[P220]

Увод у архитектуру рачунара

13

Саша Малков
Универзитет у Београду
Математички факултет
2013/2014

[P271]

Увод у архитектуру рачунара

Саша Малков

Тема 14

Процесори - Имплементација инструкција

(наставак)

Декодирање инструкција

- Декодирање инструкције је заједничко за све врсте инструкција
- Следи непосредно после читања инструкције
- У фази декодирања се препознају
  - код инструкције
  - број и типови операнада
  - начини адресирања операнада

Универзитет у Београду - Математички факултет

Микропрограмски контролер

- Инструкције се могу описати коначним аутоматима
- Сваки од коначних аутомата може имплементирати хардверски или софтверски

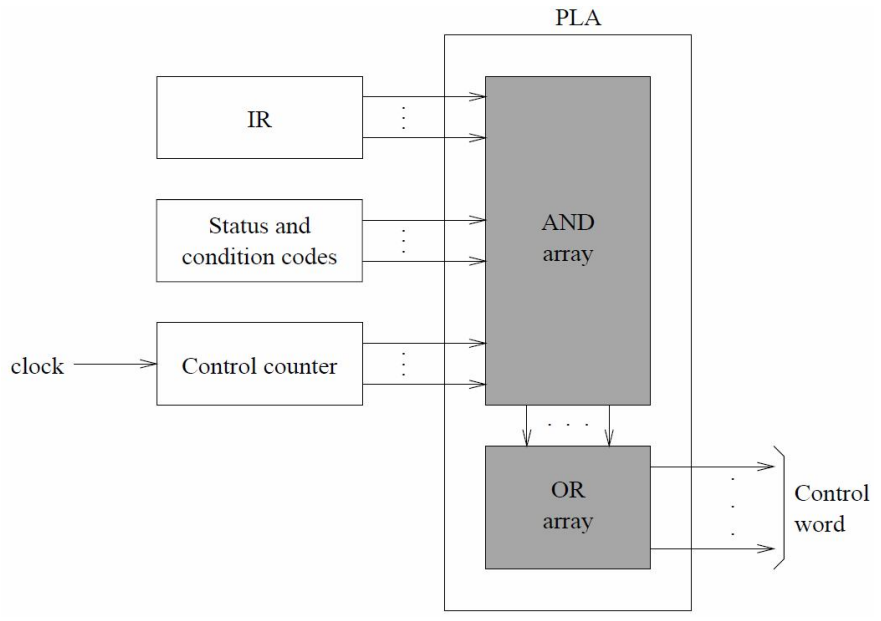
Универзитет у Београду - Математички факултет

Хардверска имплементација

- Улазе у *PLA* (програмибилни низови логичких елемената) чине три групе сигнала:
  - код операције (*opcode*)
    - операција која се имплементира
  - статусни и условни кодови
    - неопходни за имплементацију условних гранања и неких АЛ операција
  - часовник
    - служи за бројање корака и синхронизацију активности

Универзитет у Београду - Математички факултет

Дијаграм хардверске имплементације коначног аутомата



Универзитет у Београду - Математички факултет

Избор имплементације

- Ако су инструкције једноставне, онда је хардверска имплементација прихватљивија варијанта
  - *RISC*
- У случају сложених инструкција хардверска имплементација није добар избор
  - *CISC*
  - тада се сигналима управља програмски

Софтверска имплементација

- Почетком 1950-их је први пут предложено да се контрола извршавања инструкција имплементира софтверски [*Wilkes*, *Stinger*]
  - тзв. *микропрограмски контролери*
  - програмски код инструкције се назива *микро-код*
- Идеја је да се коначни аутомат трансформише у *микро-инструкције* које управљају постављањем сигнала који управљају радом процесора

Секвенца контролних сигнала

Instruction	Step	Control signals
Instruction fetch	S1	`PCbout: read: PCout: ALU=add4: Cin;`
	S2	`read: Cout: PCin;`
	S3	`read: IRbin;`
	S4	Decodes the instruction and jumps to the appropriate execution routine
`add %G9,%G5,%G7`	S1	`G5out: Ain;`
	S2	`G7out: ALU=add: Cin;`
	S3	`Cout: G9in: end;`

*** PCbout поставља сигнал на системску магистралу, а PCout на интерну магистралу A**

Универзитет у Београду - Математички факултет

Секвенца контролних сигнала (2)

- Ако претпоставимо да се сваки корак израчунава у једном циклусу:
  - потребно је 3 циклуса за читање
  - бар један циклус за декодирање
  - три циклуса за сабирање
- Већина сигнала је из претходног описа, осим:
  - *read* – постављање сигнала за читање на системску магистралу
  - *ALU* – је група контролних сигнала АЛЈ која укључује одговарајућу операцију
  - *end* – сигнал који означава крај и иницира циклус читања наредне инструкције

Универзитет у Београду - Математички факултет

Други пример

- Разматрамо само извршавање инструкције
  - регистарско индиректно адресирање једног сабирка

Instruction	Step	Control signals
add %G9,[%G5],%G7	S1	G5out: MARin: MARbout: read;
	S2	read;
	S3	read: MDRbin: MDRout: Ain;
	S4	G7out: ALU=add: Cin;
	S5	Cout: G9in: end;

Универзитет у Београду - Математички факултет

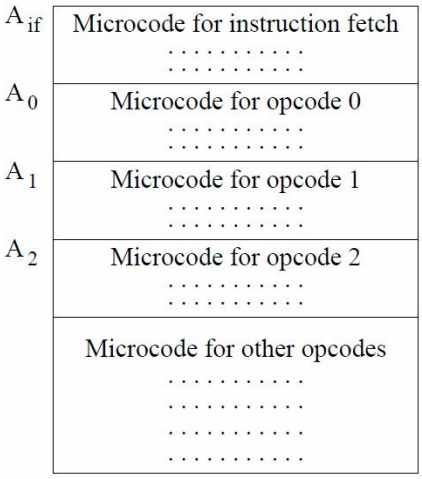
Концепт микро-кода

- Идеја:
  - сви сигнали једног корака се кодирају у тзв. *кодну реч*
  - добијена кодна реч представља *микроинструкцију*
  - низ микроинструкција који одговара једној инструкцији процесора гради *микрорутину*
  - помоћу микроинструкција се пишу *микропрограми*

Универзитет у Београду - Математички факултет

Упрошћена организација микрокода

- Линеарна организација микрокода почиње од читања инструкције
- Након декодирања инструкције наставља се од микрорутине која одговара коду операције
- Извршавање микрорутине је секвенцијално
- Када се дође до сигнала *end* понавља се читање (наредне) инструкције

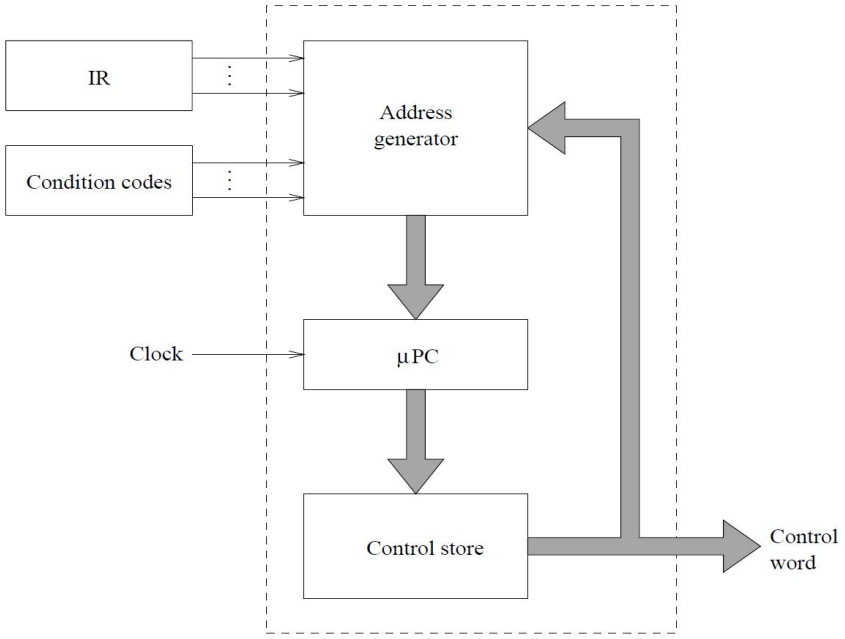


Универзитет у Београду - Математички факултет

Микропрограмски контролер

- Микропрограм се извршава од стране микропрограмског контролера
- Извршавање почива на микропрограмском бројачу (μPC) – слично бројачу инструкција
- Коло за генерисање адресе
  - иницира почетну адресу (тј. адресу микрорутине за читање инструкције)
  - имплементира микропрограмске скокове
  - користи се и да на основу прочитане инструкције условног гранања и стања контролних битова одабере одговарајући микрокод

Универзитет у Београду - Математички факултет



Сложена организација микрокода

- Линеарна организација микрокода има за последицу да се неки заједнички делови микрокода понављају више пута
- Сложена организација микрокода подразумева да се заједнички делови ефикасније организују у микрорутине са гранањима
  - свака микроинструкција се проширује адресом наредне микроинструкције
  - због тога се повећава контролна реч
  - али се штеди на дужини микропрограма

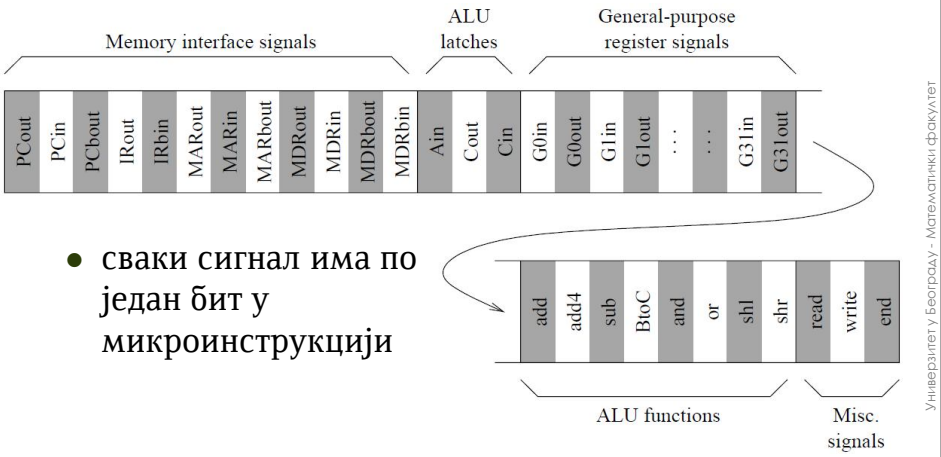
Формат микроинструкција

- Свака микроинструкција се састоји од одговарајућих контролних сигнала
  - првих 12 контролних сигнала долази из имплементације пута података:
    - *PCin, PCbout, PCout, IRbin, IRout, MARin, MARbout, MARout, MDRbin, MDRin, MDRbout, MDRout*
  - наредних 3 бита контролишу резе A и C
    - *Ain, Cin, Cout*
  - 64 сигнала контролишу регистре опште намене
    - *Gxin, Gxout* – по 2 за сваки регистар *Gx*
  - потребан број битова за операције процесора

Формат микроинструкција (2)

- Потребан број битова за операције процесора зависи од скупа подржаних операција
  - претпоставимо да су подржане операције:
    - *add, sub, and, or*
    - *shl, shr* – померање за један бит улево и удесно
    - *add4* – за повећавање програмског бројача
    - *BtoC* – преписивање из једног у други регистар (или меморију)
      - може и без тога, сигналима: *Gxout: Gyin:*

Хоризонталан формат



- сваки сигнал има по један бит у микроинструкцији

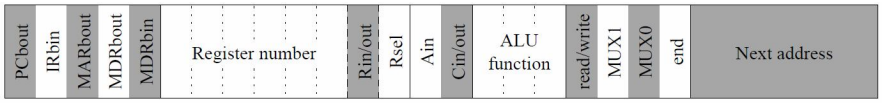
Хоризонталан формат (2)

- Сваком сигналу одговара по један бит
- Нема кодирања и декодирања сигнала
- Предности:
  - омогућава навођење великог броја сигнала у једној инструкцији
    - нпр: преписивање *G0* у више регистара:
      - *G0out: G2in: G3in: G4in: G5in: G6in: end;*
- Мане:
  - величина микроинструкције
    - у претходном примеру читавих 90 битова

Вертикалан формат

- На дужини се може уштедети ако се битови кодирају
  - на пример, уместо 64 битова за контролисање 32 регистра је довољно 5 битова за идентификовање регистра и 1 бит за избор сигнала
- Предности:
  - краће микроинструкције
- Мане:
  - у једном кораку се може навести само један регистар

Вертикалан формат (2)



- више сигнала се кодира мањим бројем битова
- 32 регистра опште намене и 4 специјална регистра (*PC*, *IR*, *MAR*, *MDR*) се кодирају са 6 битова
- 8 операција се кодирају са 3 бита

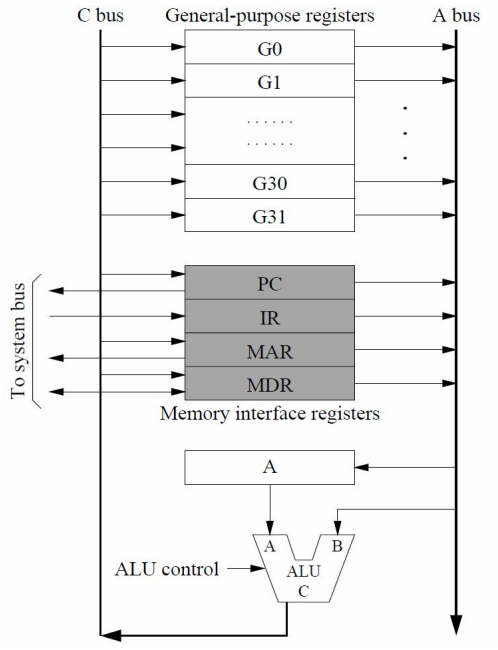
Вертикалан формат (3)

- Неопходно је користити *BtoC* при преписивању
  - на пример, преписивање *G0* у *G2-G6*:
    - *G0out: ALU=BtoC: Cin;*
    - *Cout: G2in;*
    - *Cout: G3in;*
    - *Cout: G4in;*
    - *Cout: G5in;*
    - *Cout: G6in: end;*

Путеви података са више магистрала

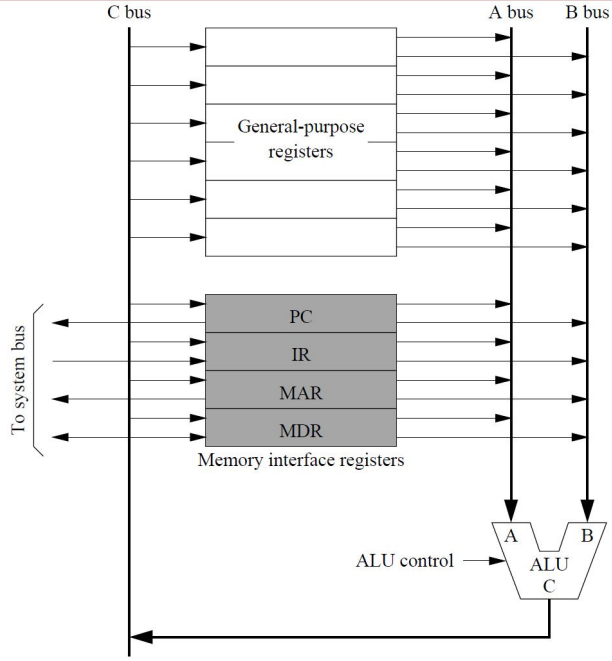
- Представљен пут података има само једну интерну магистралу A
  - основна слабост је милтиплексирање те магистрале и већи број корака у имплементацији инструкција
- Пут података може да има и више интерних магистрала
  - на пример две:
    - магистрала A – за један операнд
    - магистрала C – за резултат
  - или три:
    - магистрала A – за један операнд
    - магистрала B – за други операнд
    - магистрала C – за резултат

Пут података са две магистрале



Универзитет у Београду - Математички факултет

Пут података са три магистрале



Универзитет у Београду - Математички факултет

Примери са више магистрала

- у случају једне магистрале неопходна су три корака
- у случају две магистрале довољна су два корака:

Instruction	Step	Control signals
`add %G9,%G5,%G7`	S1	`G5out: Ain;`
	S2	`G7out: ALU=add: G9in;`

- у случају три магистрале довољан је један корак:

Instruction	Step	Control signals
`add %G9,%G5,%G7`	S1	`G5outA: G7outB: ALU=add: G9in;`

Универзитет у Београду - Математички факултет

[P271]

Увод у архитектуру рачунара

Саша Малков

Тема 15

Процесори *Intel x86*

Фамилија процесора *Intel x86*

- Формалан назив архитектуре је *Intel Architecture* (*IA*)
- Први представник је процесор 8086, из 1979. год.
  - 20-битна адресна и 16-битна магистрала података

Преглед процесора фамилије *IA* (*Intel x86*)

Ген.	Процесор	Год.	Фрекв. *MHz*	Транз.	Техн. (nm)	Језгра	Шир. рег.	Маг. под.	Маг. адр.	Макс. адр. физ./вирт.
1	8086	1979	4.77–10	29K	3000	1	16	16	20	64KiB / *1MiB*
1	8088		4.77–8	29K	3000	1	16	8	20	64KiB / *1MiB*
2	80186	1982	6	29K	2000	1	16	16	20	64KiB / *1MiB*
2	80286	1982	6–25	134K	1500	1	16	16	24	64KiB/16*MiB*/1GiB
3	80386	1985	20–33	275K	1000	1	32	32	32	4GiB / 64GiB
4	80486	1989	25–50	1.2M	1000-800	1	32	32	32	4GiB / 64GiB
5	Pentium	1993	60–200	3.1M	800-350	1	32	64	32	4GiB / 64GiB
6	Pentium Pro	1995	150–200	5.5M	600-350	1	32	64	36	64GiB
6	Pentium II	1997	233–333	7.5M	350-350	1	32	64	36	64GiB
6	Pentium III	1999	450–1400	8.2M	250-130	1	32	64	36	64GiB
7	Pentium IV	2000	1400–3400	42M	180-130	1	32	64	36	64GiB
8	P4 Prescot	2004	2800–3800	125-169M	90	1-2	64	64	64 (40)	1TiB / 16EiB
9	Core 2	2006	1860–3200	291-820M	65-45	2-4	64	64	64 (40)	1TiB / 16EiB
10	Core i7	2008	2530–3330	781M	45-32	4-8	64	64	64 (48)	256TiB / 16EiB
11	i7, Sandy Bridge	2011	2800-3500	995M-2.2G	32	4-8	64	64	64 (48)	256TiB / 16EiB

Основни сигнали *Pentium*-а

- Магистрала података: ***D0*** - ***D63***
- Магистрала адресе: ***A0*** - ***A31***
- Битови *byte enable*: ***BE0#*** - ***BE7#***
  - означавају који део магистрале података се користи
  - сваки се односи на по 8 линија магистрале података
- Битови парности: ***DP0*** - ***DP7***
  - сваки се односи на по 8 битова података
- Провера парности: ***PCHK#***
  - резултат провере парности процесора
- Омогућена провера парности: ***PEN#***
- Парност адресе: ***AP***
  - резултат провере парности адресе: **A5** - **A31**
  - не проверавају се **A0**-**A4**
- Провера парности адресе: ***APCHK#***
  - резултат провере парности адресе

Универзитет у Београду - Математички факултет

Основни сигнали *Pentium*-а (2)

- Меморија / УИ: ***M/IO#***
  - да ли магистралу користи меморија (1) или УИ (0)
- Писање / читање: ***W/R#***
  - да ли се пише (1) или се чита (0)
- Подаци / код: ***D/C#***
  - да ли се приступа подацима (1) или коду (0)
- Могуће кеширање: ***CACHE#***
  - у случају читања наглашава да ли се читање може кеширати, у случају писања означава брзо преписивање
- Катанац магистрале: ***LOCK#***
  - Означава да се скевенца употребе магистрале од стране процесора не сме прекинути
- Прекид: ***INTR***
- Немаскирани прекид: ***NMI***
- Часовник: ***CLK***

Универзитет у Београду - Математички факултет

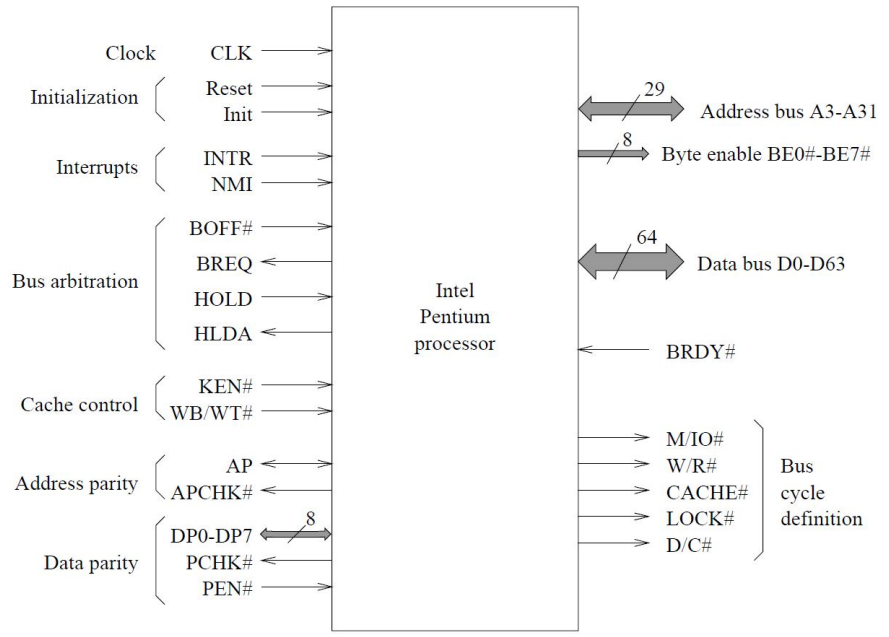
Основни сигнали *Pentium*-а (3)

- Спреман за употребу магистрале: ***BRDY#***
  - користе га спољни уређаји за проширивање циклуса (стања чекања)
- Захтев за магистралу: ***BREQ***
  - процесор га поставља увека када користи магистралу, користи се од стране спољашњег арбитра
- Прекид магистрале: ***BOFF#***
  - улазни сигнал који прекида текућу секвенцу на магистрали – биће поновљена са следећим циклусом, али тек пошто овај сигнал буде склоњен
  - користи се за разрешавање мртвих петљи на магистрали
- Задржавање магистрале: ***HOLD***
  - улазни сигнал који обавештава процесор да након завршавања секвенце на магистрали мора да је ослободи
- Препуштање магистрале: ***HLDA***
  - излазни сигнал којим процесор потврђује да је препустио магистралу

Основни сигнали *Pentium*-а (4)

- Омогућавање кеша: ***KEN#***
  - улазни сигнал који се користи за установљавање да ли процесор у овом циклусу може да подржи пуњење лијије кеша
- Писање са преписивањем или са пропуштањем: ***WB/WT#***
  - сигнал означава да ли би линија кеша требало да користи политику писања са преписивањем или са пропуштањем
- Поништавање: ***RESET***
  - сигнал принудно враћа процесор у основно стање
  - процесор почиње извршавање кода од адресе *FFFFFFF0H*
  - празне се све врсте кеша (и регистри у покретном зарезу)
- Иницијализација: ***INIT***
  - сигнал принудно враћа процесор у основно стање
  - процесор почиње извршавање кода од адресе *FFFFFFF0H*
  - не празни се кеш ни регистри у покретном зарезу

Преглед основних сигнала процесора *Intel Pentium*



Регистри процесора *Pentium*

- Регистри података
- Индексни и показивачки регистри
- Контролни регистри
- Сегментни регистри

Регистри података (16 bit)

- Регистри података:
  - 8-битни: *AH*, *AL*, *BH*, *BL*, *CH*, *CL*, *DH*, *DL*
  - 16-битни: *AX*, *BX*, *CX*, *DX* или
- Индексни и показивачки регистри
  - 16-битни: *DI*, *SI*, *BP*, *SP*

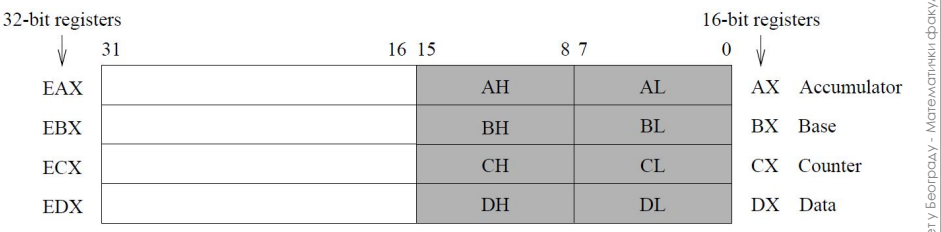
Универзитет у Београду - Математички факултет

Регистри података (32 bit)

- Регистри података:
  - 8-битни: *AH*, *AL*, *BH*, *BL*, *CH*, *CL*, *DH*, *DL*
  - 16-битни: *AX*, *BX*, *CX*, *DX* или
  - 32-битни: *EAX*, *EBX*, *ECX*, *EDX* или
- Индексни и показивачки регистри
  - 32-битни: *EDI*, *ESI*, *EBP*, *ESP* или
  - 16-битни: *DI*, *SI*, *BP*, *SP*

Универзитет у Београду - Математички факултет

Регистри података



Регистри података (64 bit)

- Даље се проширују регистри и додају се нови
- Регистри података:
  - 8-битни: *AH*, *AL*, *BH*, *BL*, *CH*, *CL*, *DH*, *DL*
  - 16-битни: *AX*, *BX*, *CX*, *DX* или
  - 32-битни: *EAX*, *EBX*, *ECX*, *EDX* или
  - 64-битни: *RAX*, *RBX*, *RCX*, *RDX* или
- Индексни и показивачки регистри
  - 64-битни: *RDI*, *RSI*, *RBP*, *RSP* или
  - 32-битни: *EDI*, *ESI*, *EBP*, *ESP* или
  - 16-битни: *DI*, *SI*, *BP*, *SP*
- Додатно у 64-битним процесорима:
  - 64-битни: *R8*, *R9*,... *R15* или
  - 32-битни: *R8D*, *R9D*,... *R15D* или
  - 16-битни: *R8W*, *R9W*,... *R15W* или
  - 8-битни: *R8B*, *R9B*,... *R15B* или

Контролни регистри

- Контролни регистри
  - показивач инструкција (бројач)
    - 32-битни: *EIP* или
    - 16-битни: *IP*
  - заставице (контролни битови)
    - 32-битни: *EFLAGS* или
    - 16-битни: *FLAGS*

Регистар (*E*)*FLAGS (1)*

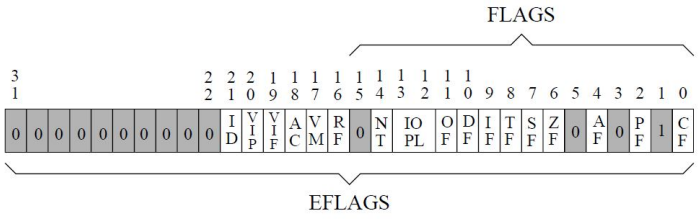
- Статусни битови (заставице)
  - *CF* (*carry flag*) – пренос
  - *PF* (*parity flag*) – парност
  - *AF* (*auxilary carry flag*, *adjust flag*) – пренос са прва 4 бита (за *BCD* аритметику)
  - *ZF* (*zero flag*) – нула
  - *SF* (*sign flag*) – знак
  - *OF* (*overflow flag*) – прекорачење
- Контролни битови (заставице)
  - *DF* (*direction flag*) – инструкције које аутоматски мењају индексне регистре их повећавају (0) или смањују (1)

Регистар (E)FLAGS (2)

- Важнији системски битови (заставице)
  - *TF* (*trap flag*) – замка после сваке инструкције
  - *IF* (*interrupt flag*) – да ли су допуштени прекиди
  - *IOPL* (*I/O privilege level*) – ниво У/И привилегије процеса
  - *VM* (*virtual* 8086 *mode*) – да ли процесор ради у реалном режиму (као 286)
  - *AC* (*alignment check*) – да ли је допуштен само рад са поравнатим адресама
  - *ID* (*ID flag*) – да ли процесор подржава инструкцију *CPUID* за једнозначно идентификовање процесора

Универзитет у Београду - Математички факултет

Контролни регистар процесора *Intel Pentium*



Status flags

CF = Carry flag
PF = Parity flag
AF = Auxiliary carry flag
ZF = Zero flag
SF = Sign flag
OF = Overflow flag

Control flags

DF = Direction flag

System flags

TF = Trap flag
IF = Interrupt flag
IOPL = I/O privilege level
NT = Nested task
RF = Resume flag
VM = Virtual 8086 mode
AC = Alignment check
VIF = Virtual interrupt flag
VIP = Virtual interrupt pending
ID = ID flag

Универзитет у Београду - Математички факултет

Сегментни регистри

- Сегментни регистри су 16-битни
  - *CS* (*code*)
    - идентификује сегмент у коме се налази код
    - регистар *IP* адресира у оквиру сегмента *CS*
  - *DS* (*data*)
    - идентификује сегмент у коме се налазе подаци
    - већина начина адресирања се односи на овај сегмент
  - *SS* (*stack*)
    - идентификује сегмент у коме се налази стек
    - сви видови индиректног адресирања где је базна адреса у регистру *SP* или *BP* се односе на овај сегмент
  - *ES* (*extra*), *FS* (*extra*), *GS* (*extra*)
    - идентификују додатне сегменте података
    - користе се као и *DS*, али искључиво експлицитно

[P271]

Увод у архитектуру рачунара

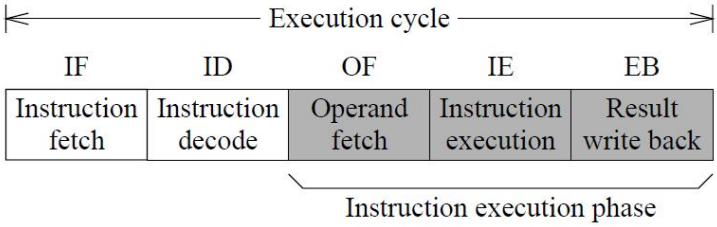
Саша Малков

Тема 16

Суперскаларни и векторски процесори

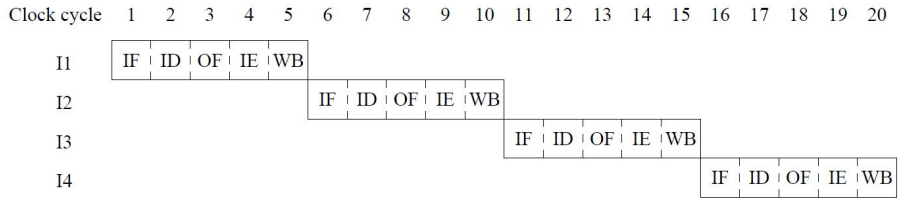
Фазе извршавања инструкције

- Извршавање инструкције се дели на више корака (фаза)
  - (енгл. *stage*)
- Обично је за сваку фазу задужен посебан део процесора
- Пример фаза:



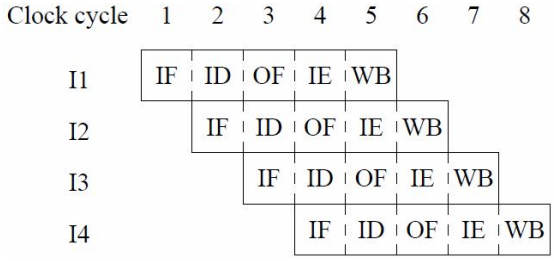
Серијско извршавање

- Серијско извршавања инструкција подразумева да се све фазе једне инструкције изврше пре него што започне прва фаза извршавања друге инструкције



Преклапање инструкција

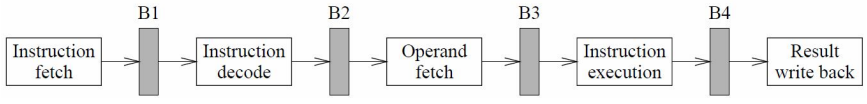
- Преклапање извршавања инструкција значи да се у истом тренутку се одвијају различите фазе извршавања различитих инструкција
- Подиже се пропусна моћ система
- Преклапање инструкција ради најбоље ако све фазе имају исто трајање



Универзитет у Београду - Математички факултет

Бафери

- За преклапање је неопходно да између сваке две фазе постоји бафер, који прихвата информације прикупљене током претходне фазе
  - инструкцију
  - операнде



Универзитет у Београду - Математички факултет

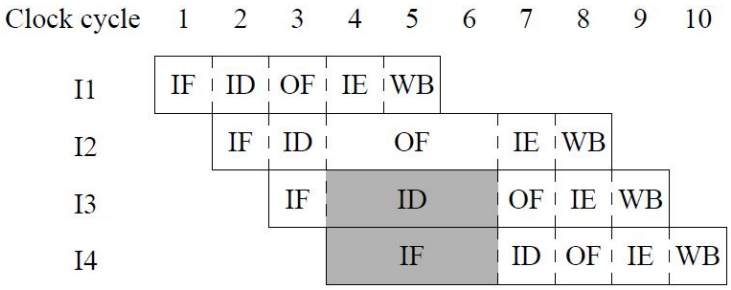
Потенцијални проблеми

- Преклапање почиње да прави проблеме
  - ако нека фаза неке инструкције потраје дуже
    - онда се успорава не само та инструкција него и све остале инструкције које су у наредним фазама
  - ако постоје међузависности међу неким корацима различитих инструкција

Универзитет у Београду - Математички факултет

Пример различитог трајања

- У овом примеру се претпоставља да фаза читања операнда инструкције *I2* траје три циклуса



Универзитет у Београду - Математички факултет

Узроци различитог трајања фаза

- Трајање читања инструкције (*IF*)
  - зависи од дужине инструкција
- Трајање фазе читања операнада (*OF*)
  - зависи од локације операнда (да ли је у регистру, кешу, меморији или у оквиру инструкције)
  - зависи од начина адресирања
    - сложена адресирања захтевају више времена
- Трајање извршавање (*IE*)
  - зависи од сложености инструкције
    - може бити више циклуса
- Трајање уписивања резултата (*WB*)
  - зависи од локације операнда
  - зависи од начина адресирања

Универзитет у Београду - Математички факултет

Застоји у току извршавања

- До застоја при извршавању токова инструкција може доћи из више разлога
  - ти разлози се називају *хазарди*
- Постоје три врсте хазарда
  - хазарди ресурса
    - када више инструкција које су у току желе исте ресурсе
  - хазарди података
    - када резултат неке од инструкција које су у току представља операнд неке од наредних инструкција
    - мора се водити рачуна о редоследу израчунавања
  - хазарди контроле
    - у случају промене тока извршавања (скокова или позива процедура), одбацују се све инструкције које су у току

Универзитет у Београду - Математички факултет

Пример сукоба око ресурса

- Претпоставимо претходно представљен пример преклапања са 5 фаза
- Занемаримо кеш и претпоставимо да су сви подаци у брзој меморији са једним комуникационим портом
  - подржана је једна операција у једном тренутку
- Онда се из меморије у једном тренутку може
  - или читати инструкција
  - или читати операнд
  - или писати резултат
- Трећа инструкција неће моћи по плану да чита инструкцију
  - најпре мора да чека на читање операнда прве инструкције
  - затим мора да чека на читање операнда друге инструкције
  - па на писање резултата прве инструкције
  - па на писање резултата друге инструкције

Универзитет у Београду - Математички факултет

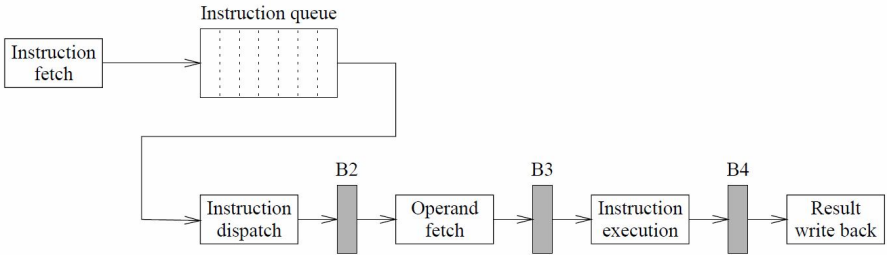
Решавање сукоба око ресурса (1)

- Најопштији начин решавања сукоба око ресурса је повећавање расположивих ресурса
  - ако би се обезбедиле паралелне линије за читање инструкција и операнада, онда би се смањила међузависност корака
    - то је мотив за раздвајање магистрале кода од магистрале података (раније помињана архитектура *Харвард*) и раздвајање кеша за програм и податке
  - …

Универзитет у Београду - Математички факултет

Решавање сукоба око ресурса (2)

- ...
- ако би се подаци (и код) из меморије читали унапред (енгл. *prefetching*), у бафер који има више комуникационих линија, било би мање сукоба
  - кеш са више комуникационих линија
  - специјализован ред између корака *IF* и *ID*



Универзитет у Београду - Математички факултет

Хазарди података

- Међузависност података може нарушити перформансе
  - пример:

```
I1:     add     R2, R3, R4      /* R2 = R3 + R4 */
I2:     sub     R5, R6, R2      /* R5 = R6 – R2 */
```

Универзитет у Београду - Математички факултет

Врсте сукоба око података

- Читање-после-писања (*RAW*)
  - једна инструкција мења неки податак
  - нека каснија инструкција чита тај исти податак
- Писање-после-читања (*WAR*)
  - једна инструкција чита неки податак
  - нека каснија инструкција мења тај исти податак
  - сукоб настаје у контексту суперскаларности
- Писање-после-писања (*WAW*)
  - једна инструкција мења неки податак
  - нека каснија инструкција мења тај исти податак
  - сукоб настаје у контексту суперскаларности
- У случају читања-после-читања (*RAR*) нема сукоба

Последице сукоба око података

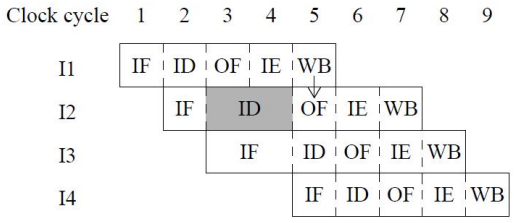
- Проблем исправности
  - ако се сукоб не уочи на време и наредна операција не заустави на време, може се догодити да се у обраду укључе неисправни подаци
- Проблем ефикасности
  - ако се не осмисли неко додатно решење, застоји у току инструкција могу трајати значајно дуго
- Основне технике за разрешавање:
  - прослеђивање регистара (енгл. *register forwarding*)
  - закључавање регистара (енгл. *register interlocking*)

Прослеђивање регистара

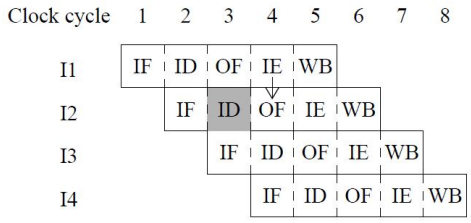
- Техника се назива и *премошћавање* (енгл. *bypassing*)
- Основна идеја је да се резултат испоручи чим постане доступан у току података извршавања инструкције
  - начин 1: резултат се начини доступним већ током фазе писања
  - начин 2: резултат се начини доступним већ током фазе израчунавања

Прослеђивање регистара

- техника 1:

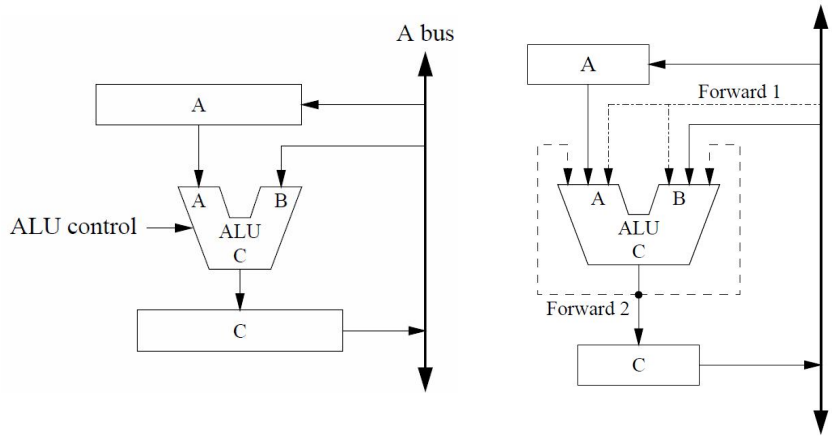


- техника 2:



Прослеђивање регистара

- основни пут података
- пут података са прослеђивањем регистара



Универзитет у Београду - Математички факултет

Прослеђивање регистара

- Након што инструкција израчуна резултат, он је доступан на излазу из АЛЈ
- Међутим, уз уобичајен ток података, он се не може користити још неколико циклуса
  - најпре се уписује у *C*
  - затим се преписује из *C* у одговарајући регистар
  - затим може да се користи – (по потребисе препише у *A*)
- У случају прослеђивања регистара
  - путем 1, из *C* на улазне гране АЛЈ или регистар *A*
    - штеди се један циклус, *WB* → *OF*
  - путем 2, са излаза АЛЈ на на улазне гране АЛЈ или регистар *A*
    - штеде се два циклуса, *IE* → *OF*

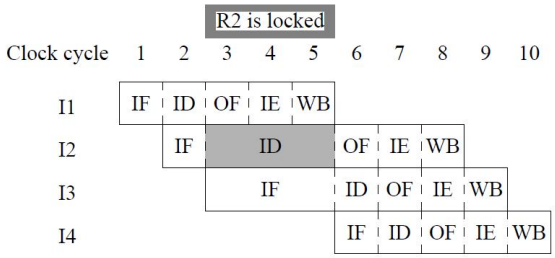
Универзитет у Београду - Математички факултет

Закључавање регистара

- Уопштена техника за решавање проблема исправности података
- Сваком регистру се додаје бит који означава исправност
  - ако је вредност 0, регистар је исправан и може се користити
  - ако је вредност 1, регистар је закључан и не сме се користити

Закључавање регистара

- Пример:
  - инструкција 1 закључава регистар R2 од 3. до 5. циклуса
  - инструкција 2 мора да чека на откључавање регистра



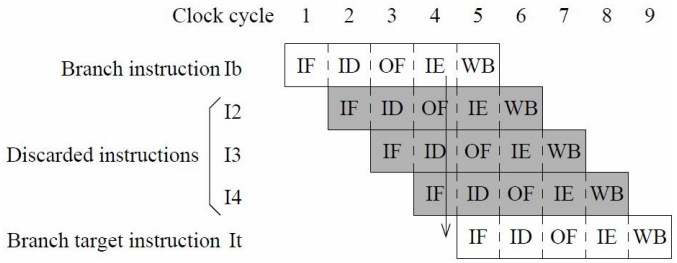
Однос техника

- Прослеђивање регистара
  - може да омогући више перформансе
  - има ефекта само ако су потребне вредности већ у току података
- Закључавање регистара
  - може да се примењује у општем случају
  - не повећава перформансе већ само пружа безбедност

Универзитет у Београду - Математички факултет

Проблем гранања

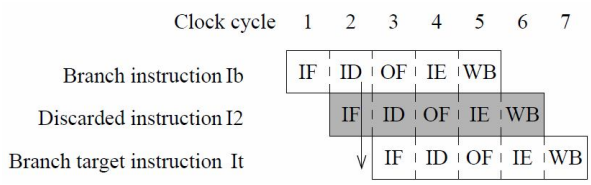
- Пример:
  - инструкција *Ib* је инструкција гранања
  - инструкција *It* је циљна инструкција
  - већ извршени делови инструкција *I2*, *I3*, *I4* се одбацују
    - тзв. “цена гранања”



Универзитет у Београду - Математички факултет

Решавање проблема гранања

- Гранања могу да утичу неповољно на преклапање
- Како смањити цену гранања
  - да се не чека да инструкција гранања дође до завршне фазе (*WB*) да би се започела циљна инструкција
  - на пример, ако се циљна инструкција израчуна већ у фази декодирања (*ID*), штеде се два циклуса



Универзитет у Београду - Математички факултет

Решавање проблема гранања (2)

- Шта је потребно да би било могуће већ у фази декодирања сазнати циљну адресу?
  - да се циљна адреса прочита већ у фази читања инструкције
    - ово је врло чест случај, посебно у случају релативних гранања
  - да се услов гранања може проверити већ у фази декодирања
    - тј. да не укључује додатне операнде

Универзитет у Београду - Математички факултет

Алтернативе

- Одложено гранање
  - већ смо о томе говорили...
  - обично се изводи од стране преводиоца (или асемблера)
    - није потребно мануелно
- Предвиђање гранања
  - идеја је да се са што вишом вероватноћом претпостави какав ће бити исход гранања
  - три врсте предвиђања
    - фиксно
    - статичко
    - динамичко

Фиксно предвиђање гранања

- Идеја је да се увек врши иста врста процене:
  - да до гранања никада неће доћи или
  - да ће до гранања увек доћи
- Пример:
  - *Motorola 68020*, *VAX 11/780* су користили приступ “никада неће бити гранања”
    - процесор наставља да чита и припрема инструкције редом
    - тиме се минимизује цена грешке ако исход буде супротан
      - читање наредних инструкција је јевтинија варијанта
    - ипак, није добар приступ у случају петље
  - ако би се користио приступ “увек ће бити гранања”
    - наредна инструкција би се увек читала са циљне адресе
    - можда уз грешку страничења (веома скупо)

Статичко предвиђање гранања

- Предвиђање се ради на основу кода операције
  - безусловно гранање се “увек” дешава
  - условно гранање се “никада” не дешава
  - гранање петље се “увек” дешава
  - позив процедуре и повратак се “увек” дешавају
- Даје релативно добре резултате

Универзитет у Београду - Математички факултет

Статичко предвиђање гранања

- Ако се знају учесталости појављивања инструкција и њихов исход, може се израчунати тачност процене:

Instruction type	Instruction distribution (%)	Prediction: Branch taken?	Correct prediction (%)
Unconditional branch	70 × 0.4 = 28	Yes	28
Conditional branch	70 × 0.6 = 42	No	42 × 0.6 = 25.2
Loop	10	Yes	10 × 0.9 = 9
Call/return	20	Yes	20

Overall prediction accuracy = 82.2%

Универзитет у Београду - Математички факултет

Динамичко предвиђање гранања

- Стратегије динамичког предвиђања почивају на прикупљању података о претходним гранањима
  - памти се претходних n исхода
    - за сваки од типова гранања или
    - за конкретне инструкције гранања
  - на основу претходног искуства (већина исхода) се предвиђа наредно понашање
- Више прикупљених података обезбеђује већу тачност предвиђања али и подиже цену
- Испоставља се да је довољно памтити претходна два исхода за веома добро предвиђање (изнад 90%)

Универзитет у Београду - Математички факултет

Динамичко предвиђање гранања

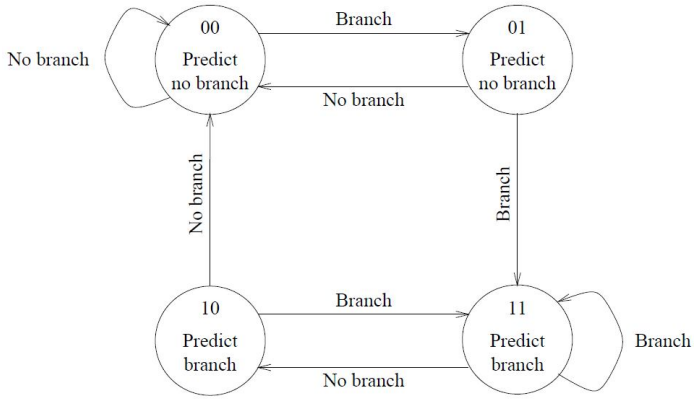
- Пример резултата статистичког истраживања:

	Type of mix		
n	Compiler	Business	Scientific
0	64.1	64.4	70.4
1	91.9	95.2	86.6
2	93.3	96.5	90.8
3	93.7	96.6	91.0
4	94.5	96.8	91.8
5	94.7	97.0	92.0

Универзитет у Београду - Математички факултет

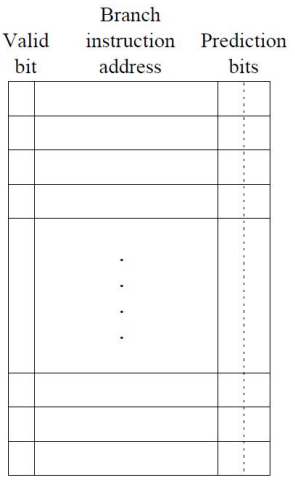
Пример имплементације

- Коначни аутомат са 4 стања:



Памћење претходног понашања

- Посебан напредак се остварује ако се претходни исходи не памте на нивоу врсте инструкције већ за појединачне инструкције условног гранања



Избор адресе

- Проблем са предвиђањем наступа у случају индиректног адресирања
  - тада адреса није непосредно доступна и није тривијално почети пуњење тока инструкција
- Једно решење је да се прави интерна таблица циљних адреса које су одговарале претходним гранањима
  - тада, бар у случају када је скок на исту адресу као раније, успешно предвиђање скока има за резултат и исправну циљну адресу

Универзитет у Београду - Математички факултет

Памћење циљне адресе

Valid bit	Branch instruction address	Target address	Prediction bits
	. . . .	. . . .	

Универзитет у Београду - Математички факултет

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