[P220]
Увод у архитектуру рачунара
12

Саша Малков
Универзитет у Београду
Математички факултет
2013/2014

[P271]
Увод у архитектуру рачунара
Саша Малков

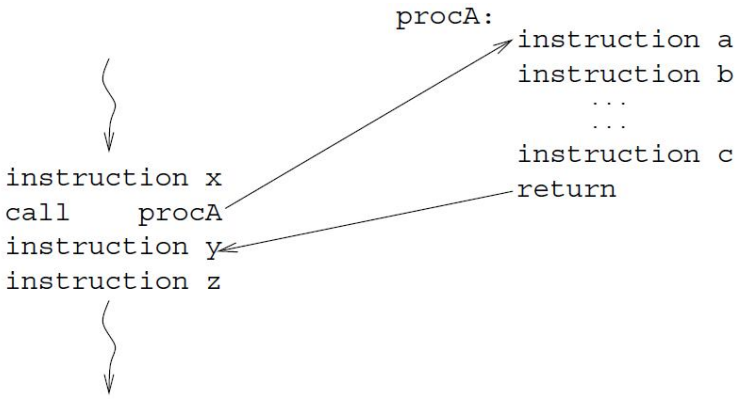
Тема 13
Процесори - Архитектура инструкција
(наставак)

Позивање процедура

- Гранања су једносмерне промене тока извршавања
  - дејство је ограничено на једну промену тока извршавања
- Позивања процедура су двосмерне промене
  - након иницијалног позивања, касније следи повратак на место одакле је извршено позивање
- Да би повратак био могућ неопходне су две ствари:
  - експлицитна ознака краја процедуре
    - за то служи инструкција *return*
      - *Intel Pentium* има инструкцију *ret*
      - *MIPS* има инструкцију *jr*
  - адреса повратка
    - мора бити сачувана при позивању процедуре

Универзитет у Београду - Математички факултет

Пример позивања процедуре



Универзитет у Београду - Математички факултет

Чување адресе повратка

- Може се чувати
  - у регистру
    - намењеном за то
      - шта је са рекурзијом или другим позивањима?
    - било ком регистру
      - на пример *MIPS*
  - на стеку
    - нпр. *Intel x86*
- Чува се
  - адреса инструкције позивања
    - *SPARC*
  - адреса прве инструкције након инструкције позивања
    - већина процесора, укључујући *Intel x86*, *MIPS*

Универзитет у Београду - Математички факултет

Чување адресе повратка у регистрима

- Може се користити
  - регистар посебно намењен за то
  - било који регистар
    - на пример *MIPS*
    - пример повратка, уз претпоставку да је адреса повратка у регистру $*ra*

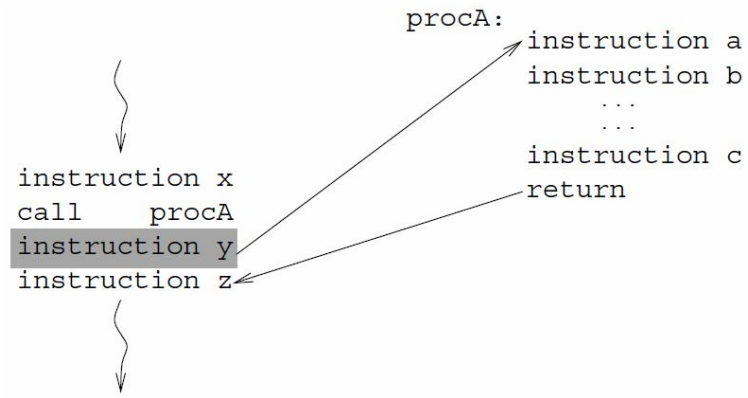
```
jr  $ra
```

- Проблеми
  - шта је са рекурзијом?
  - шта је са позивањем потпроцедура?
  - у оба случаја је неопходно додатно старање о адресама повратка

Универзитет у Београду - Математички факултет

Одложено позивање

- Процесори који подржавају одложено гранање обично подржавају и одложено позивање



Универзитет у Београду - Математички факултет

Преношење параметара

- Две основне технике су:
  - помоћу регистара
    - параметри се записују у регистрима пре позивања
    - овај метод је бржи
    - не омогућава рекурзију
    - често захтева чување претходних вредности регистара
    - коришћен код *RISC* процесора
      - нпр. *MIPS*, *PowerPC*
  - помоћу стека
    - параметри се постављају на стек пре позивања
    - флексибилнији начин
    - мање ефикасан
    - коришћен код *CISC* процесора
      - нпр. *Intel x86*

Универзитет у Београду - Математички факултет

Пројектовање скупа инструкција

- Пројектовање скупа инструкција има неколико важних аспеката:
  - типови операнада
  - начини адресирања
  - типови инструкција
  - формати инструкција

Универзитет у Београду - Математички факултет

Типови операнада

- Уобичајено је да процесори препознају само елементарне типове података
  - *char, int, float*
- Често исте инструкције раде са подацима различите величине, у зависности од начина навођења операнада
  - на пример (*Intel x86*):

```
mov AL, addr    ; преписује 8-битни податак
mov AX, addr    ; преписује 16-битни податак
mov EAX, addr   ; преписује 32-битни податак
```

Универзитет у Београду - Математички факултет

Типови операнада

- У случају *RISC* процесора уобичајено је да се из нотације инструкције препознаје величина операнда, на пример:

```
lb   Rdest, address   ; преписује 8-битни податак (бајт)
lh   Rdest, address   ; преписује 16-битни податак (пола речи)
lw   Rdest, address   ; преписује 32-битни податак (реч)
ld   Rdest, address   ; преписује 64-битни под. (двострука реч)
```


Начини адресирања (1)

- Начини адресирања описују како се одређује операнд инструкције
- Операнди могу бити
  - константе
    - режим непосредног адресирања
  - у регистрима
    - режим регистарског адресирања
  - у меморији
    - режим меморијског адресирања
    - постоји много различитих начина адресирања података у меморији

Начини адресирања (2)

- Сви процесори подржавају бар два основна начина адресирања:
  - Режим непосредног адресирања
    - навођење константне вредности операнда
    - не постоји приступање меморији
      - осим читања инструкције
    - назива се и *непосредно адресирање*
  - Режим регистарског адресирања
    - навођење регистра који садржи вредност операнда
    - не постоји приступање меморији
      - осим читања инструкције
    - назива се и *регистарско адресирање*

Универзитет у Београду - Математички факултет

Начини адресирања (3)

- Разлика између *RISC* и *CISC* процесора је у подржаним начинима адресирања података у меморији
  - *RISC* процесори користе архитектуру *load / store*
    - све инструкције, осим *load* и *store*, подржавају само непосредно и регистарско адресирање
    - подаци који су у меморији могу се адресирати само у оквиру инструкција *load* и *store*
    - број начина адресирања је обично сасвим скроман
  - *CISC* процесори подржавају мноштво начина адресирања
    - убичајено је да све инструкције подржавају адресирање података у меморији
    - број начина адресирања је обично велики

Универзитет у Београду - Математички факултет

Начини адресирања (4)

- Већина *RISC* процесора подржавају:
  - Индиректно регистарско адресирање
    - циљна адреса операнда (*TA*, енгл. *target address*) је дата као вредност регистра:
    - *TA* = [*reg*]
    - вредност операнда је записана у меморији на адреси [*reg*]
  - Адресирање са базним регистром и удаљењем
    - циљна адреса операнда се рачуна као збир вредности базног регистра и датог константног индекса
    - *TA* = [*reg*] + *index*
    - назива се и *индексно реч. адресирање са непосредним индексом*
    - потенцијално се вредност индекса множи величином података
  - Адресирање са базним регистром и удаљењем у регистру
    - циљна адреса операнда се рачуна као збир вредности базног регистра и датог индексног регистра
    - *TA* = [*reg1*] + [*reg2*]
    - назива се и *индексно реч. адресирање са регистарским индексом*
    - потенцијално се вредност индекса множи величином података

Други значајни начини адресирања (1)

- Апсолутно адресирање
  - циљна адреса операнда се непосредно наводи
  - *TA* = *address*
  - вредност операнда је у меморији, на адреси [*address*]
  - назива се и *директно меморијско адресирање*
- Индиректно меморијско адресирање
  - циљна адреса операнда је записана у меморији на датој адреси
  - *TA* = [*address*]
  - назива се и *индиректно адресирање*

Други значајни начини адресирања (2)

- Адресирање са датом базном адресом и удаљењем у регистру
  - циљна адреса операнда се рачуна као збир непосредно дате базне адресе и вредности датог индексног регистра (индекса)
  - *TA* = *base_address* + [*reg*]
  - назива се и *индексно индиректно адресирање са регистарским индексом*
  - потенцијално се вредност индекса (регистра) множи величином података
- Адресирање са базним регистром, индекс регистром и удаљењем
  - циљна адреса операнда се рачуна као збир вредности базног регистра (основа) и збира вредности датог индексног регистра и непосредно наведеног удаљења (овај збир је индекс)
  - *TA* = [*base_reg*] + [*index_reg*] + *address*
  - потенцијално се вредност индекса (регистра) множи величином података

Универзитет у Београду - Математички факултет

Скалирано адресирање

- За адресирање се каже да је *скалирано* ако се један део адресе (индекс) множи величином податка
- Суштина је у ослобађању програмера од узимања величине података у обзир при писању кода
- Скалирање се примењује као опција (или обавеза) при различитим адресирањима са удаљењем (индексним адресирањима)

Универзитет у Београду - Математички факултет

Адресирање са удаљењем

- За адресирање се каже да је *са удаљењем* ако се један део адресе додаје као константа на израчунату адресу
- Користи се за одређивање удаљености податка од почетка сложене структуре података у којој се налази
- Удаљење се примењује као опција (или обавеза) при различитим индексним адресирањима

Универзитет у Београду - Математички факултет

Имплицитно адресирање

- За адресирање се каже да је *имплицитно* ако се адреса операнда израчунава аутоматски, без експлицитног навођења
- Имплицитно адресирање није адресирање у правом смислу речи
- Пример су операције које раде са стеком

Универзитет у Београду - Математички факултет

Адресирање програмског кода

- При адресирању програмског кода примењују се следећи начини адресирања:
  - Апсолутно адресирање
    - циљна адреса (*TA*) се непосредно наводи
      - пример: *jump address*
    - *TA = address*
  - Релативно адресирање
    - наводи се разлика између циљне и текуће адресе
      - пример: *jump offset*
    - *TA = PC + offset*
    - (посебан облик адресирања са базним регистром и удаљењем)
  - Регистарско индиректно адресирање
    - наводи се регистар који садржи циљну адресу
      - пример: *jump reg*
    - *TA = [reg]*
    - (посебан облик адресирања са базним регистром и удаљењем)

Универзитет у Београду - Математички факултет

Врсте инструкција

- Инструкције за премештање података
- Аритметичке и логичке инструкције
- Инструкције за контролу тока
- Улазно / излазне инструкције

Универзитет у Београду - Математички факултет

Инструкције за премештање података

- Подржавају их сви процесори
- Деле се на инструкције које премештају податке:
  - између меморије и регистара
    - посебна подврста за рад са стеком
  - између регистара

Инструкције за премештање података (2)

- Код *RISC* процесора је премештање података између процесора и меморије строго ограничено на инструкције:
  - *load*
  - *store*
  - неки од *RISC* процесора не омогућавају непосредно премештање података између регистара, већ само у оквиру других инструкција (нпр. сабирање), на пример:
    - *add Rdest, Rsource, 0 /* Rdest = Rsource + 0 */*

Инструкције за премештање података (3)

- Код *CISC* процесора се уместо две обично имплементира само једна инструкција за премештање података која равноправно третира регистре и меморију
  - на пример, код *Intel x86*:
    - ***mov dest, src***
    - највише један од аргумената може бити у меморији
    - MOV CX,20
    - MOV CX, [BX]
    - MOV CX, [50000]

Аритметичке инструкције

- Аритметичке инструкције обухватају како целобројне тако и операције у покретном зарезу
- Већина процесора подржава бар 4 основне аритметичке операције
  - сабирање и одузимање захтевају по једну инструкцију
  - множење и дељење захтевају посебне операције за означене и неозначене аргументе
  - неки процесори не подржавају дељење у потпуности
    - потпуна подршка је рачунање количника и остатка
    - *Intel x86, MIPS* пружају пуну подршку
    - *PowerPC, Sparc* рачунају само количник

Логичке инструкције

- Логичке операције подразумевају скуп операција на нивоу битова
  - практично сви процесори подржавају *and* и *or*
  - већина процесора подржава *not* и *xor*

Контролни битови

- Скоро све аритметичке и логичке инструкције постављају контролне битове процесора при свом извршавању
  - називају се и *заставице* или *условни кодови*
- Уобичајени контролни битови су:
  - *S* – бит знака (0=позитиван, 1=негативан)
  - *Z* – бит нуле (0=резултат није нула, 1=резултат је нула)
  - *O* – бит прекорачења (0=нема пр., 1=прекорачење)
  - *C* – бит преноса (0=нема преноса, 1=има преноса)
- Контролни битови се употребљавају
  - као улазни подаци за неке операције (нпр. сабирање са преносом, померање са преносом и сл.)
  - у инструкцијама гранања

Контролни битови (2)

- пример за *Intel x86*:
  cmp count, 25
  je target
- Код неких процесора све А/Л инструкције постављају контролне битове
- Код неких процесора постоје верзије инструкција које постављају и верзије које не постављају контролне битове
  - на пример, процесор *SPARC* има посебне инструкције:
    - *ADD*
    - *ADDcc*

Инструкције за контролу тока

- Инструкције за контролу тока програма су
  - инструкције гранања
  - инструкције за позивање процедура
    - овде спадају и инструкције за враћање из процедура
- (размотрено на претходном часу)

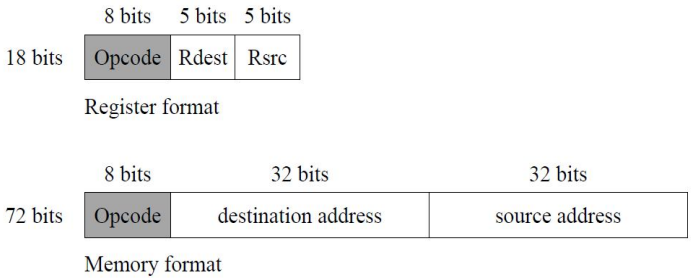
Улазно / излазне инструкције

- Улазно / излазне инструкције се значајно разликују између процесора
- Оне постоје само код процесора који подржавају изоловано пресликавање улаза и излаза
  - ако процесор подржава само меморијско пресликавање У/И, онда нема ове инструкције
- Уобичајене су две инструкције:
  - ***in Reg, io_port***
  - ***out io_port, Reg***
- Величина инструкције зависи од подржане дужине адресе порта

Формат инструкција

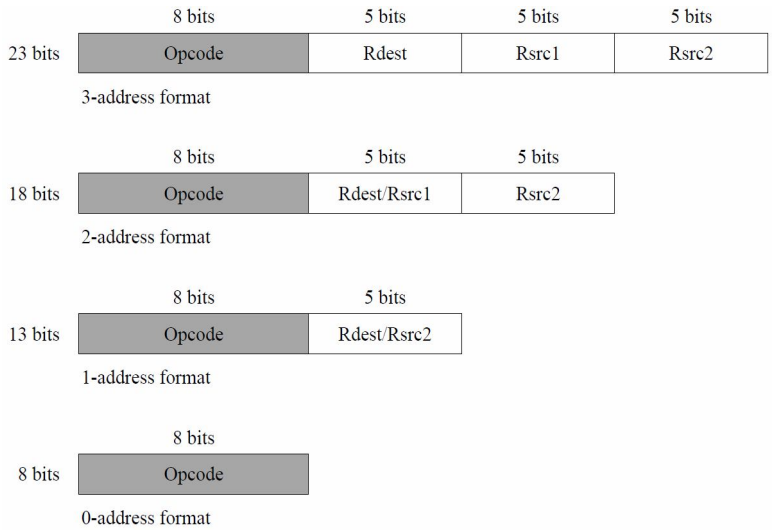
- Формат инструкције подразумева начин кодирања (тј. бинарног записивања) инструкције
- Два основна типа формата инструкција су:
  - формат фиксне дужине инструкција
    - уобичајен за *RISC* процесоре
    - *MIPS*, *PowerPC*, *Sparc* имају иснтрукције дужине 32 бита
  - формат променљиве дужине инструкција
    - уобичајен за *CISC* процесоре
    - нпр. *Intel x86*

Врста аргумената и формат инс.



Универзитет у Београду - Математички факултет

Фиксан формат инструкција



Универзитет у Београду - Математички факултет

[P271]

Увод у архитектуру рачунара

Саша Малков

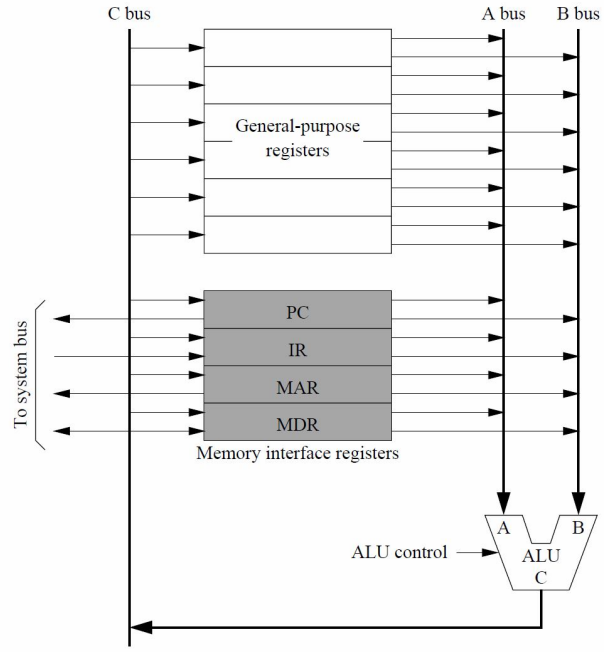
Тема 14

Процесори
-
Имплементација инструкција

Извршавање инструкција

- Да би процесор могао да изврши инструкцију потребно је да:
  - адресира и прочита инструкцију из меморије
  - декодира инструкцију
  - адресира и прочита потребне аргументе
  - изврши одговарајућу операцију
  - адресира и запише израчунат резултат

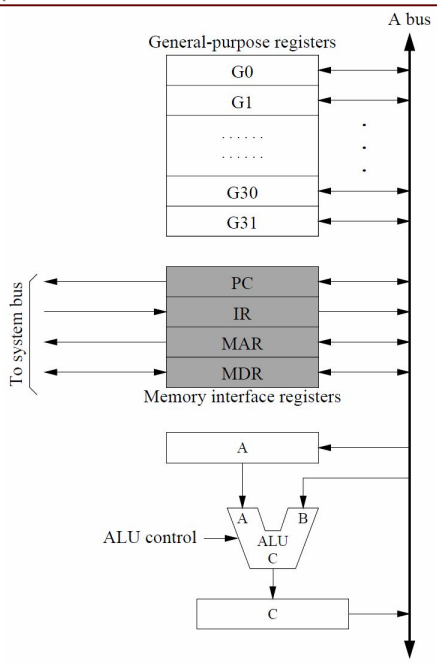
Уопштени пут података



Конкретнији пример

- Претпоставимо да процесор има:
  - јединствену магистралу (A) ширине 32 бита кроз коју пролазе све адресе и подаци у оквиру процесора
  - 32 регистра опште намене (G1-G32)
  - могућност обраде само 32-битних података

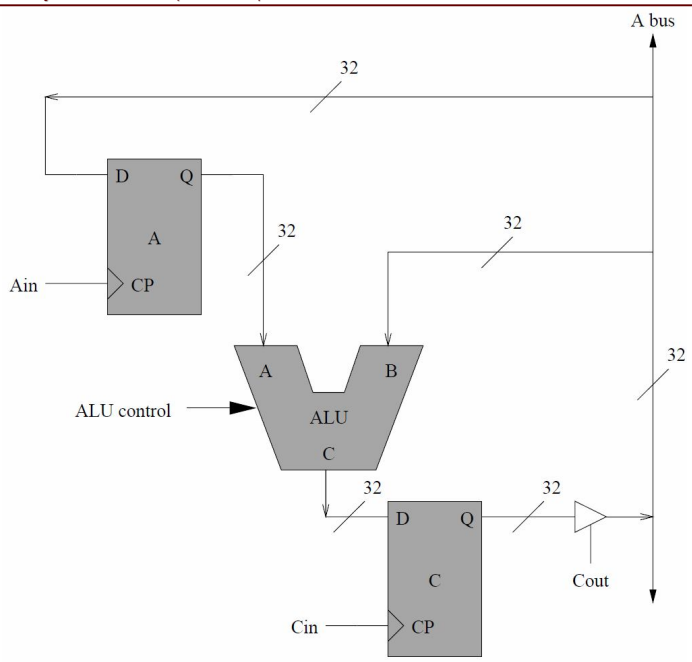
Конкретнији пример пута података



Додатни регистри

- Због тога што постоји јединствена интерна магистрала A, потребни су помоћни регистри:
  - регистар *A* чува вредност првог операнда док се други адресира
    - увек је доступан за читање
  - регистар *C* чува резултат док се не пренесе даље
    - увек је доступан претходни резултат за писање
- Имплементирају се помоћу *D* флип-флопова

Имплементација помоћних регистара



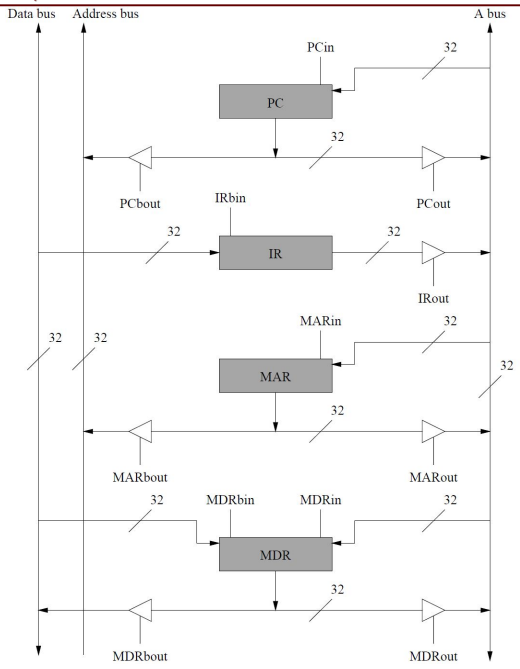
Универзитет у Београду - Математички факултет

Меморијски интерфејс

- Меморијски интерфејс користи четири помоћна регистра
  - ови регистри посредују између системске магистрале и интерне процесорске магистрале A
- Регистар *PC* је бројач инструкција
- Регистар *IR* је регистар инструкције
- Регистар *MAR* је регистар меморијске адресе
- Регистар *MDR* је регистар меморијског податка

Универзитет у Београду - Математички факултет

Имплементација пута података



Универзитет у Београду - Математички факултет

Регистар *PC*

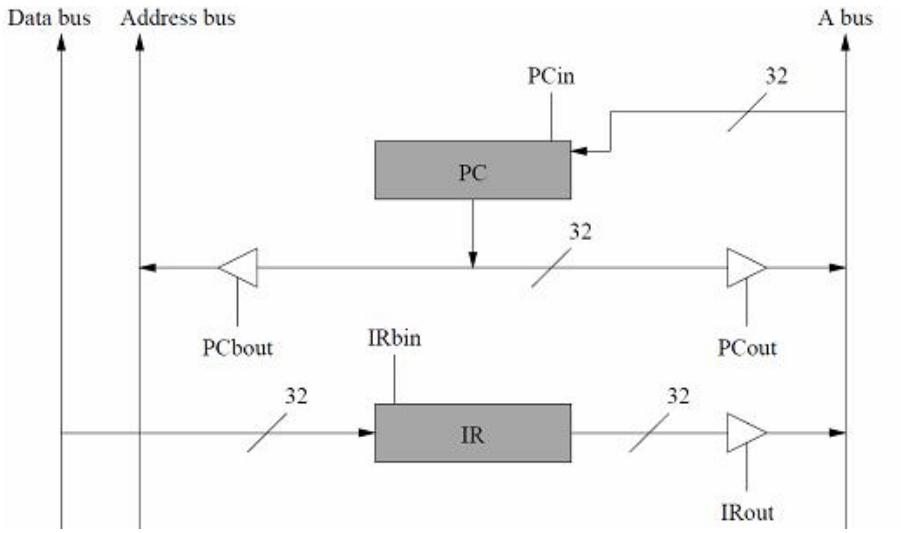
- Бројач инструкција
  - садржи адресу наредне инструкције
    - контролни сигнал *PCin* поставља вредност регистра
  - садржај ставља на системску адресну магистралу ради читања инструкције из меморије
    - контролни сигнал *PCbout*
  - садржај ставља на магистралу A ради омогућавања релативних скокова и позивања процедура
    - контролни сигнал *PCout*
  - може симултано да иде на обе магистрале

Универзитет у Београду - Математички факултет

Регистар *IR*

- Регистар инструкције
  - садржи инструкцију која се тренутно извршава
    - контролни сигнал *IRbin* поставља вредност регистра
    - контролни сигнал *IRout* ставља вредност регистра на магистралу A

Имплементација пута података



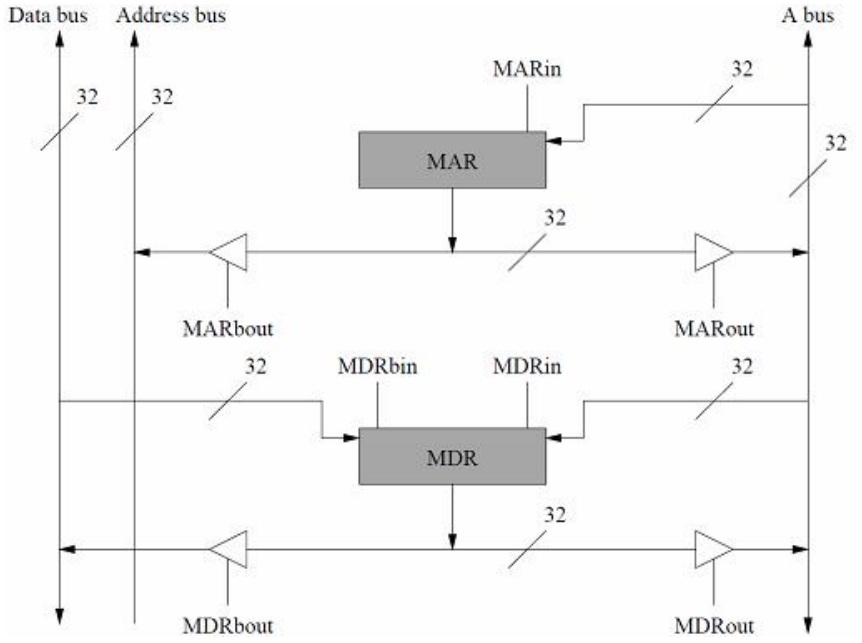
Регистар *MAR*

- Регистар меморијске адресе
  - садржи адресу операнда који је у меморији
  - користи се при адресирању података који су у меморији
  - ради слично регистру *PC:*
    - контролни сигнал *MARin* поставља вредност регистра
    - контролни сигнал *MARout* ставља вредност регистра на магистралу A
    - контролни сигнал *MARbout* ставља вредност регистра на системску адресну магистралу

Регистар *MDR*

- Регистар меморијског податка
  - садржи вредност операнда који је у меморији
  - користи се при читању операнда из меморије
  - адреса операнда је у регистру *MAR*
  - има двосмеран интерфејс*:*
    - контролни сигнал *MDRin* поставља вредност регистра са магистрале A
    - контролни сигнал *MDRbin* поставља вредност регистра са системске магистрале података
    - контролни сигнал *MDRout* ставља вредност регистра на магистралу A
    - контролни сигнал *MDRbout* ставља вредност регистра на системску магистралу података

Имплементација пута података



Регистри опште намене

- Регистри опште намене су везани само на инстерну магистралу A
- Сваки од регистара *Gx* има по два контролна сигнала
  - *Gxin*
  - *Gxout*

Пример инструкције

- Пратимо контролне сигнале на примеру инструкције:
  add %G9, %G5, %G7 / G9 = G5 + G7 */*
- Најпре се садржај једног регистра мора сачувати у помоћном регистру A, а затим сабирати са другим...
  - корак 1: преписујемо *G5* у помоћни регистар *A*
  - корак 2: стављамо *G7* на магистралу *A* и рачунамо
  - корак 3: резултат уписујемо у регистар *G9*
- Нећемо експлицитно наглашавати искључивање сигнала између корака

Корак 1:

- Преписујемо *G5* у помоћни регистар *A*
  - поставља се сигнал *G5out* да би се садржај регистра *G5* ставио на магистралу A
  - поставља се сигнал *Ain* да би се подаци са магистрале A уписали у помоћни регистар *A*

Корак 2:

- Стављамо *G7* на магистралу *A* и рачунамо
  - поставља се сигнал *G7out* да би се садржај регистра *G7* ставио на магистралу A
  - (садржај регистра A је увек доступан АЛ јединици)
  - поставља се сигнал *Cin* да би се резултат уписао у регистар *C*
  - АЛ јединици се налаже да израчуна резултат

Универзитет у Београду - Математички факултет

Корак 3:

- Резултат уписујемо у регистар *G9*
  - поставља се сигнал *G9in* да би се садржај регистра *G9* прочитао са магистрале A
  - поставља се сигнал *Cout* да би се садржај регистра *C* ставио на магистралу A

Универзитет у Београду - Математички факултет

Трајање циклуса

- На основу дужине трајања корака се дефинише трајање циклуса
  - У идеалном случају би сваки корак требало да траје једнако дуго
  - По потреби се може доделити већи број циклуса захтевнијим корацима
    - у претходном примеру су кораци 1 и 3 слични, али корак 2 може бити захтевнији (зависно од операције)

Читање инструкције

- У претходном опису је намерно прескочено представљање читања и декодирања инструкције
- Читање инструкције обухвата:
  - корак 1: адресирање инструкције
  - корак 2: уписивање нове адресе инструкције
  - корак 3: читање инструкције

Корак 1:

- Адресирање инструкције
  - поставља се сигнал *PCbout* да би се садржај регистра *PC* ставио на системску адресну магистралу
  - поставља се сигнал *PCout* да би се садржај регистра *PC* ставио на магистралу A
  - АЛ јединици се даје инструкција *add4* да би сабрала вредност на свом улазу *B* (вредност регистра *PC*) са 4
  - поставља се сигнал *Cin* да би се резултат (*PC+4*) уписао у помоћни регистар *C*

Универзитет у Београду - Математички факултет

Корак 2:

- Уписивање нове адресе инструкције
  - чекамо (бар) један циклус да меморија прочита и достави инструкцију
  - поставља се сигнал *Cout* да би се резултат (*PC+4*) ставио на магистралу A
  - поставља се сигнал *PCin* да би се резултат (*PC+4*) са магистрале A уписао у регистар *PC*

Универзитет у Београду - Математички факултет

Корак 3:

- Читање инструкције
  - поставља се сигнал *IRbin* да би се прочитана инструкција ставила у помоћни регистар *IR*

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.*