Увод у организацију и архитектуру рачунара 1

Александар Картељ

kartelj@matf.bg.ac.rs

Напомена: садржај ових слајдова је преузет од проф. Саше Малкова

Процесор

Архитектура скупа инструкција: CISC и RISC

Архитектура скупа инструкција

- Један од основних аспеката архитектуре процесора је скуп инструкција

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

CISC процесори

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

RISC процесори

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

Однос *RISC* и *CISC* процесора

- *RISC* концепти се развијају од 1975. године
- Проблем са *CISC* процесорима:
  - Сложена имплементација отежава подизање радне фреквенције
    - Радна фреквенција се одређује на основу најспорије операције
  - *RISC* омогућава значајно подизање радне фреквенције
- Данас су уобичајене архитектуре које користе комбинацију CISC и RISC архитектуре

Однос *RISC* и *CISC* процесора (2)

	CISC		RISC
Characteristic	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

Процесор

Број адреса у инструкцијама

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

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

Процесори са 3 адресе

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

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
```

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

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
```

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

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
```

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

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       ; <>
```

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

Процесор

Архитектура *load/store*

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
```

Процесор

Оквирни механизми извршења инструкција

Архитектура регистара

- Регистри се деле на
  - регистре опште намене и
  - посебне регистре (или регистре посебне 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*)
  - садржи адресу наредне инструкције
  - чим се инструкција прочита, бројач се повећава тако да показује на наредну инструкцију
- Програм се у начелу извршава секвенцијално
- Секвенцијално извршавање се по потреби може изменити
  - гранањем и
  - петљама

Гранање

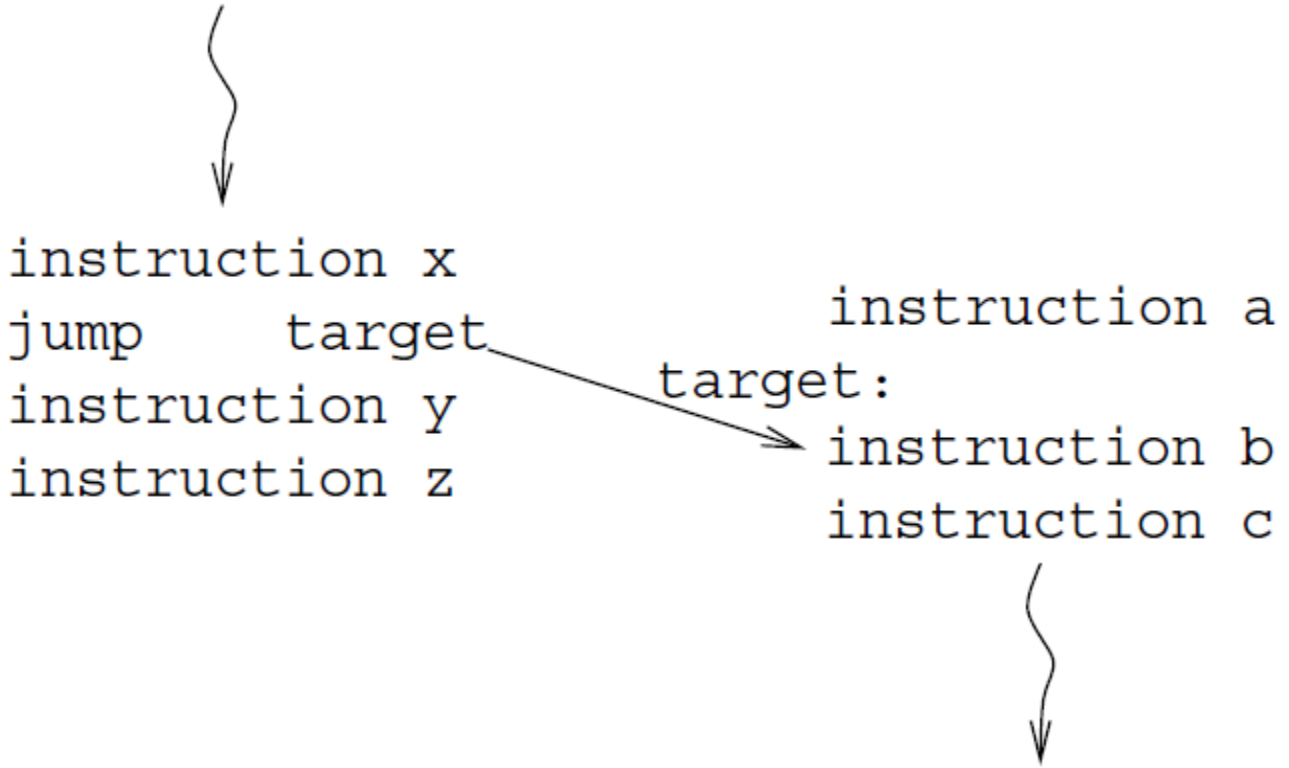
- Гранање се имплементира инструкцијама гранања
  - Оне експлицитно мењају вредност бројача инструкција

- Постоје две врсте инструкција гранања:
  - безусловне (или *експлицитне*) и
  - условне

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

Пример условног гранања

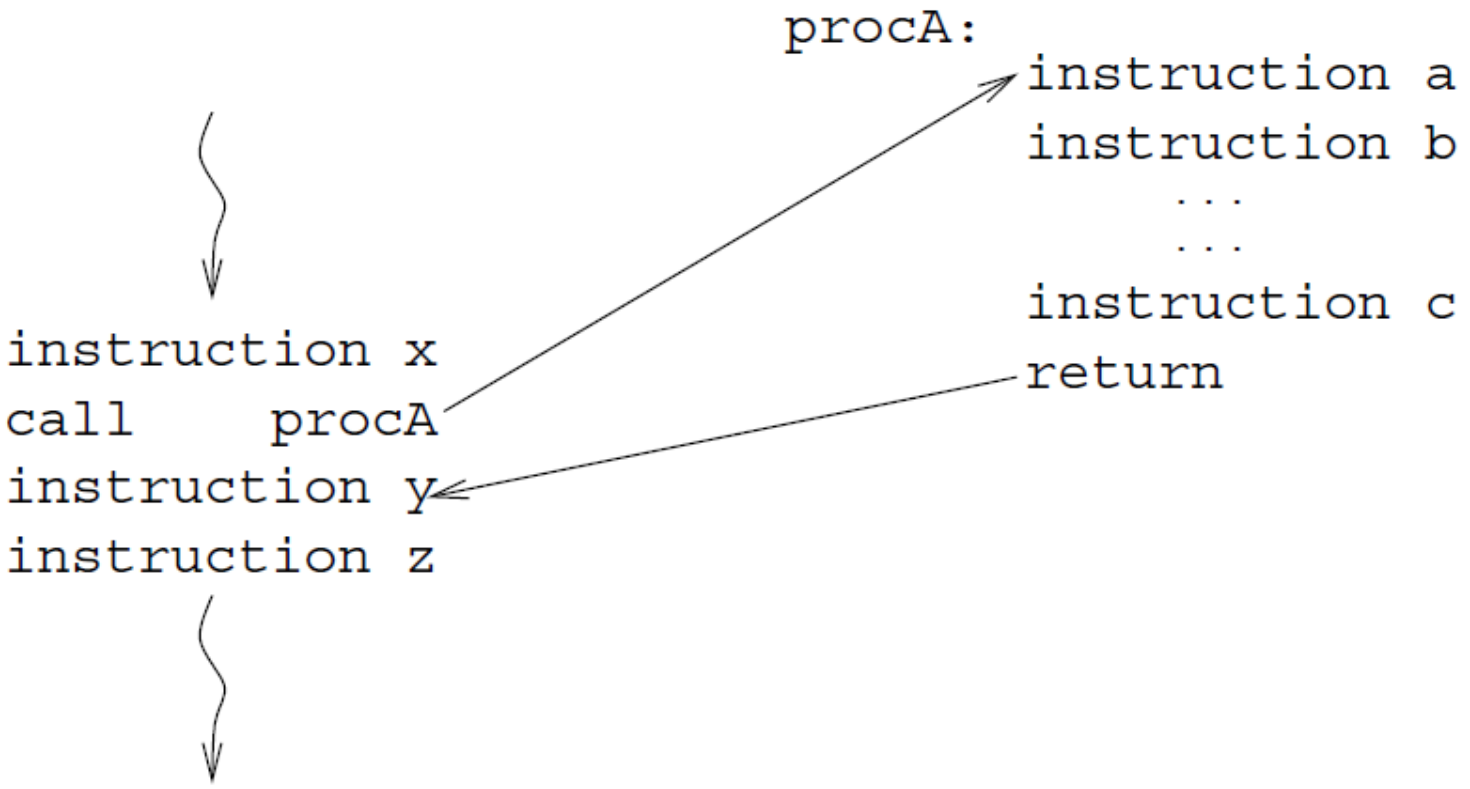
- Пример у случају процесора *Intel Pentium*

```
    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
```

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*
  - адреса повратка
    - мора бити сачувана при позивању процедуре

Пример позивања процедуре



Процесор

Пројектовање скупа инструкција

Пројектовање скупа инструкција

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

Начини адресирања

- 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

Врсте инструкција

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

Инструкције за премештање података

- Деле се на инструкције које премештају податке:
  - између меморије и регистара
  - између регистара
- Код *RISC* процесора је премештање података између процесора и меморије строго ограничено на инструкције:
    - *load*
    - *Store*
- Код CISC обично једна инструкција:
    - ***mov dest, src***
    - 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

Логичке инструкције

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

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 у претходној секцији)

Улазно / излазне инструкције

- Улазно / излазне инструкције се значајно разликују између процесора
- Уобичајене су две инструкције:
  - ***in Reg, io_port***
  - ***out io_port, Reg***

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 исинтрукције дужине 32 бита
  - формат променљиве дужине инструкција
    - уобичајен за *CISC* процесоре
    - нпр. *Intel x86*

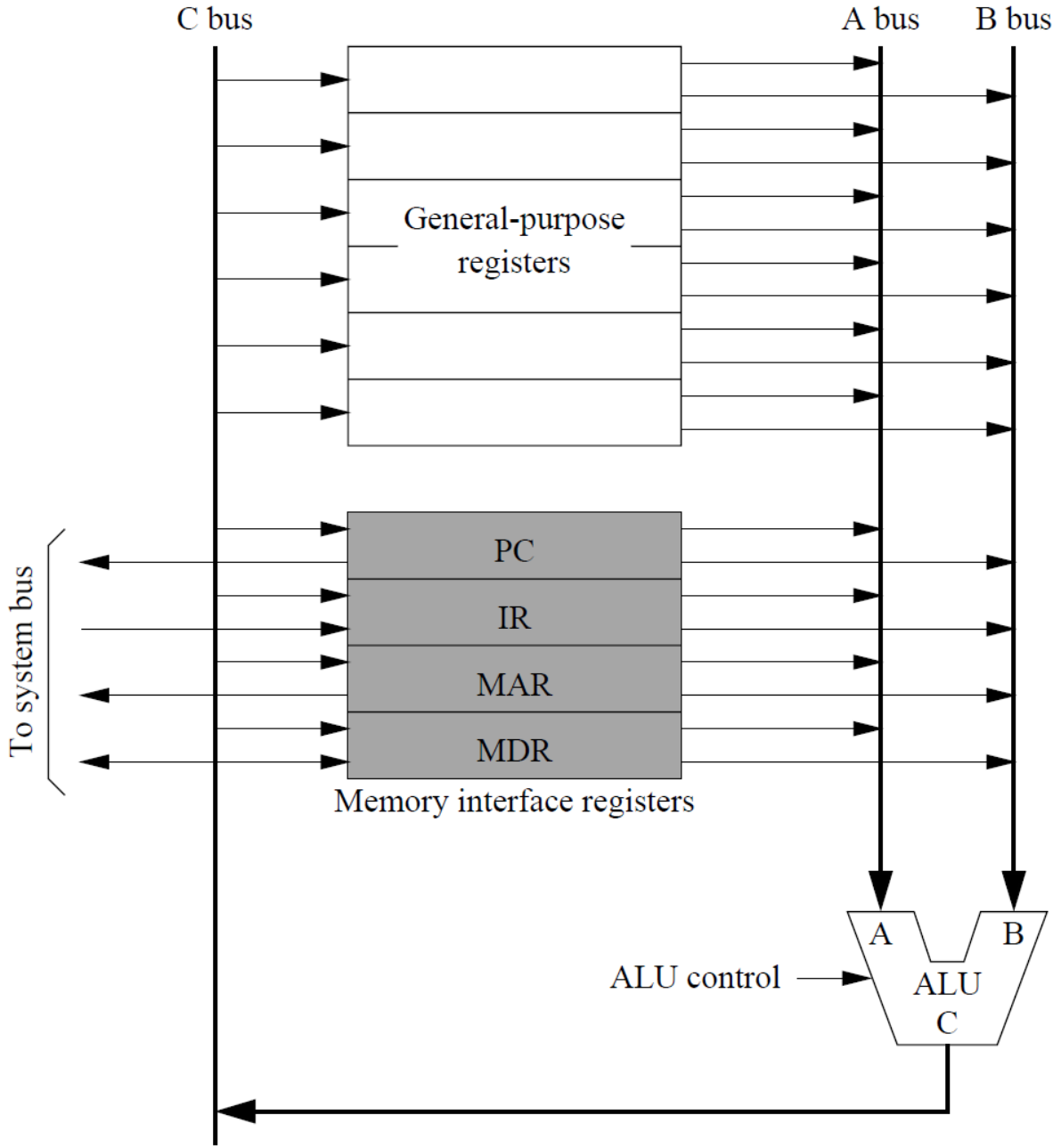
Процесор

Имплементација инструкција

Оквирни садржај процесора

- Процесор садржи:

1. Аритметичко логичку јединицу ALU
2. Регистре специјалне намене
3. Регистре опште намене
4. Везе (магистрале)
   - Пример са 3 магистрале: A, B и C
5. Контролну јединицу CU
   - Генерише контролне сигнале
   - Није нацртана на слици



Извршење инструкција

- Извршење инструкције се угрубо гледано састоји из две фазе:

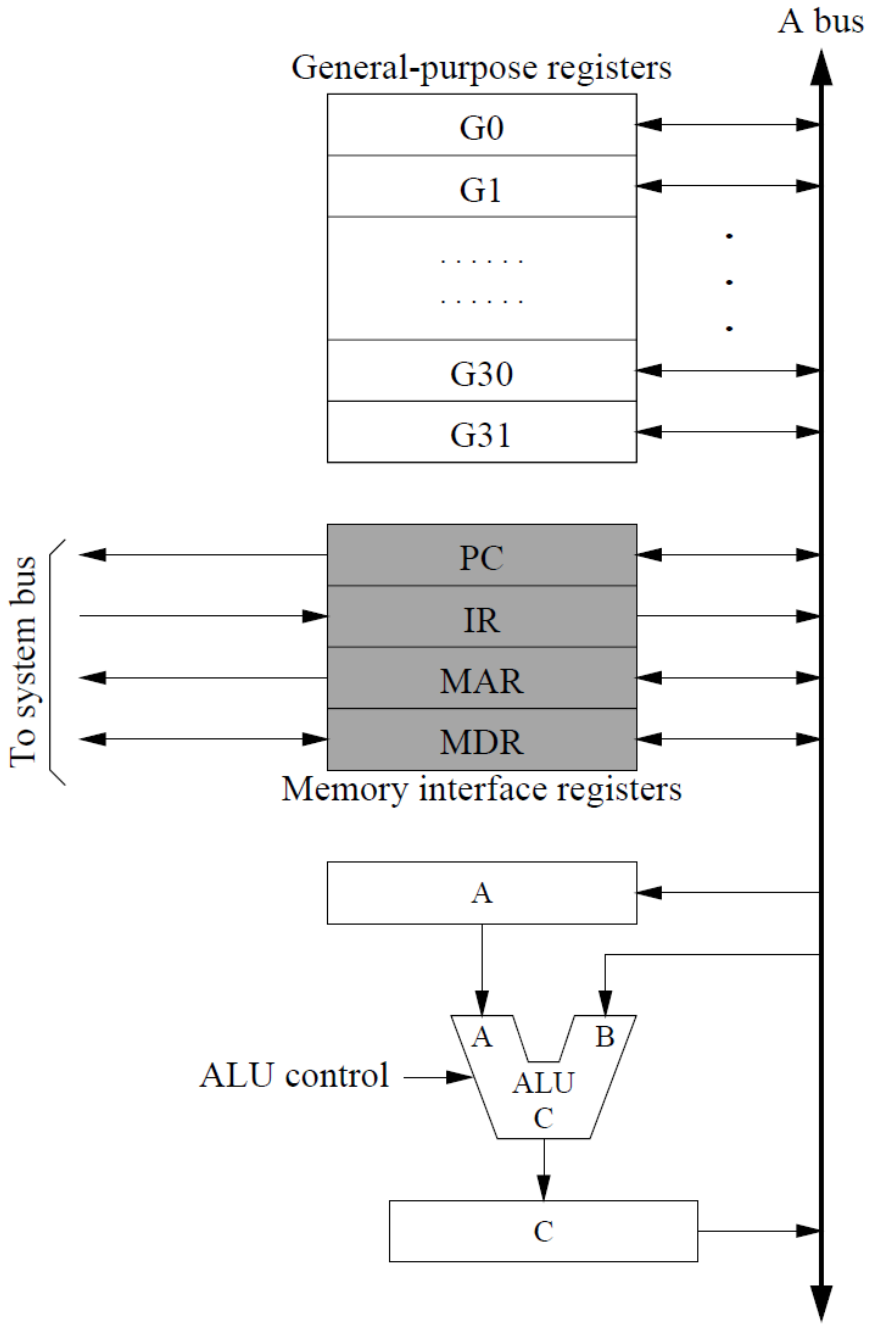
1. Читање инструкције (енг. Instruction fetch)
   a. Адресирање и читање инструкције из меморије
   b. Декодирање инструкције
   c. Адресирање и читање потребних операнада
2. Дословно извршење инструкције (енг. Instruction execute)
   a. Извршење одговарајуће операције у ALU
   b. Адресирање локације за резултат и записивање у меморију

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 (2)

- Само магистрала А
- Неки од регистара су повезани и са системском магистралом (о овоме касније...)
- Имплементација са јединственом магистралом јефтинија
- Али захтева постојање додатних регистара А и С

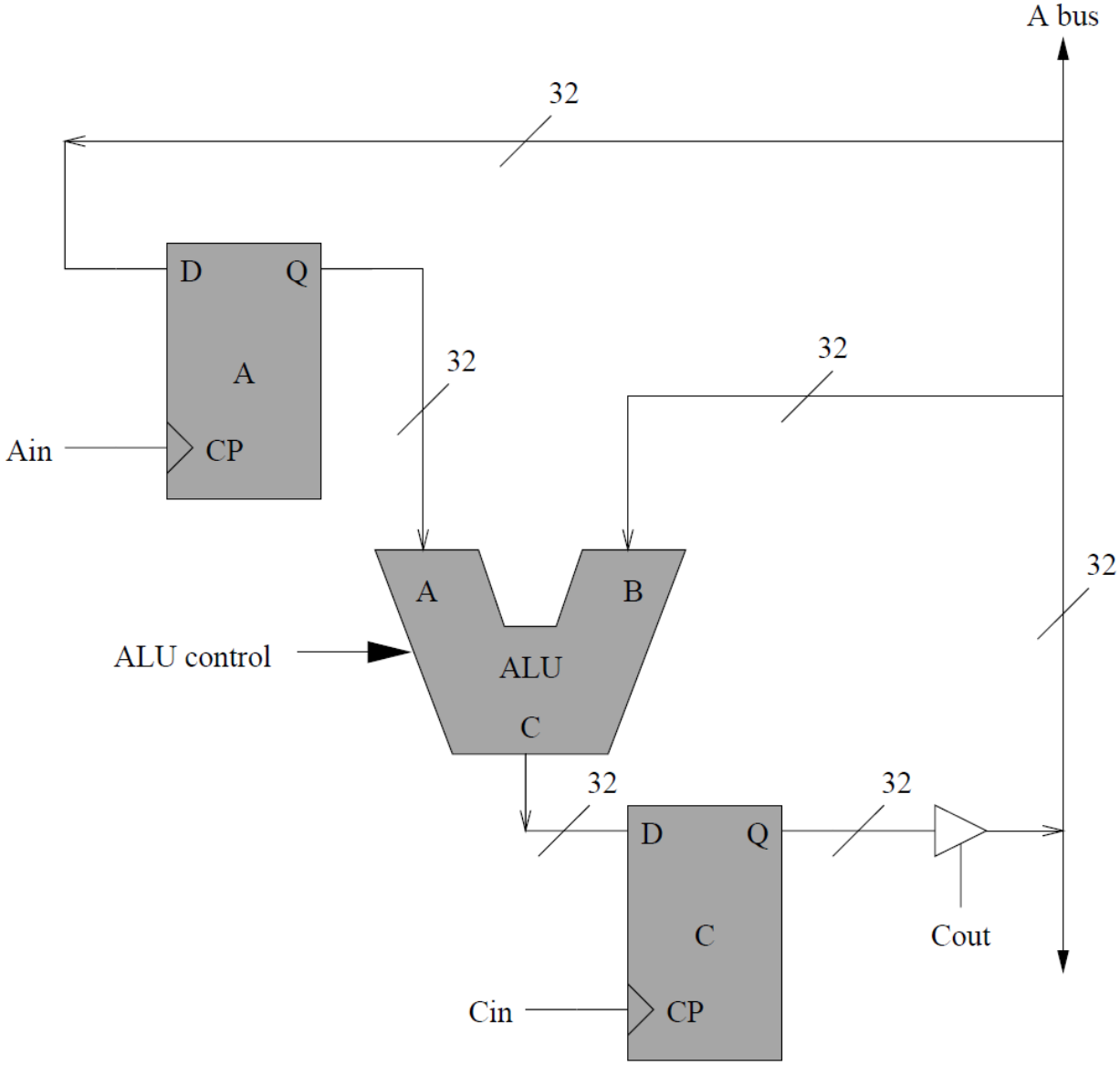


Додатни регистри

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

Фаза 2а: дословно извршење инструкције

1. Убацити први операнд у регистар A (сигнал Ain)
2. На магистралу A ставити други операнд, који ће ући на улаз B
3. Послати ALU сигнал за извршење одговарајуће операције
4. Резултат извршења ALU уписати у регистар C (сигнал Cin)
5. Садржај регистра C уписати на магистралу A (сигнал Cout)



Меморијски интерфејс

- Меморијски интерфејс чине 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

Регистар *PC*

- Бројач инструкција - садржи адресу наредне инструкције
- Три контролна сигнала
  1. ***PCin*** поставља вредност регистра
  2. ***Pcbout*** садржај ставља на системску адресну магистралу ради читања инструкције из меморије
  3. ***PCout*** садржај ставља на магистралу A ради омогућавања релативних скокова и позивања процедура

Регистар *IR*

- Регистар инструкције - садржи инструкцију која се тренутно извршава
- Два контролна сигнала:
  1. ***IRbin*** поставља вредност регистра
  2. ***IRout*** ставља вредност регистра на магистралу A

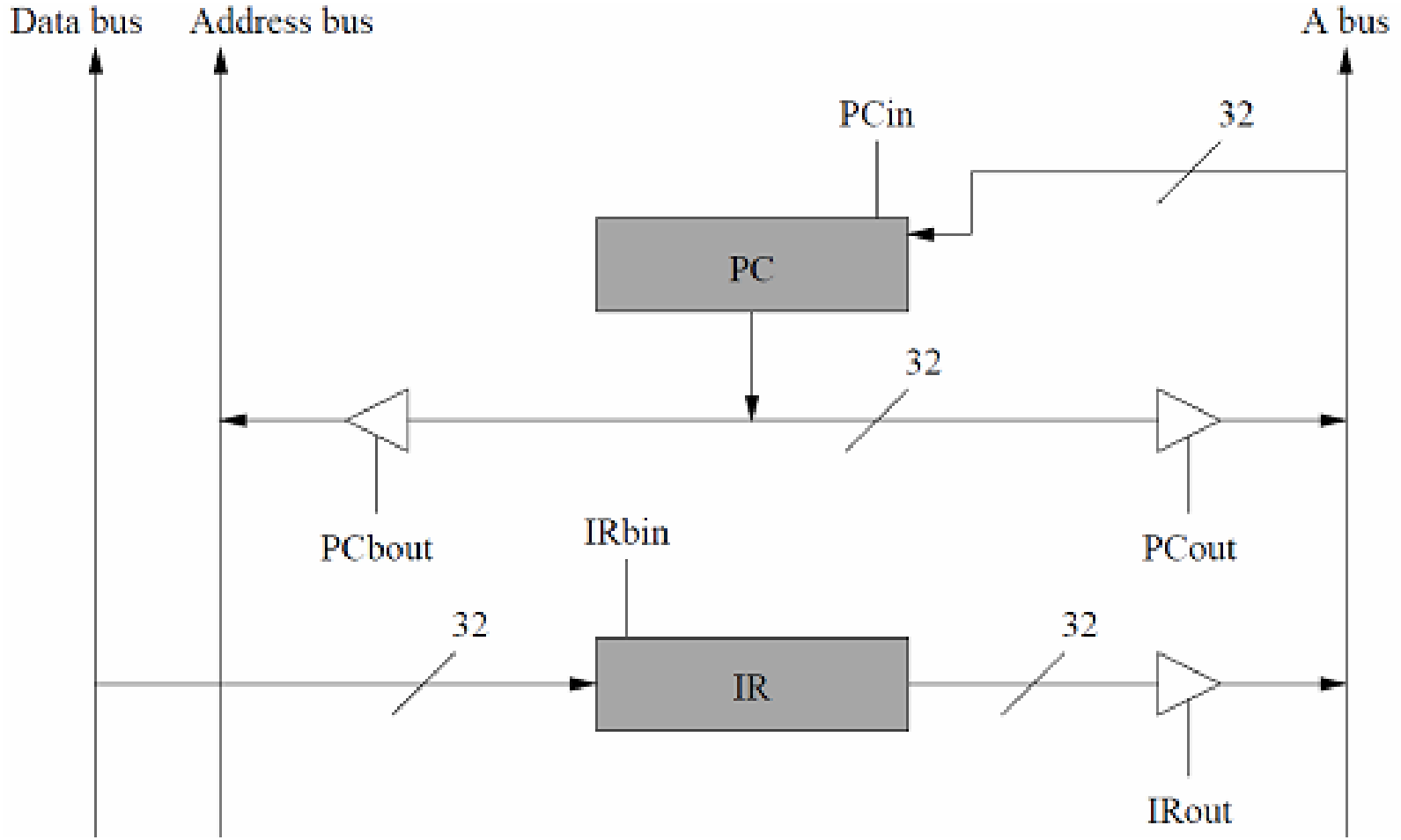
Фаза 1а: читање инструкције из меморије

PC увек садржи адресу следеће инструкције

1. PCbout – шаље адресу на системску адресну магистралу

Након неког времена инструкција се смешта на системску магистралу за податке

2. IRbin – учитава ову инструкцију у IR



Фаза 1b: декодирање инструкције

- Декодирање инструкције подразумева:
  1. Издвајање дела инструкције који се односи на тип операције
     - Овај део се после прослеђује ALU
       како би јој се нагласило шта треба да уради
  2. Издвајање делова који се односе на адресе операнада
     - Пре него што се започне дословно извршење инструкције,
       у регистре опште намене се морају допремити сви потребни операнди
     - Да би се ово урадило користе се специјални регистри MAR и MDR

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*
- Три контролна сигнала:
  1. ***MARin*** поставља вредност регистра
  2. ***MARout*** ставља вредност регистра на магистралу A
  3. ***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
- Има четири контролна сигнала:
  1. ***MDRin*** поставља вредност регистра са магистрале A
  2. ***MDRbin*** поставља вредност регистра са системске магистрале података
  3. ***MDRout*** ставља вредност регистра на магистралу A
  4. ***MDRbout*** ставља вредност регистра на системску магистралу података

Фаза 1c: адресирање и читање операнада

- **Након фазе 1b, адресе операнада су у IR и могу се са IRout послати на A**

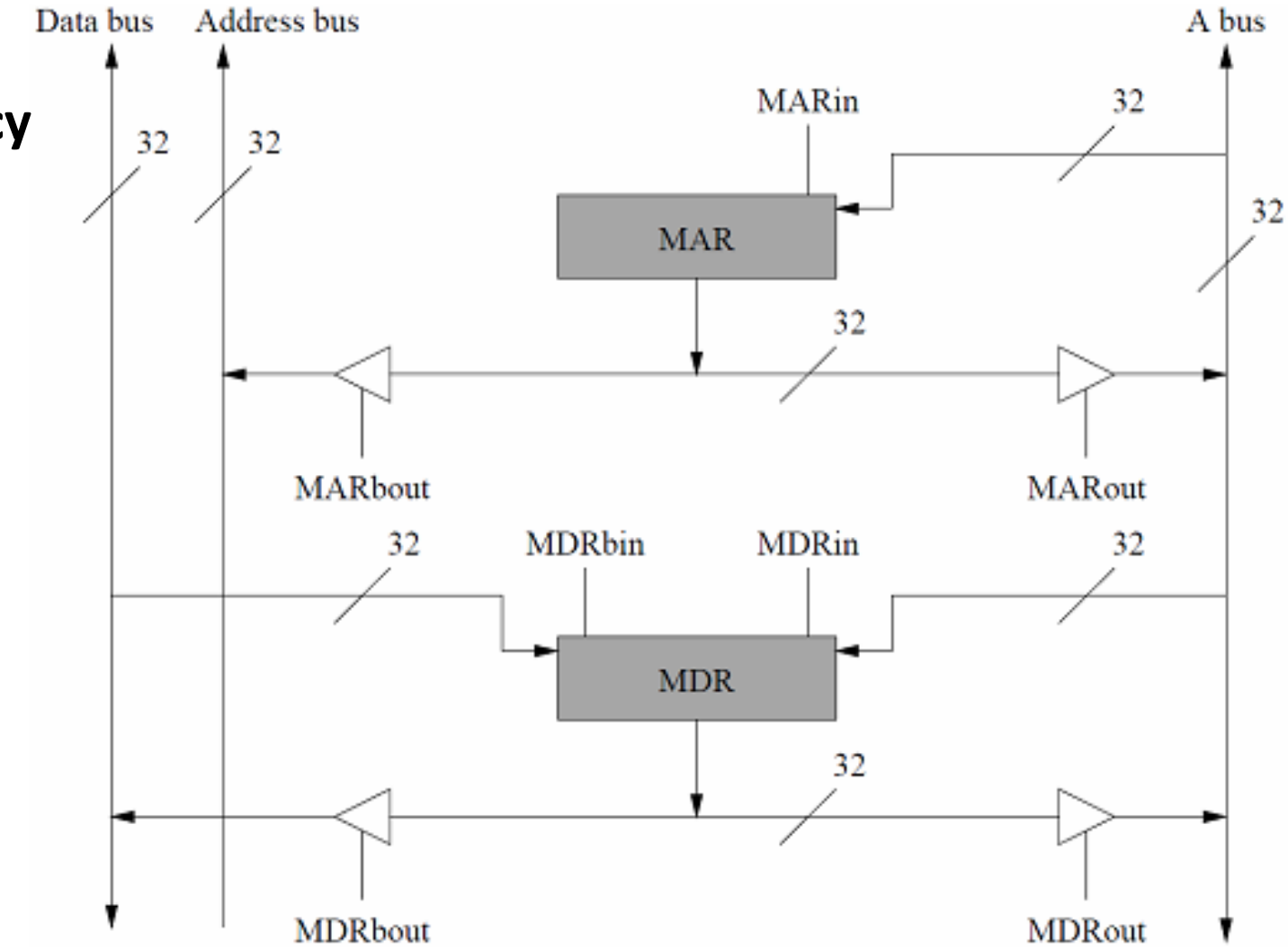
1. MARin – прихвата адресу са A и смешта у MAR

2. MARout – шаље на адресну системску магистралу

Након неког времена операнд се смешта на системску магистралу за податке

3. MDRbin – учитава операнд у MDR

4. MDRout – омогућава даље прослеђивање операнда на A
па даље у регистре опште намене



Фаза 2b: запис резултата у меморију

- **Регистри MAR и MDR се користе и за запис резултата назад у меморију**

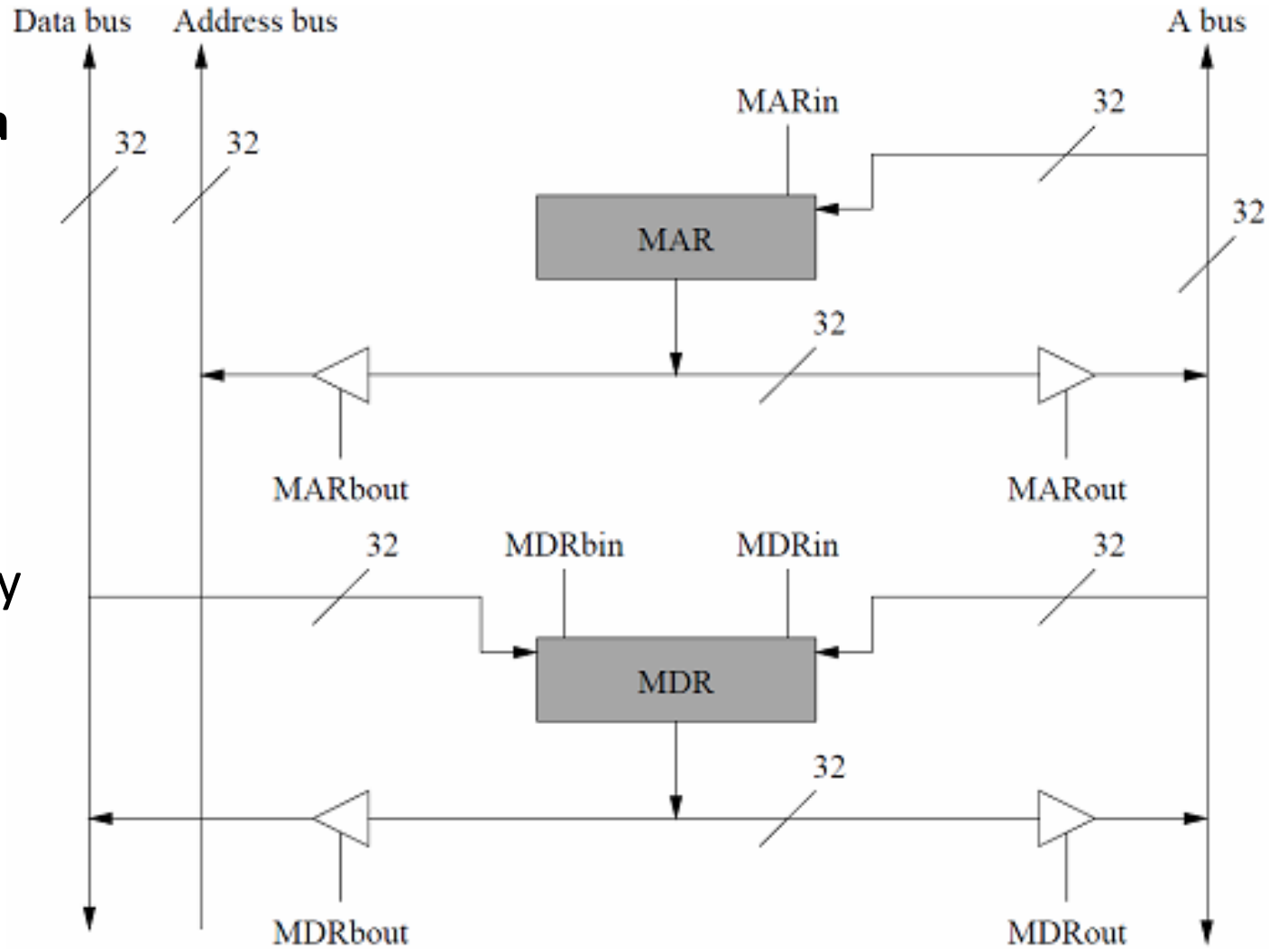
1. MARin – прихвата адресу са A и смешта у MAR

2. MARout – шаље на адресну системску магистралу

3. MDRin – са магистрале A учитава резултат који треба уписати у меморију

4. MDRbout – шаље податак на системску магистралу за податке

Након неког времена резултат из MDR ће бити уписан у меморији на адреси која је задата у MAR



Регистри опште намене

- Регистри опште намене су везани само на интерну магистралу А
- Сваки од регистара *Gx* има по два контролна сигнала
  1. ***Gxin*** учитава садржај са магистрале А
  2. ***Gxout*** пише на магистралу А

Пример инструкције – опис целог поступка

- Пратимо контролне сигнале на примеру инструкције (CISC процесор):

  add %G9, %G5, %G7 / G9 = G5 + G7 */*

- Пример урађен на табли