које процесор може да извршава
  - додају се инструкције за преписивање података на стек и са стека

Универзитет у Београду - Математички факултет

Имплементација

- Обично се имплементирају тако да се неколико последњих података на стеку налази у тзв. *стек регистрима* процесора
  - број стек регистара се назива *дубина стека*
  - на тај начин се значајно убрзавају операције са стеком, зато што није потребно приступати меморији

Универзитет у Београду - Математички факултет

Поређење начина адресирања

- Сваки од представљених приступа има предности и мане
- Што се више адреса наводи у инструкцијама
  - број приступа меморији је већи
  - запис инструкција је већи
  - програми се састоје од мање инструкција

Пример процене ефикасности

- Троадресни рачунар:
  - свака инструкција захтева 4 приступа меморији
    - један за инструкцију, два за податке, један за резултат
  - пет инструкција
    - укупно 20 приступа меморији
- Двоадресни рачунар:
  - свака операција и даље захтева 4 приступа меморији
    - један за инструкцију, два за податке, један за резултат
    - инструкција *load* захтева три приступа
  - пет операција и једно преписивање
    - укупно 23 приступа меморији

Пример процене ефикасности

- Једноадресни рачунар:
  - свака операција захтева 2 приступа меморији
    - један за инструкцију и један за податке
    - акумулатор је регистар, а не меморија
    - инструкција *load* захтева два приступа
  - седам инструкција
    - укупно 14 приступа меморији
- Безадресни рачунар:
  - свака операција захтева 1 приступ меморији
    - један за инструкцију
    - претпостављамо да је стек довољно дубок да је пример стао у стек регистре
    - инструкције *push* и *pop* захтевају по два приступа
  - пет операција по 1 и седам инструкција *push* и *pop*
    - укупно 19 приступа меморији

Универзитет у Београду - Математички факултет

Пример процене ефикасности

- Пример сугерише да је у претходном примеру једноадресни приступ најефикаснији, међутим:
  - поређење једноадресног и безадресног рачунара је фер, зато што се у оба случаја претпоставља постојање регистара
  - са друге стране, и неки од адресираних података у случају дво- и троадресних рачунара могу бити регистри
- Ако претпоставимо да дво- и троадресни рачунар имају на располагању један регистар *T*, онда се однос мења:
  - двоадресни има 13 приступа меморији
  - троадресни има свега 12 приступа меморији

Универзитет у Београду - Математички факултет

Пример процене ефикасности

- У претходним примерима није узета у обзир величина инструкција:
  - што се више адреса наводи, то је инструкција већа
  - величина није увек једноставно предвидива

Ограничавање врста адреса

- Постоји већи број различитих начина адресирања података
- Код *RISC* процесора је уобичајено да се у већини инструкција могу адресирати само различити регистри процесора
- Процесор *Intel Pentium* је двоадресни, али највише један од операнада сме бити у меморији

Архитектура *load / store*

- Концепт:
  - све операције се извршавају искључиво над регистрима процесора
  - само операције *load* и *store* могу да приступају меморији

Архитектура *load / store* (2)

- *RISC* и векторски процесори често користе овакву архитектуру
  - значајно се смањује величина инструкција
  - значајно се редукује сложеност декорирања и имплементирања инструкција
  - омогућава се висок степен преклапања инструкција
    - дужина извршавања није непосредно пропорционална броју инструкција и приступа меморији

Пример кода

- На троадресном проц. са арх. *load / store* израз:
  A = B + C * D – E + F + A
- може да се имплементира као:

```
load    R1, B          ; R1 = B
load    R2, C          ; R2 = C
load    R3, D          ; R3 = D
load    R4, E          ; R4 = E
load    R5, F          ; R5 = F
load    R6, A          ; R6 = A
mult    R2, R2, R3     ; R2 = C*D
add     R2, R2, R1     ; R2 = B + C*D
sub     R2, R2, R4     ; R2 = B + C*D – E
add     R2, R2, R5     ; R2 = B + C*D – E + F
add     R2, R2, R6     ; R2 = B + C*D – E + F + A
store   A, R6
```


Архитектура регистара

- Регистри процесора служе за чување
  - података
  - инструкција
  - стања процесора
- Регистри се деле на
  - регистре опште намене и
  - посебне регистре (или регистре посебне намене)
    - посебни регистри доступни корисничким програмима
    - посебни регистри резервисани за системске потребе

Архитектура регистара опште намене

- Број и врста регистара опште намене су обично повезани са архитектуром адресирања
  - безадресни процесори не захтевају регистре опште намене
    - мада имају имплицитне стек-регистре
  - код дво- и троадресних процесора регистри опште намене нису неопходни
    - уводе се због подизања перформанси
  - *RISC* процесори по правилу имају већи број регистара опште намене

Архитектура регистара посебне намене

- Пример регистара посебне намене су:
  - регистри за вођење стека
  - бројач инструкција
  - интерни регистар инструкције (који садржи текућу инструкцију)

Контрола тока програма

- Бројач инструкција (или *програмски бројач*) има улогу контролора тока
  - садржи адресу наредне инструкције
  - чим се инструкција прочита, бројач се повећава тако да показује на наредну инструкцију
  - код архитектура са фиксном величином инструкције, увек се увећава за фиксан број
    - нпр. код процесора *MIPS* и *SPARC*, све инструкције су 32-бита
- Програм се у начелу извршава секвенцијално
- Секвенцијално извршавање се по потреби може изменити
  - гранањем и
  - петљама

Универзитет у Београду - Математички факултет

Гранање

- Гранање се имплементира инструкцијама гранања
  - Оне експлицитно мењају вредност бројача инструкција
- Постоје две врсте инструкција гранања:
  - безусловне (или *експлицитне*) и
  - условне
- Гранање може бити
  - тренутно или
  - одложено

Универзитет у Београду - Математички факултет

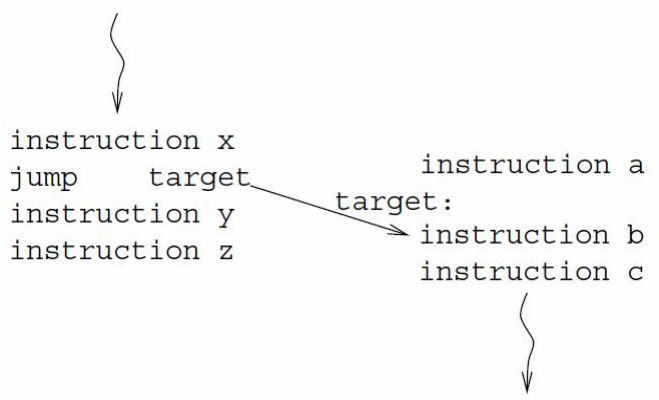
Начин навођења нове адресе

- Нова адреса извршавања се наводи као
  - апсолутна
    - наводи се пуна нова адреса
    - подржавају практично сви процесори
  - релативна
    - наводи се разлика између нове адресе и текуће адресе
    - не подржавају сви процесори
    - предност је у померљивости кода
      - без обзира на локацију у меморији гранање је једнако исправно
    - ако разлика није велика, може имати краћи запис него навођење апсолутне адресе

Безусловно гранање

- Безусловно гранање је експлицитна и безусловна промена тока извршавања програма

Безусловно гранање - пример



Условно гранање

- Условно гранање је експлицитна промена тока извршавања програма у случају важења неког наведеног услова
- Постоје две основне врсте условног гранања:
  - постави-па-скочи и
  - провери-и-скочи

Гранање постави-па-скочи

- Основна идеја је да се раздвоје проверавање услова и гранање
  - најпре се посебним инструкцијама проверају услови или другачије поставља одговарајуће стање процесора
  - затим се инструкцијама гранања само проверава стање процесора и по потреби извршава промена бројача инструкција
- Гранање постави-па-скочи је примењено код фамилије процесора *Intel x86*

Пример – постави-па-скочи

- Пример у случају процесора *Intel Pentium*

```
    cmp AX,BX      ; поређење вредности AX и BX
    je target      ; ако су једнаке, контрола се преноси на target
    sub AX,BX      ; иначе се наставља од ове инструкције
    . . .
target:
    add AX,BX      ; у случају једнакости се наставља одавде
```

Гранање провери-и-скочи

- Основна идеја је да једна иста инструкција проверава услов и промени адресу
- Овакав вид гранања примењен је код већег броја процесора, укључујући и *MIPS*

Пример – провери-и-скочи

- Пример у случају процесора *MIPS*
  - наредна инструкција пореди вредности регистара *$t0* и *$t1* и прелази на дату адресу *target* ако су вредности једнаке

```
beq $t1,$t0,target
```

Регистри стања

- Да би било могуће имплементирати гранање постави-па-скочи, неопходно је да процесор има *регистар стања* који чува резултате поређења и других операција
  - овакви регистри се називају и *регистри кодова поређења*
- Овакве регистре
  - имају *Intel x86*, *SPARC*, *PowerPC*
  - нема *MIPS*

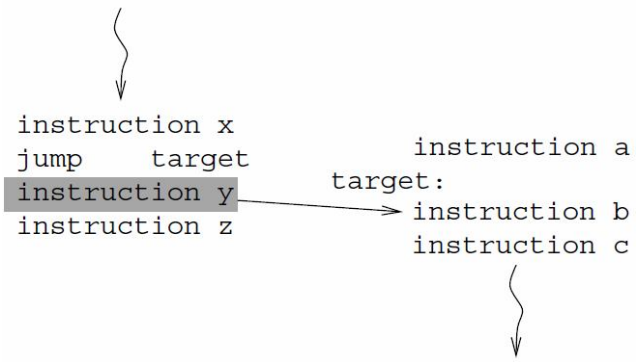
Универзитет у Београду - Математички факултет

Тренутно и одложено гранање

- Гранање је *тренутно* ако се нова адреса поставља практично током извршавања инструкције гранања
  - сви до сада представљени примери се односе на тренутно гранање
- Гранање може бити и *одложено*, ако се нова адреса поставља тек по извршавању наредне инструкције програма
- Одложено гранање користи процесор *SPARC*

Универзитет у Београду - Математички факултет

Илустрација одложеног гранања



Одложено гранање

- Смисао одложеног гранања се види тек у контексту преклапања извршавања инструкција
  - у време извршавања гранања наредна инструкција је већ прочитана и декодирана
  - ако се не би извршила, то време би било изгубљено
  - штавише, ако се не изврши, у то време се не може извршити ништа друго, зато што инструкција која је на новој адреси још није спремна за извршавање

Литература

- *Sivarama Dandamudi*, ***Fundamentals of Computer Organization and Design***, *Springer, 2002.*
- *Andrew Tanenbaum*, ***Архитектура и организација рачунара***, *Микро књига, 2007.*
- *Ненад Митић*, ***Основи рачунарских система***, *Математички факултет, 2002.*

Универзитет у Београду - Математички факултет