Еволуција на компјутерите

Chapter 2
Computer Evolution and Performance

ENIAC

- Electronic Numerical Integrator And Computer
- Eckert и Mauchly
- University of Pennsylvania
- Табели за траектории на оружја
- Започнат 1943
- Завршен 1946
  - Предоцна за војните напори
- Се користи се до 1955

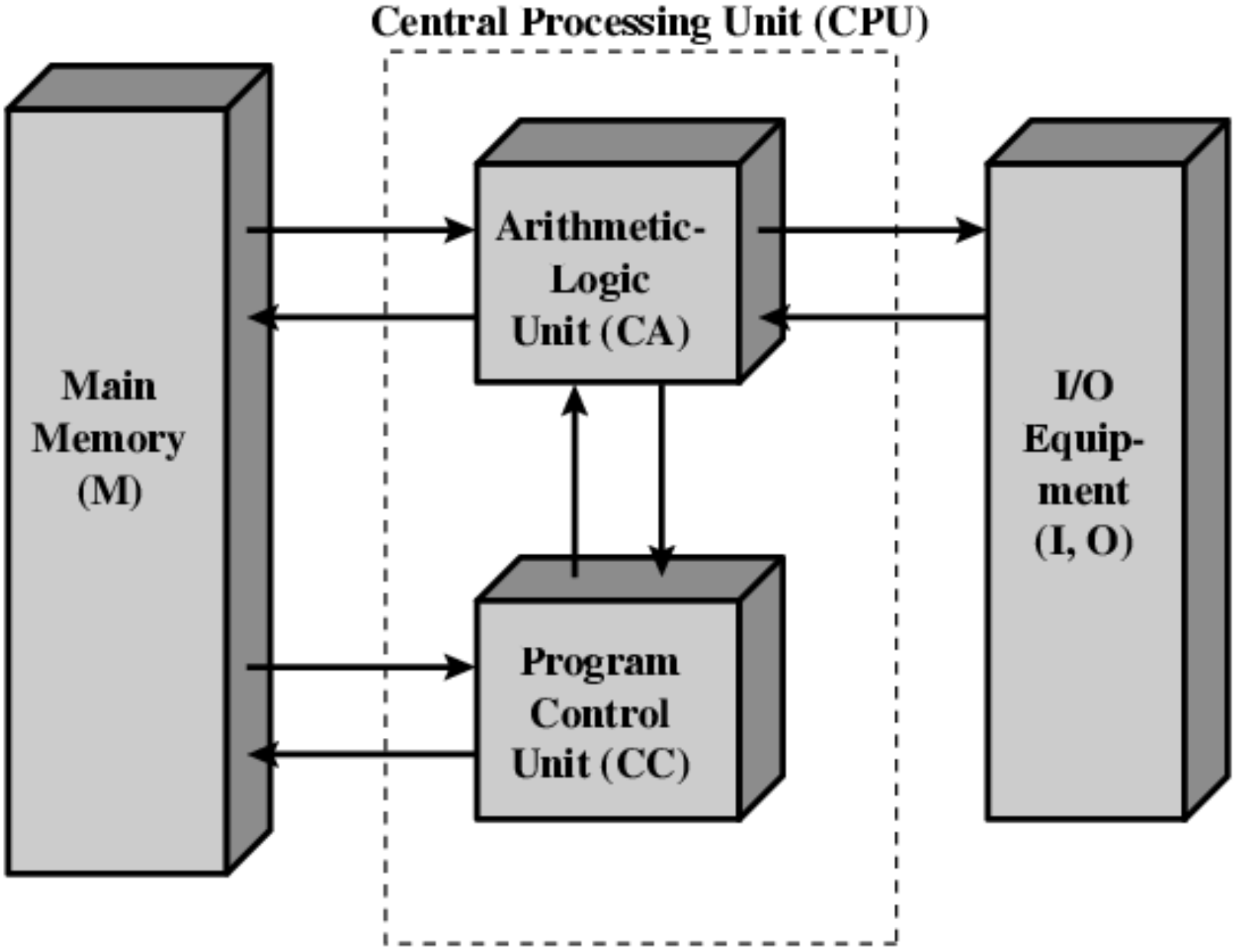
ENIAC - детали

- Декаден (не бинарен)
- 20 акумулатори со по 10 цифри
- Се програмирал рачно со прекинувачи
- 18,000 вакуум цевки
- 30 тони
- 140 m^2
- 140 kW потрошувачка
- 5,000 собирања во секунда

von Neumann/Turing

- Концепт на складирана програма (Stored Program)
- Во главната меморија се чуваат програмите и податоците
- ALU работи врз бинарни податоци
- Контролната единица ги интерпретира инструкциите од меморијата и ги извршува
- В/И опрема се контролира преку контролната единица
- Princeton Institute for Advanced Studies
  - IAS
  - Завршен 1952

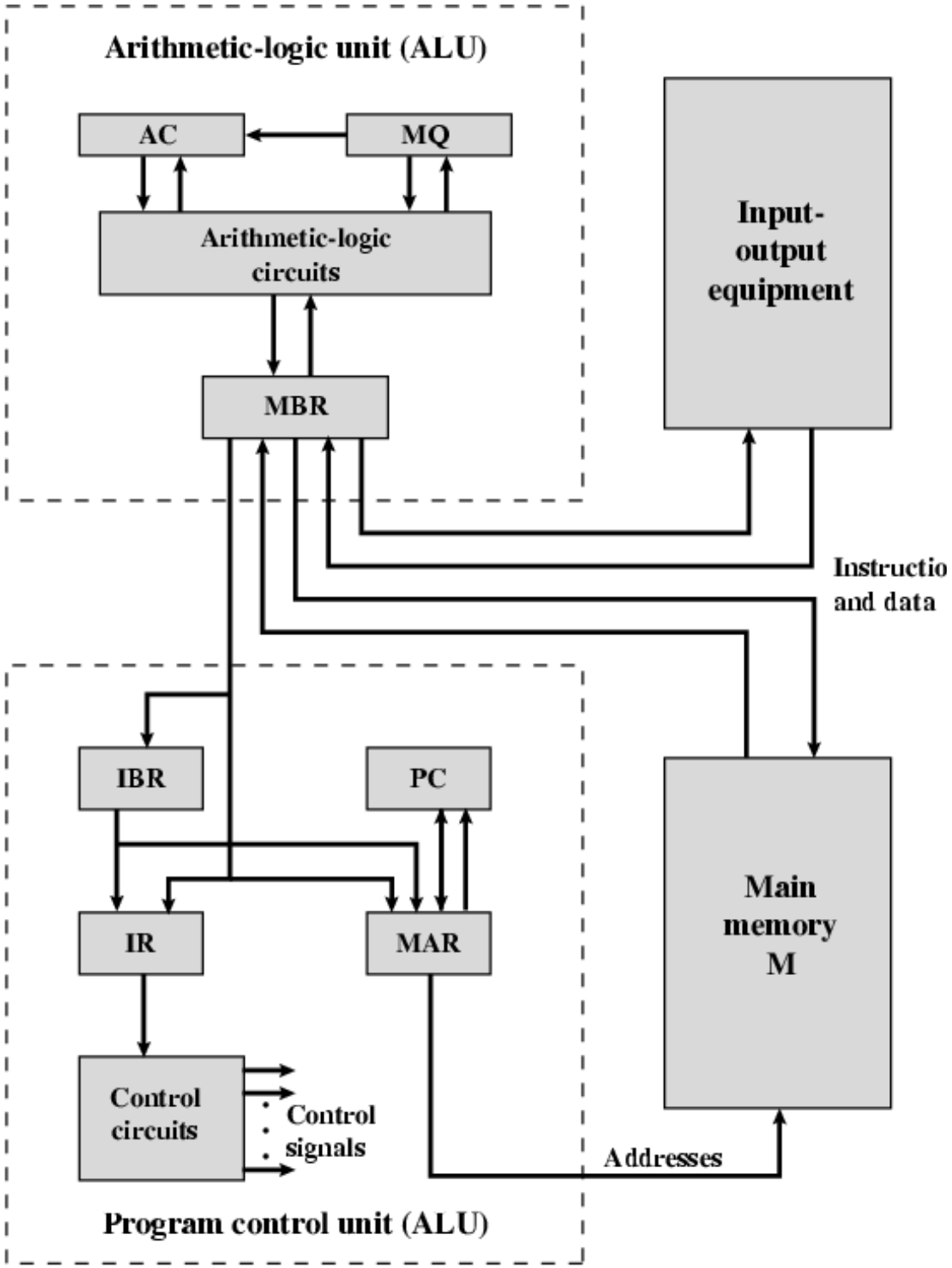
Структура на von Neumann машина



IAS - детали

- 1000 x 40 bit зборови
  - Бинарен број
  - 2 x 20 bit инструкции
- Множество регистри (складишта во CPU)
  - Memory Buffer Register
  - Memory Address Register
  - Instruction Register
  - Instruction Buffer Register
  - Program Counter
  - Accumulator
  - Multiplier Quotient

Структура на IAS – детали



Комерцијални компјутери

- 1947 - Eckert-Mauchly Computer Corporation
- UNIVAC I (Universal Automatic Computer)
- US Bureau of Census во 1950 за пресметки
- Станува дел од Sperry-Rand Corporation
- Доцни 1950ти - UNIVAC II
  - Побрз
  - Поголема меморија
  - Backward compatible

IBM

- Опрема за обработка на дупчени картички
- 1953 - 701
  - Првиот компјутер со складирана програма на IBM
  - Научни пресметки
- 1955 - 702
  - Бизнис апликации
- Води кон 700/7000 серијата

2Г - Транзистори

- Ги заменуваат вакуумските цевки
- Помали
- Поефтини
- Помалку дисипација на топлина
- Solid State device
- Направен од силициум
- Измислен во 1947 во Bell Labs
- William Shockley и други

Компјутери базирани на транзистори

- Втора генерација машини
- NCR & RCA произведуваат мали транзисторски машини
- IBM 7000
- Појава на системски софтвер
- DEC - 1957
  - Го произвел PDP-1

Микроелектроника

- Компјутерот е направен од порти, мемориски ќелии и меѓуповрзувања
- Сите овие може да се произведат на полупроводник
- Пример силициумска плочка

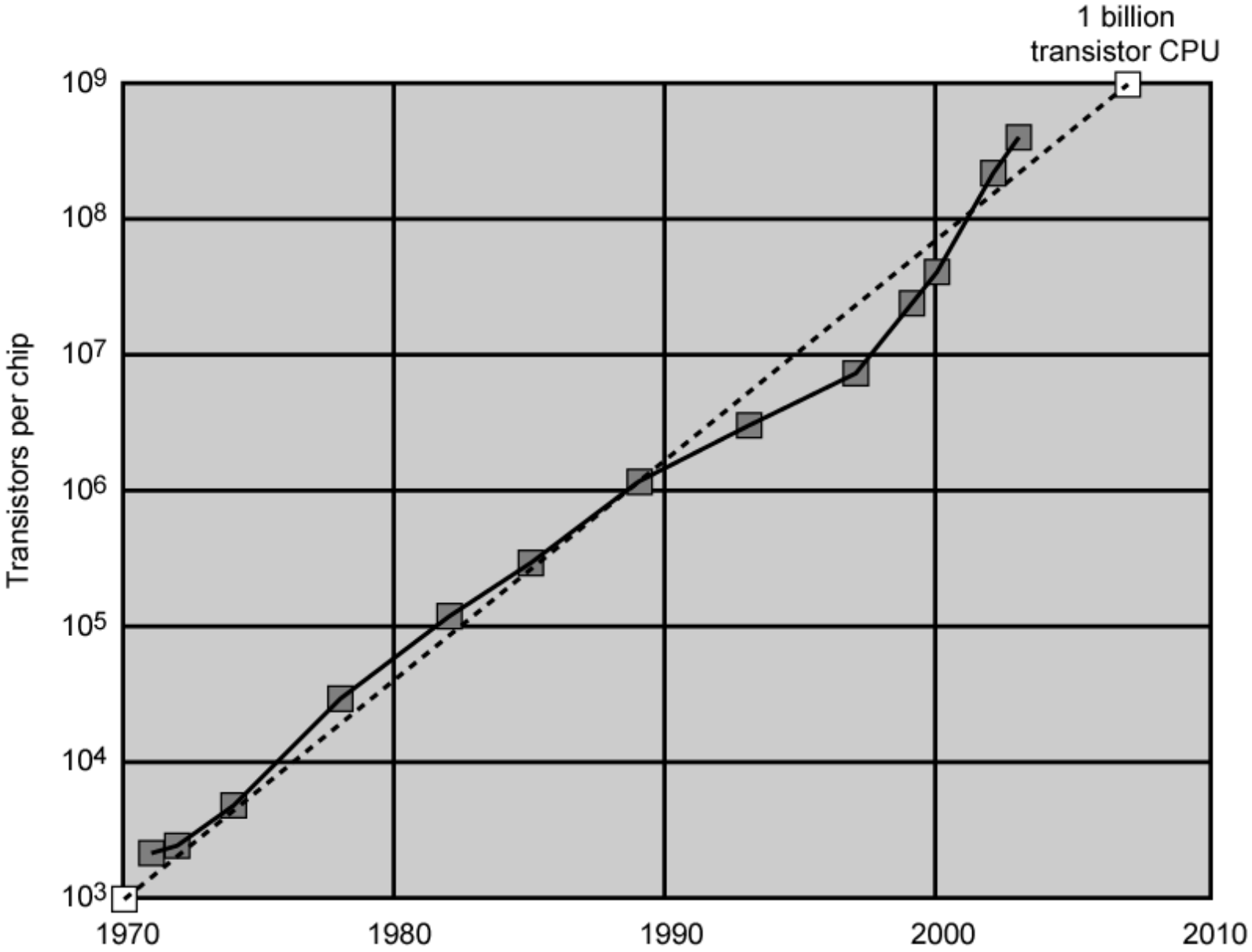
Генерации компјутери

- Вакуумски цевки- 1946-1957
- Транзистори- 1958-1964
- Small scale integration - 1965 натаму
  - До 100 уреди на чип
- Medium scale integration - до 1971
  - 100-3,000 уреди на чип
- Large scale integration - 1971-1977
  - 3,000 - 100,000 уреди на чип
- Very large scale integration - 1978 -1991
  - 100,000 - 100,000,000 уреди на чип
- Ultra large scale integration – 1991 -
  - над 100,000,000 уреди на чип

Муров закон

- Зголемена густина на компонентите на чип
- Gordon Moore – ко-основач на Intel
- Бројот на транзистори на чип ќе се дуплира секоја година
- од 1970's развојот малку е намален
  - Бројот на транзистори на чип се дуплира секои 18 месеци
- Цената на чипот останува скоро непроменета
- Поголема густина на пакување значи пократки електрични патеки и подобри перформанси
- Помала големина значи зголемена флексибилност
- Намалена моќност и потреба за ладење
- Помалку поврзувања ја зголемуваат надежноста

Раст во бројот на транзистори на CPU



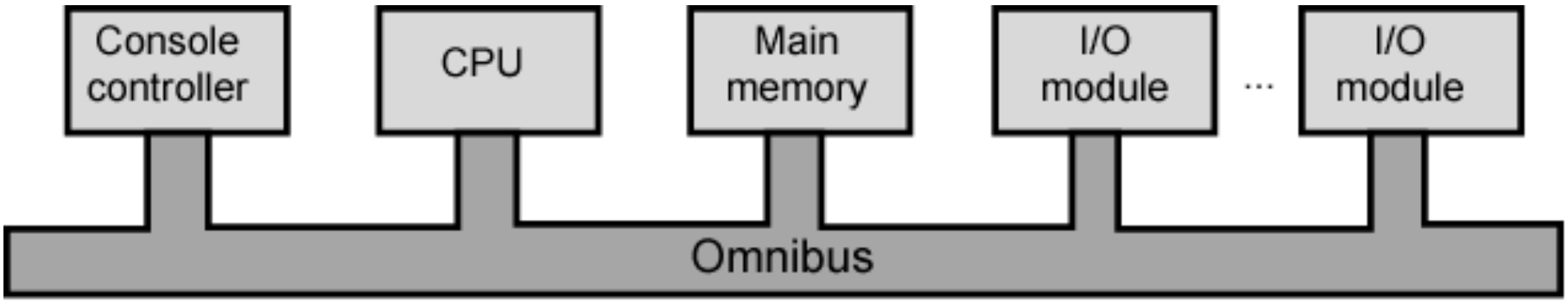
IBM 360 серија

- 1964
- Заменета со (& не-компатибилна со) 7000 серија
- Прва планирана „фамилија“ компјутери
  - Слични или идентични инструкциски множества
  - Слични или идентични O/S
  - Зголемена брзина
  - Зголемен број на I/O порти (повеќе терминали)
  - Зголемена меморија
  - Зголемена цена
- Multiplexed switch structure

DEC PDP-8

- 1964
- Првиот мини компјутер
- Не барал климатизирана соба
- Доволно мал да се стави на лабораториска клупа
- $16,000
  - $100k+ за IBM 360
- Вградливи апликации & OEM
- Структура на магистрала (BUS STRUCTURE)

DEC - PDP-8 Bus Structure



Полупроводничка меморија

- 1970
- Fairchild
- Големина на едно јадро
  - 1 bit од складирањето во магнетното јадро
- содржи 256 бита
- Не-деструктивно читање
- Побрза од јадрото
- Капацитетот приближно x 4 за секоја нова генерација

Intel

- 1971 - 4004
  - Прв микропроцесор
  - Сите CPU компоненти на еден чип
  - 4 bit
- По него во 1972 следи 8008
  - 8 bit
  - Двата дизајнирани за специфични апликации
- 1974 - 8080
  - Првиот микропроцесор за општа намена на Intel

Забрзување

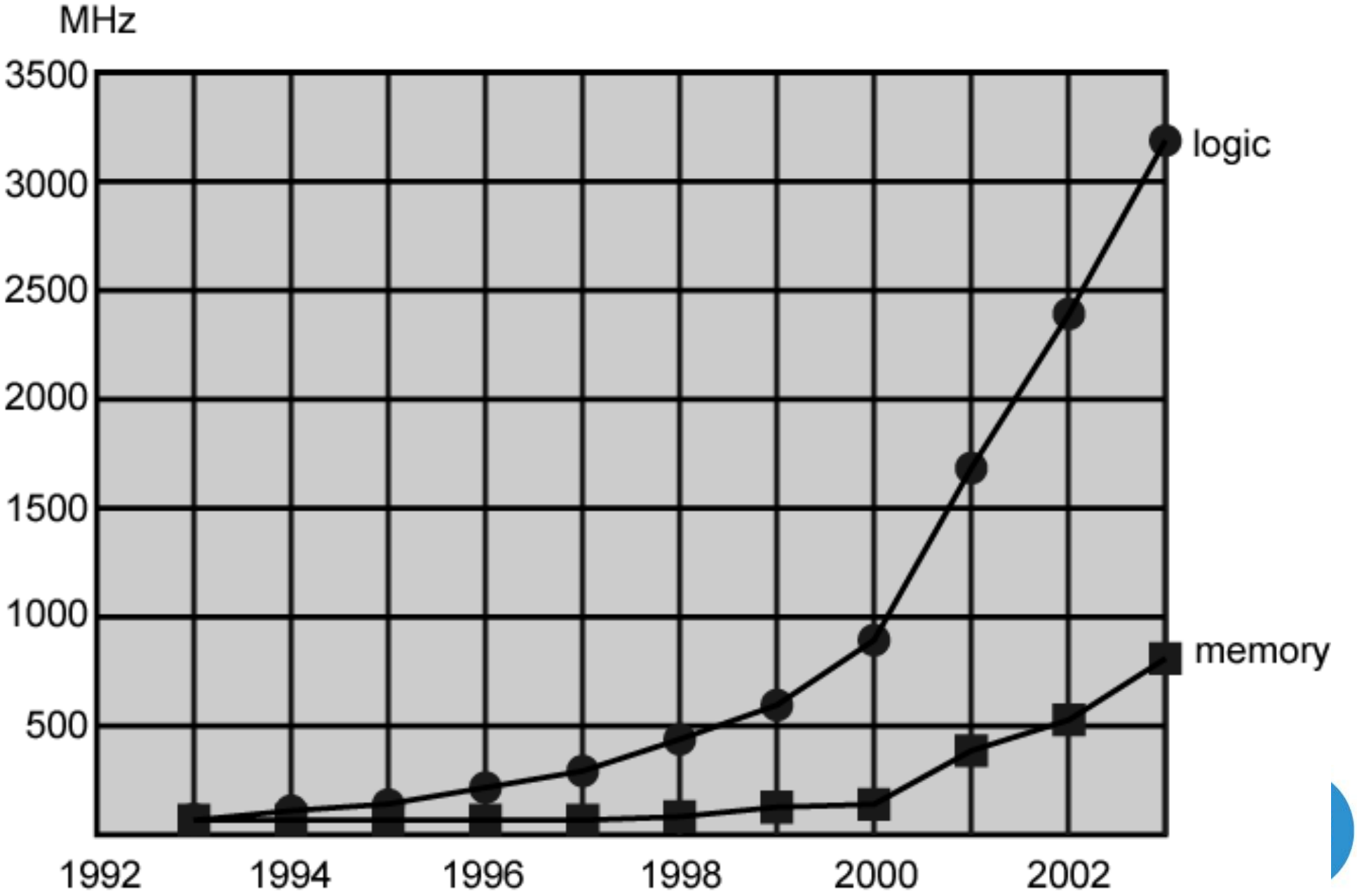
- Pipelining
- Branch prediction
- Анализа на текот на податоците
- Шпекулативно извршување (Speculative execution)

Балансирање на перформансите

- Зголемена брзина на процесорот
- Зголемен капацитет на меморијата

- Брзината на меморија заостанува зад брзината на процесорот

Разлики во перформансите



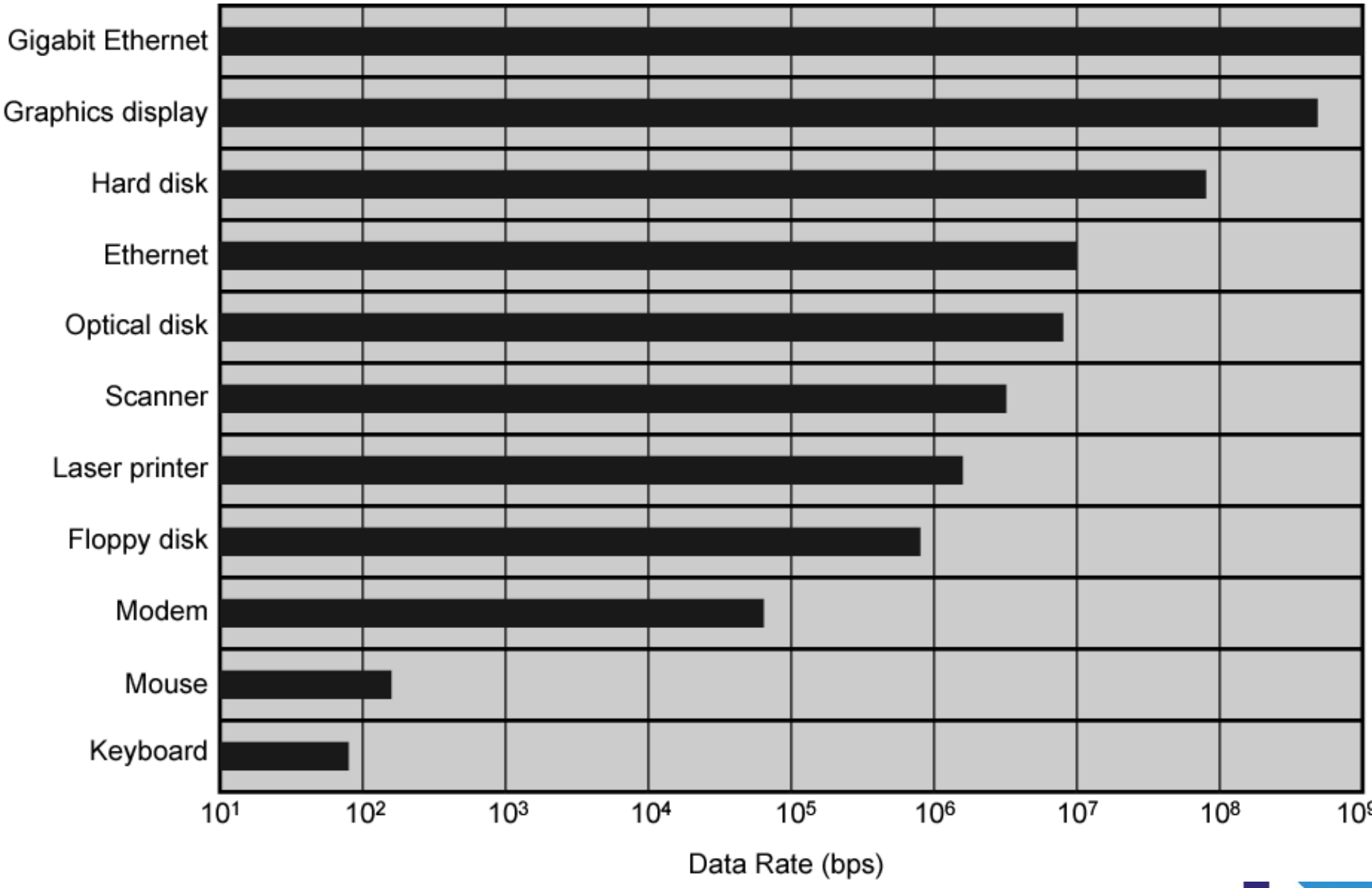
решенија

- Зголемен број на битови кои се преземаат во единица време
  - DRAM да биде “поширок” а не “подлабок”
- Промена на DRAM интерфејсот
  - Кеш
- Намалување на фреквенцијата на пристап до меморијата
  - Покомплексен кеш (L1, L2) и кеш на чип
- Зголемен пропусен опсег на меѓуповрзувањето
  - Магистрали со голема брзина
  - Хиерархија на магистрали

I/O уреди

- Периферии со интензивни I/O барања
- Барања за голема пропусност на податоци
- Процесорот може да се справи со ова
- Проблемот е преместувањето на податоци
- Решенија:
  - Кеширање
  - Баферирање
  - Магистрали со големи брзини
  - Посложена структура на магистралите
  - Конфигурации со повеќе процесори

Типични податочни брзини на I/O уреди



Клучот е во балансирањето

- Процесорски компоненти
- Главна меморија
- I/O уреди
- Меѓуповрзувачки структури

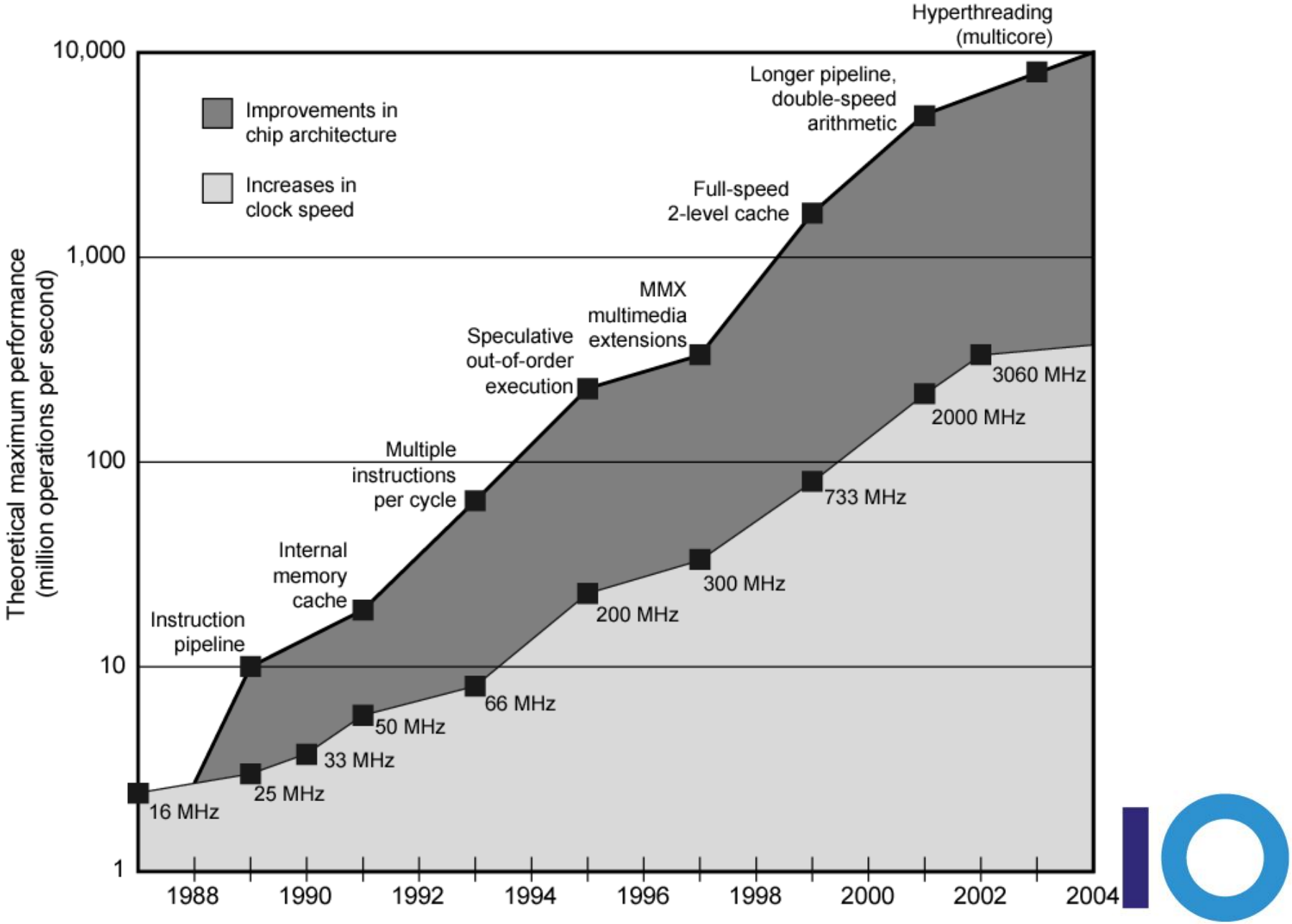
Подобрувања во организацијата и архитектурата на чипот

- Зголемување на хардверската брзина на процесорот
  - Основно поради намалување на големината на логичките порти
    - Повеќе порти спакувани поблиску, зголемена фреквенција на такт сигналот
    - Намалено време на пропагација на сигналите
- Зголемена големина и брзина на кешот
  - Посветување на дел од процесорскиот чип
    - Времето за пристап до кешот значајно се намалува
- Промена на организацијата и архитектурата на процесорот
  - Зголемување на ефективната брзина на извршување
  - Паралелизам

Проблеми со брзината на тактот и густината на логичките порти

- Моќност
  - Со зголемување на густината на логиката и брзината на тактот се зголемува и густината на моќноста
  - Дисипација на топлина
- RC доцнење
  - Брзината со која електроните се движат е ограничена со отпорноста и капацитивноста на металните жици кои ги поврзуваат
  - Доцнењето се зголемува со пораст на производот RC
  - Жиците се потенки, зголемена отпорност
  - Жиците се поблиску, зголемена капацитивност
- Латентност на меморијата
  - Меморијата заостанува зад процесорот во брзина
- Решение:
  - Повеќе внимание на организацијата и архитектурата

Перформанси на Intel микропроцесорите



Зголемен капацитет на кешот

- Типично 2 или 3 нивоа на кеш меѓу процесорот и главната меморија
- Зголемување на густината на чипот
  - Повеќе кеш меморија на чип
    - Побрз пристап до кешот
- Pentium чипот посветил околу 10% од областа на кеш
- Pentium 4 посвети околу 50%

Покомплексна логика на извршување

- Овозможување паралелно извршување на инструкциите
- Проточноста работи како линија за производство
  - Различните етапи на извршување на различни инструкции во исто време низ проточната цевка
- Суперскаларната архитектура овозможува повеќе проточни цевки во еден процесор
  - Инструкциите кои не зависат една од друга може да се извршат паралелно

Diminishing Returns

- Комплексна внатрешна организација на процесорот
  - Веќе се добива голем паралелизам
  - Идните значајни зголемувања ќе бидат најверојатно скромни
- Придобивките од кешот доаѓаат до крајната граница
- Зголемената брзина на тактот влегува во проблем со дисипација на моќноста
  - Достигнати се некои фундаментални физички ограничувања

Нов пристап – повеќе јадра

- Повеќе процесори на еден чип
  - Голем споделен кеш
- Во процесорот, зголемување на перформансите пропорционално на квадратен корен од зголемување на комплексноста
- Ако софтверот може да користи повеќе процесори, дуплирањето на бројот на процесори скоро ги дуплира перформансите
- Така, се користат 2 поедноставни процесори на чипот наместо еден многу покомплексен
- Со 2 процесори се оправдува поголем кеш
  - Потрошувачката на моќност на мемориската логика е помала од таа на процесирачката логика

x86 еволуција (1)

- 8080
  - first general purpose microprocessor
  - 8 bit data path
  - Used in first personal computer – Altair
- 8086 – 5MHz – 29,000 transistors
  - much more powerful
  - 16 bit
  - instruction cache, prefetch few instructions
  - 8088 (8 bit external bus) used in first IBM PC
- 80286
  - 16 MB memory addressable
  - up from 1Mb
- 80386
  - 32 bit
  - Support for multitasking
- 80486
  - sophisticated powerful cache and instruction pipelining
  - built in maths co-processor

x86 еволуција (2)

- Pentium
  - Superscalar
  - Multiple instructions executed in parallel
- Pentium Pro
  - Increased superscalar organization
  - Aggressive register renaming
  - branch prediction
  - data flow analysis
  - speculative execution
- Pentium II
  - MMX technology
  - graphics, video & audio processing
- Pentium III
  - Additional floating point instructions for 3D graphics

x86 еволуција (3)

- Pentium 4
  - Note Arabic rather than Roman numerals
  - Further floating point and multimedia enhancements
- Core
  - First x86 with dual core
- Core 2
  - 64 bit architecture
- Core 2 Quad – 3GHz – 820 million transistors
  - Four processors on chip

- x86 architecture dominant outside embedded systems
- Organization and technology changed dramatically
- Instruction set architecture evolved with backwards compatibility
- ~1 instruction per month added
- 500 instructions available

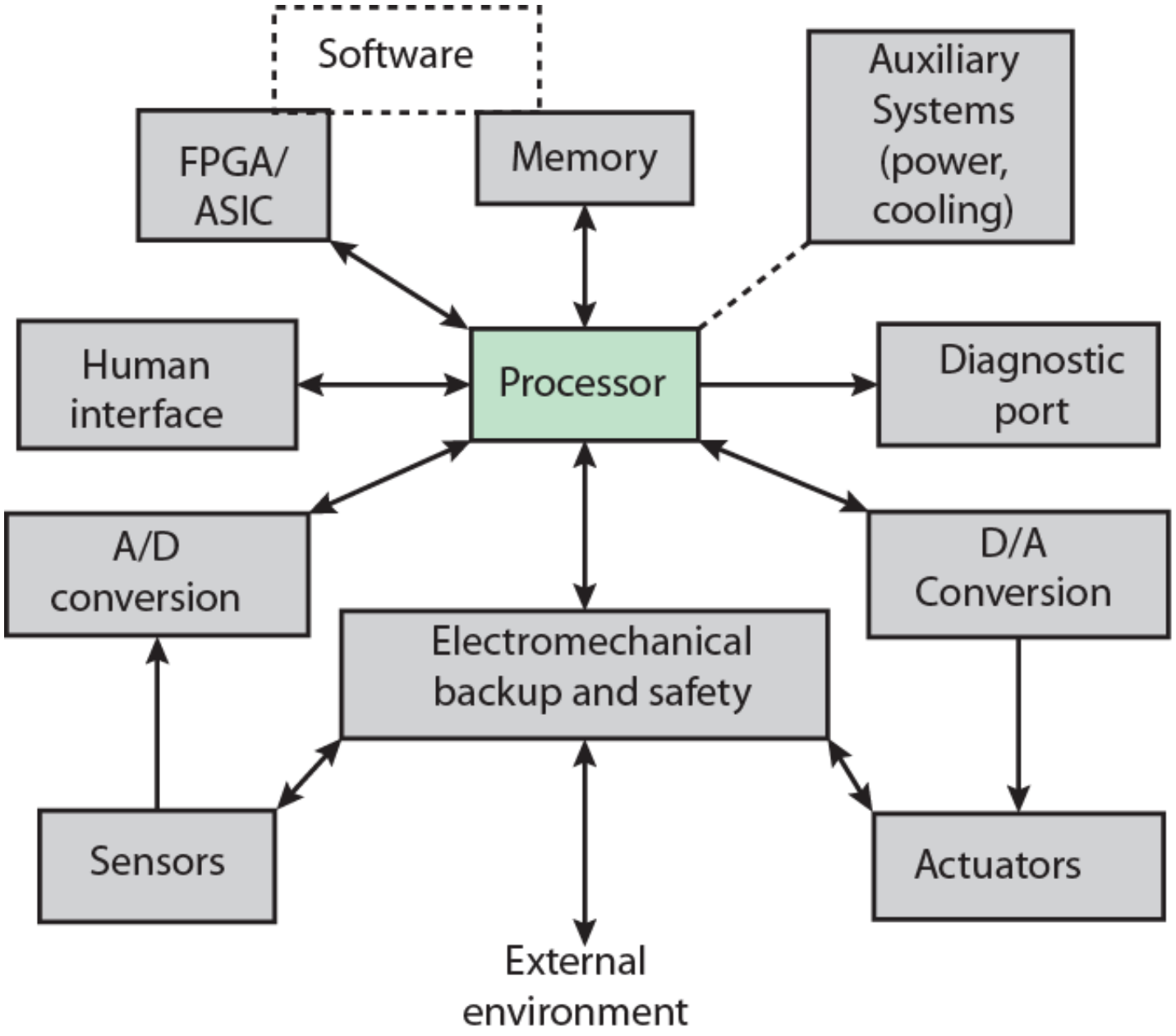
Вградливи системи ARM

- ARM развиен според RISC дизајнот
- Се користи главно во вградливи системи
  - Во рамките на продуктот
  - Не е компјутер за општа намена
  - Има специјализирана функција
  - Пример систем против блокирање на сопирачките во кола

Барања на вградливите системи

- Различни големини
  - Различни ограничувања, оптимизација, повторна употреба
- Различни барања
  - Безбедност, надежност, реално време, флексибилност
  - Животен век
  - Услови во средината
  - Статичен наспроти динамичен товар
  - Бавни наспроти брзи
  - Пресметки наспроти В/И барања
  - Дискретно настански наспроти континуални

Можна организација на вградлив систем



ARM еволуција

- Дизајниран од ARM Inc., Cambridge, England
- Лиценциран на производителите
- Голема брзина, мал, мала потрошувачка
- PDAs, рачни играчки конзоли, телефони
  - пример. iPod, iPhone
- Acorn произвел ARM1 & ARM2 во 1985 и ARM3 во 1989
- Acorn, VLSI и Apple Computer го засноваат ARM Ltd.

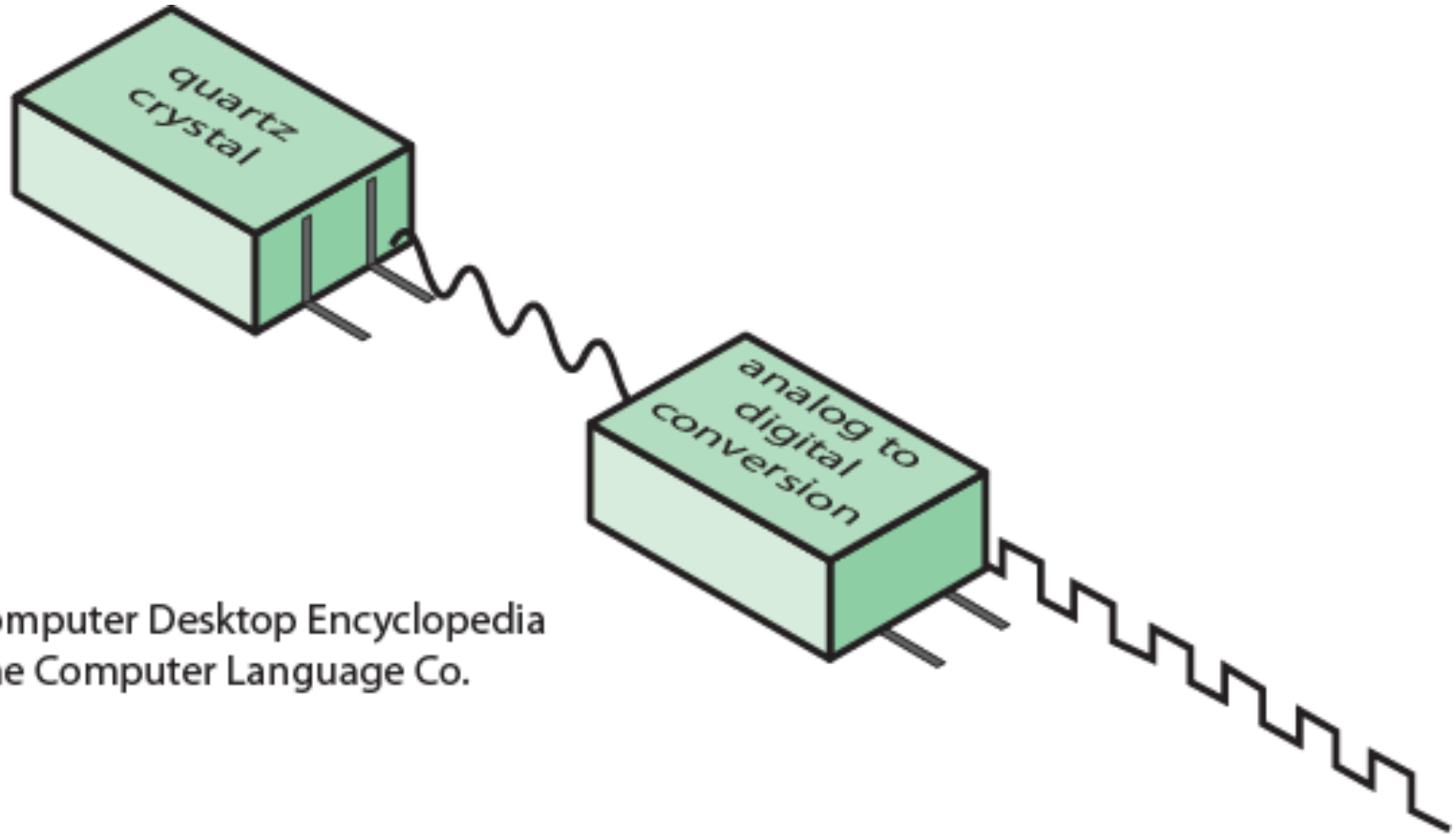
Категории на ARM системи

- Вградливи во реално време
- Апликациски платформи
  - Уреди кои работат на отворени платформи како Linux, Palm OS, Symbian OS, Windows mobile
- Безбедносни апликации

Одредување на перформансите
Брзина на тактот (clock speed)

- Клучни параметри
  - перформанси, цена, големина, безбедност, надежност, потрошувачка на моќност
- Брзина на системскиот такт
  - во Hz
  - Clock rate, clock cycle, clock tick, cycle time
- На сигналите во CPU им треба време да се стабилизираат на 1 или 0
- Сигналите може да се менуваат со различни брзини
- Операциите мора да се синхронизирани
- Инструкцијата се извршува во дискретни чекори
  - Fetch, decode, load и store, arithmetic или logical
  - Обично повеќе такт циклуси по инструкција
- Проточноста овозможува едновремено извршување на инструкции
- Затоа, **брзината на тактот не е целата приказна**

System Clock



From Computer Desktop Encyclopedia
1998, The Computer Language Co.

Брзина на извршување инструкции

- Millions of instructions per second (MIPS)
- Millions of floating point instructions per second (MFLOPS)
- Зависни од инструкциското множество, дизајнот на компајлерот, имплементацијата на процесорот, кешот и мемориската хиерархија

Benchmarks

- Програми дизајнирани за тестирање на перформансите
- Напишани во јазик на високо ниво
  - портабилни
- Претставувааат стилови на работење
  - Системски, нумерички, комерцијални
- Лесно мерливи
- Широко дистрибуирани
- Пример, System Performance Evaluation Corporation (SPEC)
  - CPU2006 за пресметковни можности
    - 17 floating point програми во C, C++, Fortran
    - 12 integer програми во C, C++
    - 3 милиони линии код
  - Метрики за брзина и фреквенција
    - Една задача и пропустливост

SPEC метрика за брзина

- Една задача
- Основно работење дефинирано за секој benchmark со користење референтна машина
- Резултатите се изразени како однос од референтното време со системското време на работа
  - $Tref_i$ време на извршување на benchmark i на референтната машина
  - $Tsut_i$ време на извршување на benchmark i на системот кој се тестира

$$r_i = \frac{Tref_i}{Tsut_i}$$

- Целокупните перформанси се добиваат како просек од односите за сите 12 integer benchmarks
  - Се користи геометриска средина
    - Соодветна за нормализирани броеви како односи

$$r_G = \left(\prod_{i=1}^{n} r_i \right)^{1/n}$$

SPEC метрика за пропусност - брзина

- Мери пропусност или брзина на машината со која се изведуваат одреден број задачи
- Повеќе копии на benchmarks работат симултано
  - Типично, колку што има процесори
- Односот се пресметува како:
  - $Tref_i$ референтно време за извршување на benchmark i
  - N број на копии кои работат симултано
  - Tsuti поминато време од почетокот на извршување на програмата на сите N процесори до завршување на сите копии од програмата
  - Пак се пресметува геометриска средина

$$r_i = \frac{N \times Tref_i}{Tsut_i}$$

Амдалов закон

- Gene Amdahl
- Потенцијално забрзување на програма со користење повеќе процесори
- Заклучил дека:
  - Кодот треба да може да се паралелизира
  - Забрзувањето е ограничено поради diminishing returns на повеќе процесори
- Зависно од задачата
  - Серверите добиваат со одржување на повеќе врски на повеќе процесори
  - Базите може да се поделат во паралелни задачи

Амдалов закон - формула

- За програма која работи на еден процесор
  - делот *f* од кодот е бесконечно паралелизабилен без дополнителен товар за распоредување
  - делот (1-*f*) од кодот е сериски по природа
  - T е вкупното време на извршување за програма на 1 процесор
  - N е бројот на процесори кои целосно ги користат паралелните делови од кодот

$$Speedup = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{T(1-f)+Tf}{T(1-f)+\frac{Tf}{N}} = \frac{1}{(1-f)+\frac{f}{N}}$$

- заклучоци
  - *f* мало, паралелните процесори имаат мал ефект
  - *N* ->∞, забрзувањето е ограничено од $1/(1-f)$
    - Diminishing returns за повеќе процесори

Глобален поглед на функцијата и меѓуповрзувањето на компјутерот

Chapter 3

Top Level View of Computer Function and Interconnection

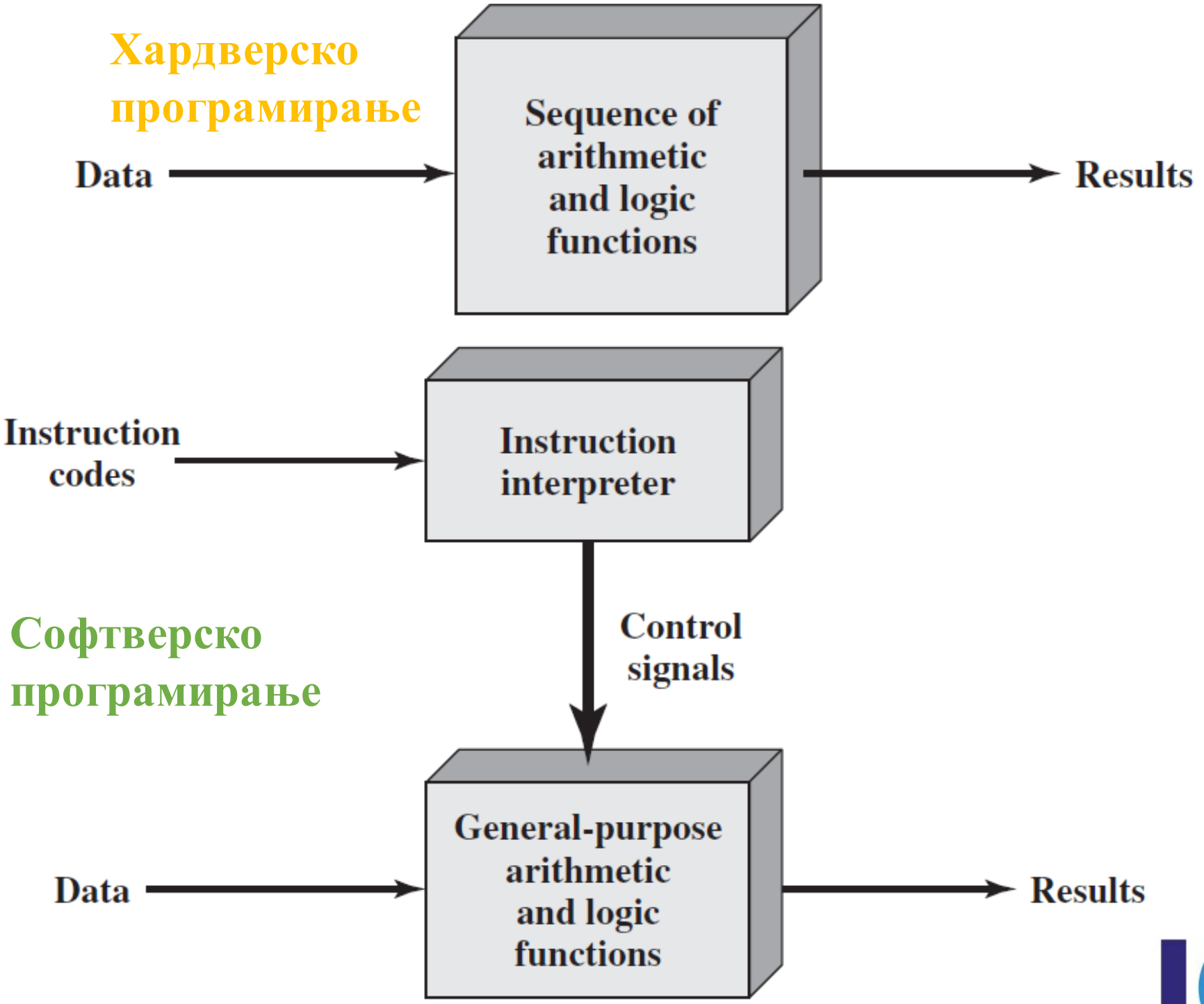
Концепт на програма

- Hardwired program = Системите со фиксна жичана врска не се флексибилни
- Хардвер за општа намена може да прави различни работи доколку се доведат точните контролни сигнали
- Наместо повторна организација на ожичувањето, се одредува ново множество на контролни сигнали
  - Софтверско програмирање

Што е програма?

- Низа од чекори
- Во секој чекор се изведува аритметичка или логичка операција
- За секоја операција потребно е друго множество контролни сигнали

Хардверско програмирање



Софтверско програмирање

Функција на контролната единица

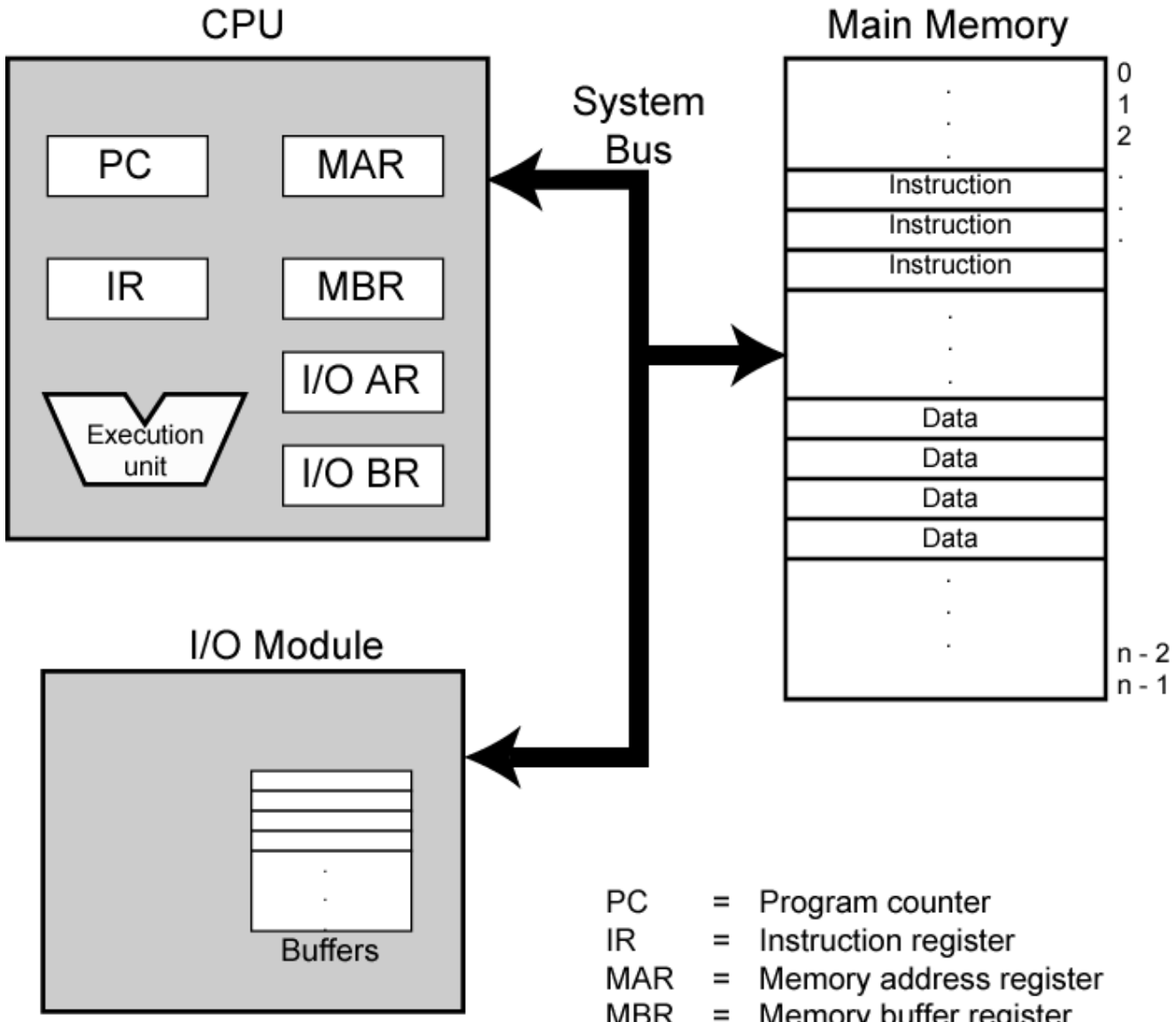
- За секоја операција се дава уникатен код - opcode
  - пр. ADD, MOVE
- Хардверскиот сегмент го прифаќа кодот и издава контролни сигнали

Компоненти

- Контролната единица и Аритметичко логичката единица ја сочинуваат Централната процесирачка единица
- Во системот влегуваат податоците и инструкциите, а излегуваат резултатите
  - Влез/излез
- Потребни се привремени складишта за кодот и резултатите
  - Главна меморија

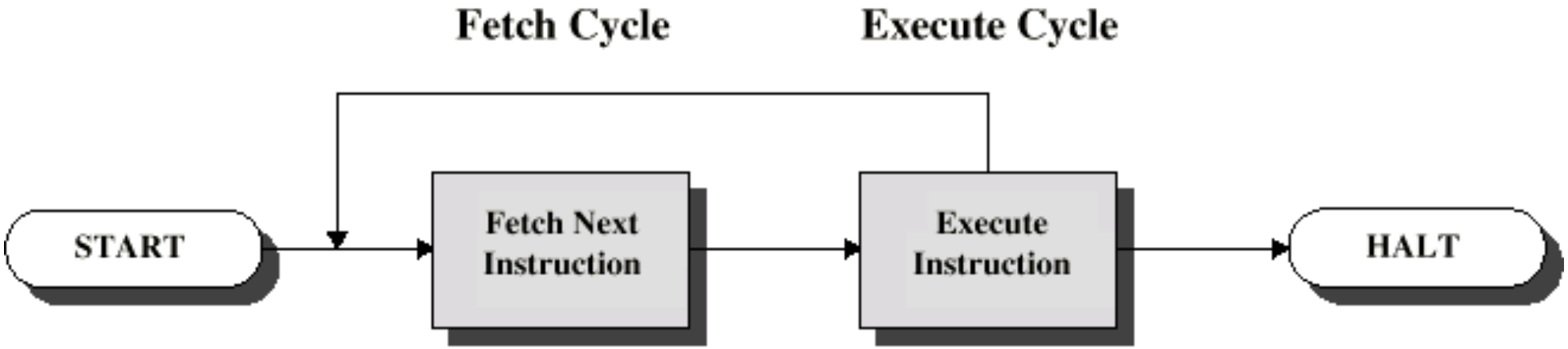
И така... добивме компјутер ☺

Компјутерски компоненти: глобален поглед



Инструкциски циклус

- Два чекори:
  - Земи - Fetch
    - Декодирај - decode
  - Изврши - Execute



Земи - Fetch циклус

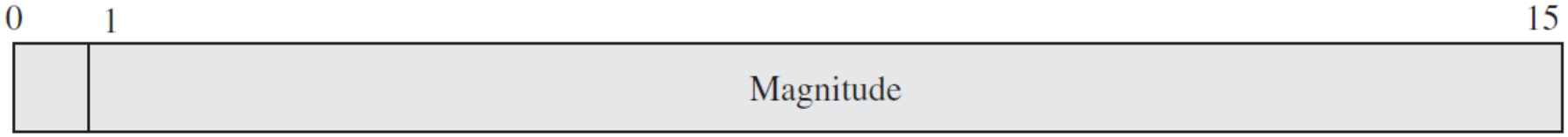
- Програмскиот бројач (PC) ја чува адресата на следната инструкција која треба да се земе
- Процесорот ја зема инструкцијата од мемориската локација кон која покажува PC
- Зголеми го PC за 1
  - Ако не е кажано поинаку
- Инструкцијата се запишува во инструкцискиот регистар (IR)
- Процесорот ја интерпретира инструкцијата и ги изведува бараните акции

Изврши - Execute циклус

- Процесор - меморија
  - Податочен трансфер меѓу CPU и главната меморија
- Процесор - В/И
  - Податочен трансфер меѓу CPU и I/O модул
- Обработка на податоци
  - Аритметичка или логичка операција врз податок
- Контрола
  - Промена на текот на низата операции
  - пр. jump
- Комбинација од горните

Пример на извршување на програма

(a) Instruction format



(b) Integer format

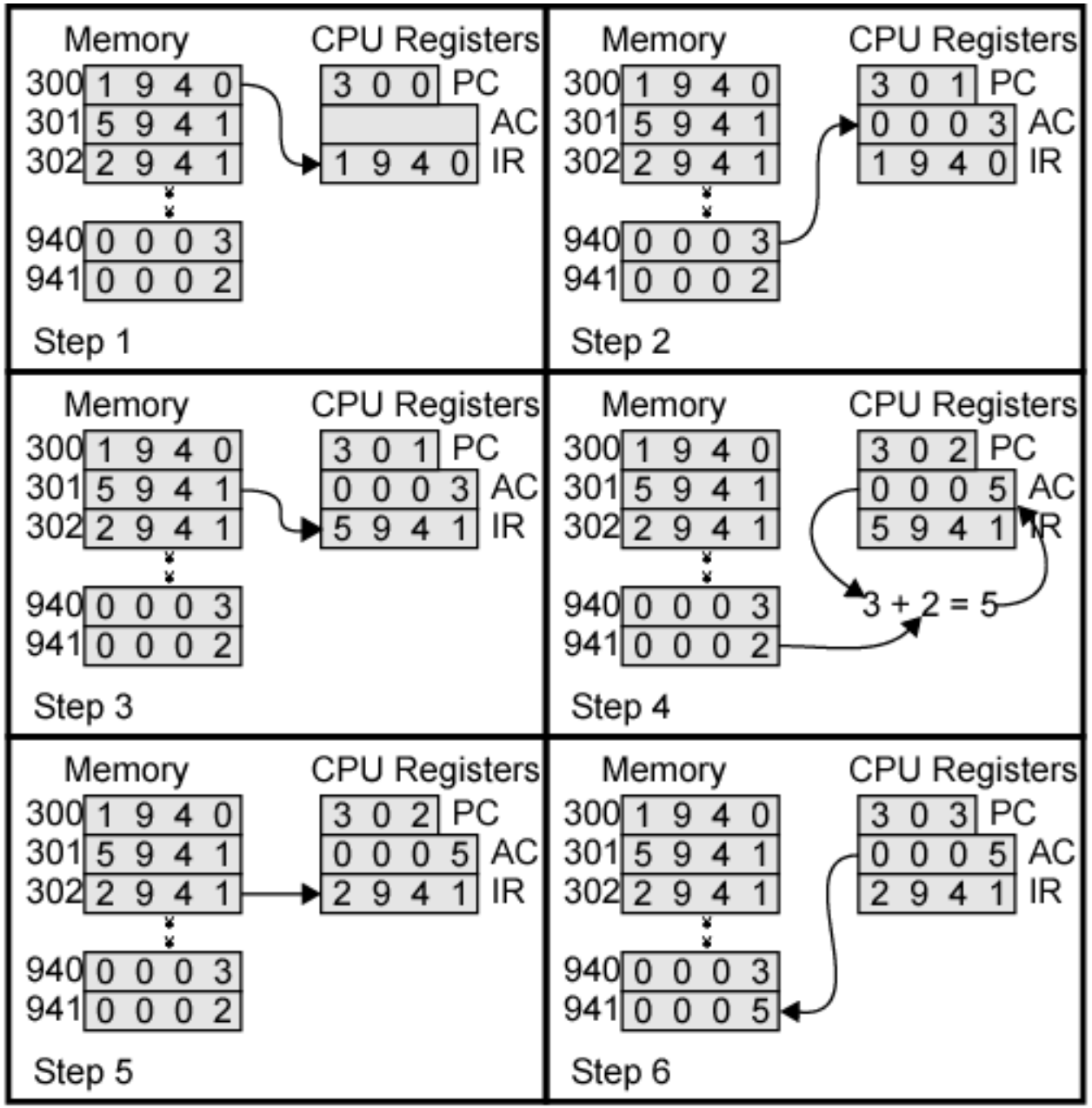
Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

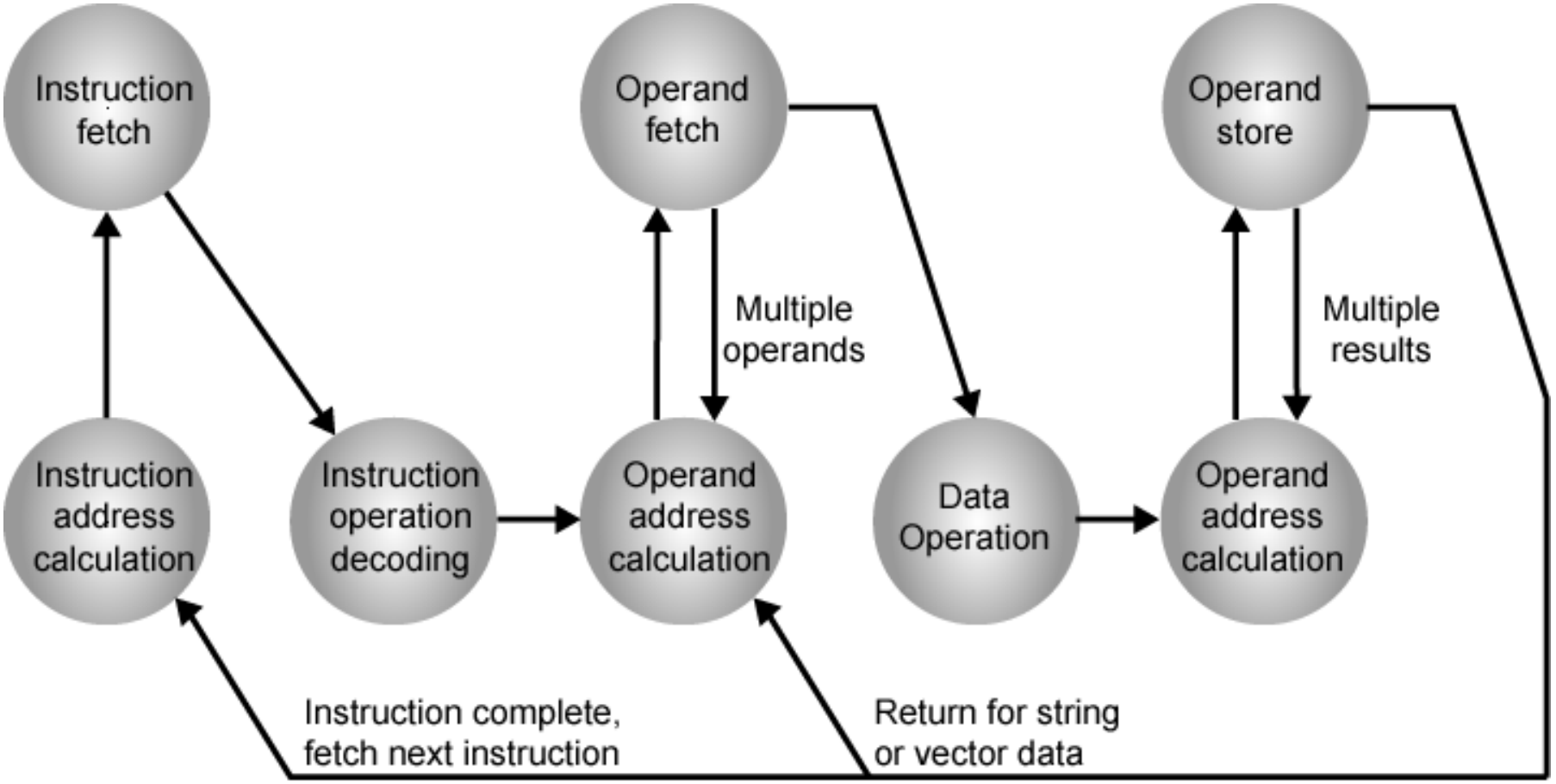
0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

Пример на извршување на програма



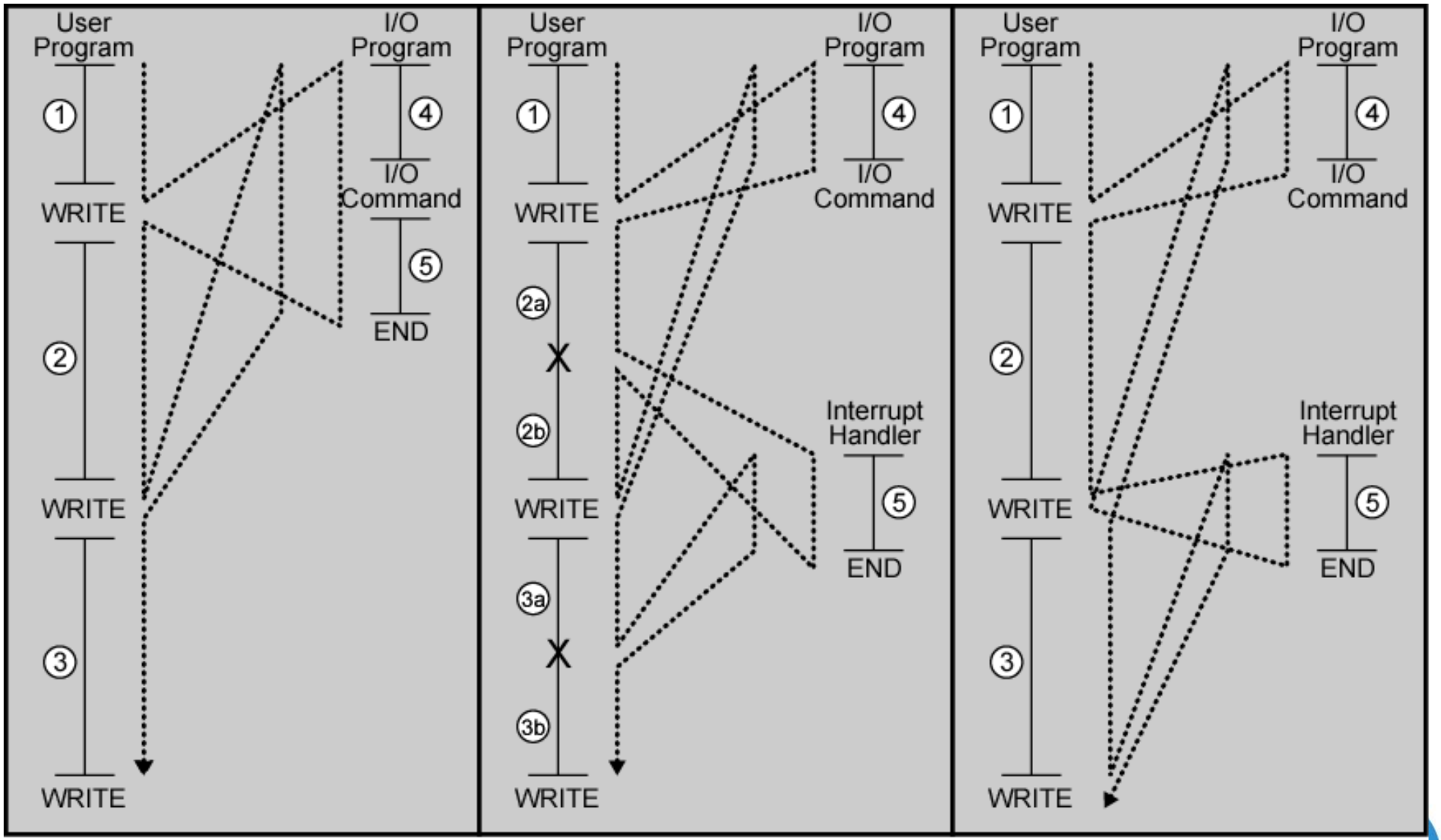
Инструкциски циклус – дијаграм на состојби



Прекини - Interrupts

- Механизам со кој други модули (пр. В/И) може да ја нарушат нормалната низа на обработка
- За подобрување на ефикасноста на процесорот
- Програмски
  - пр. overflow, делење со нула
- Тајмер
  - Генериран од внатрешен процесорски тајмер
  - Се користи во pre-emptive multi-tasking
- В/И
  - од В/И контролер
- Хардверски пад
  - пр. Грешка за парност на меморијата

Тек на контрола на програмата



(a) No interrupts

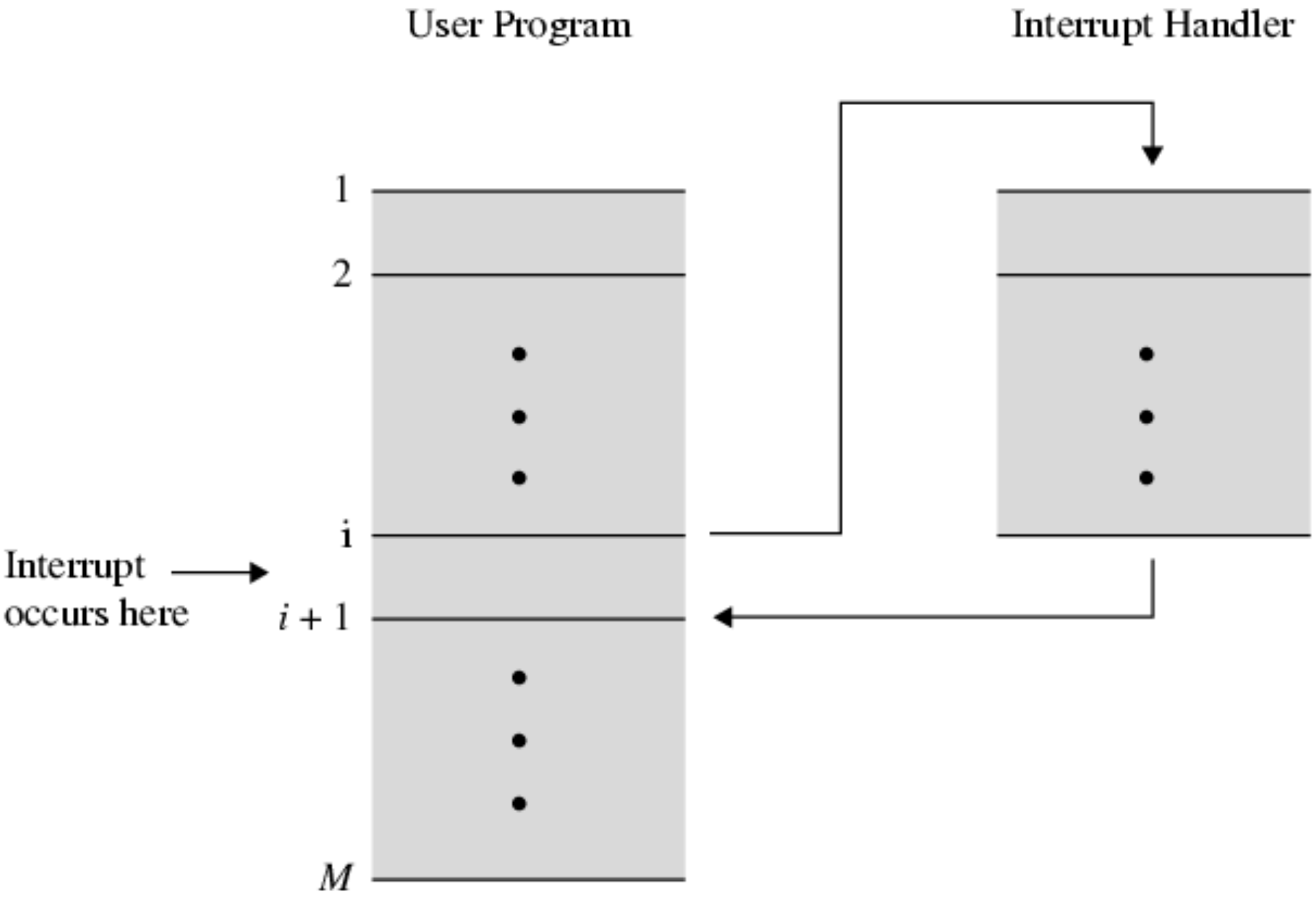
(b) Interrupts; short I/O wait

(c) Interrupts; long I/O wait

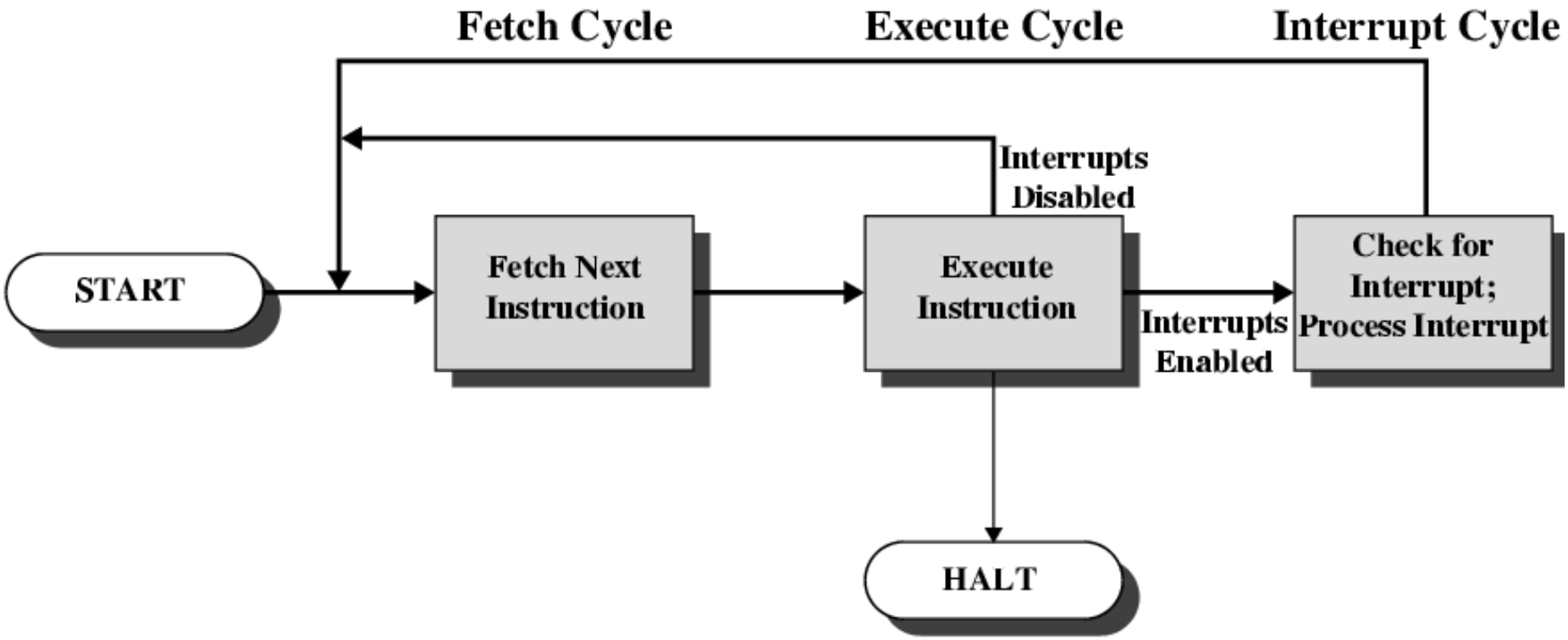
Циклус на прекини

- Додаден на инструкцискиот циклус
- Процесорот проверува дали има прекин
  - Се јавува преку сигнал за прекин
- Ако нема прекин, земи ја следната инструкција
- Ако има најавен прекин:
  - Се прекинува извршувањето на програмата
  - Се запишува содржината
  - Се поставува PC на почетната адреса на процедурата за обработка на прекин
  - Се обработува прекинот
  - Се враќа содржината и се продолжува со прекинатата програма

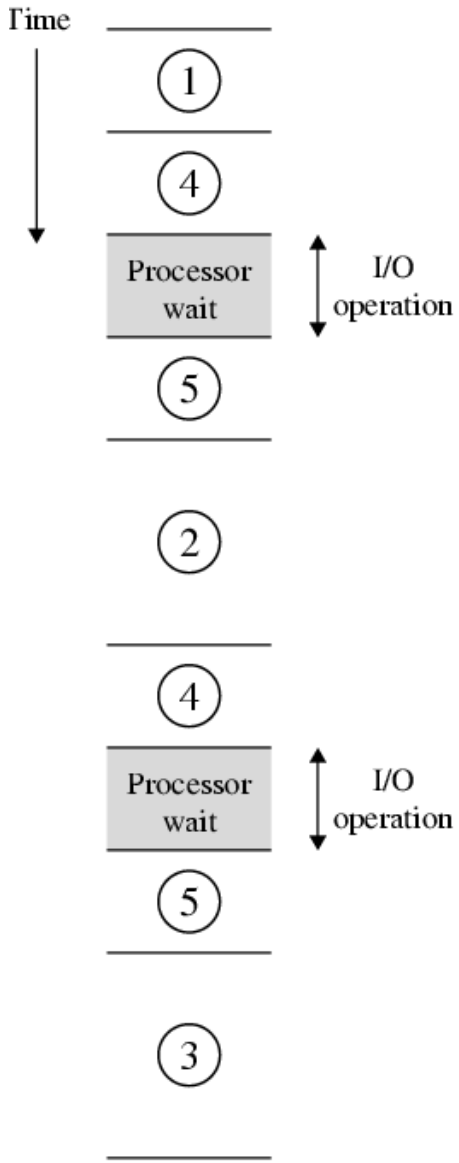
Пренос на контрола при прекин



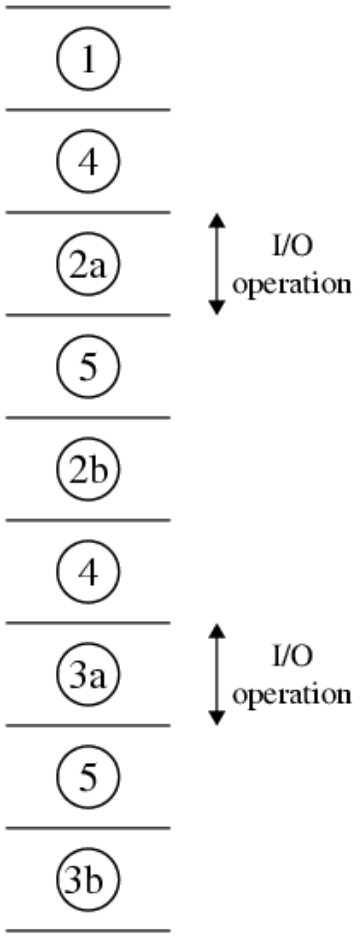
Инструкциски циклус со прекини



Program Timing Short I/O Wait

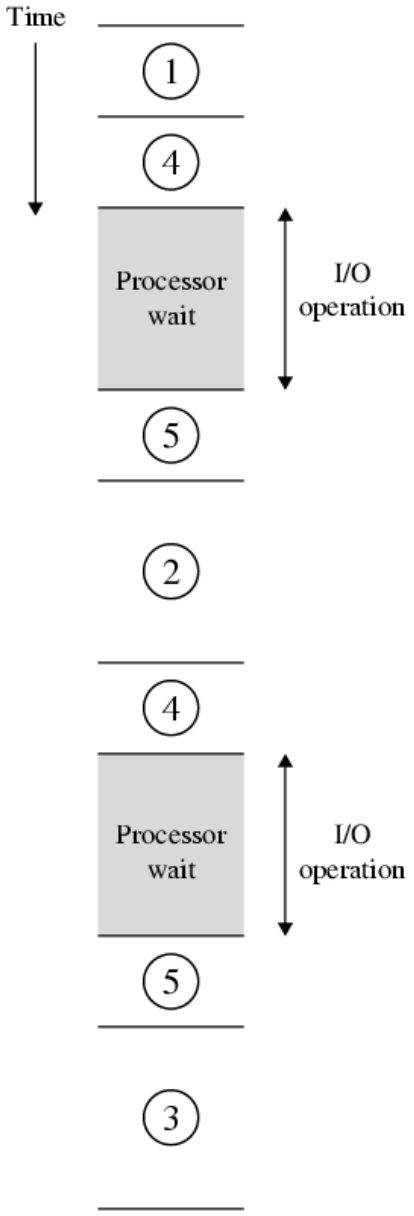


(a) Without interrupts

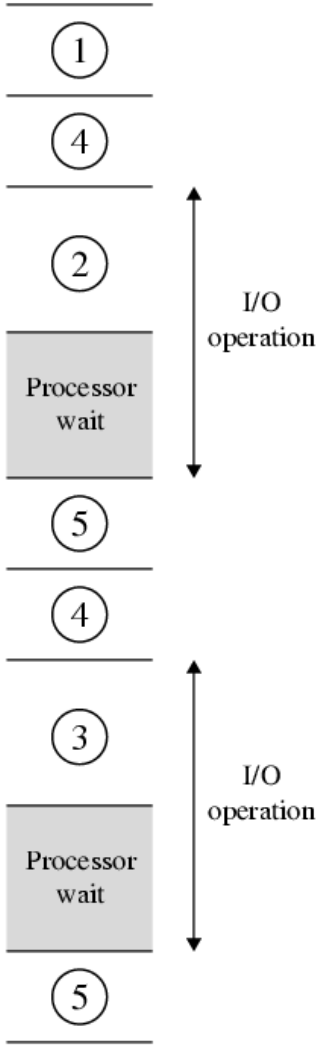


(b) With interrupts

Program Timing Long I/O Wait

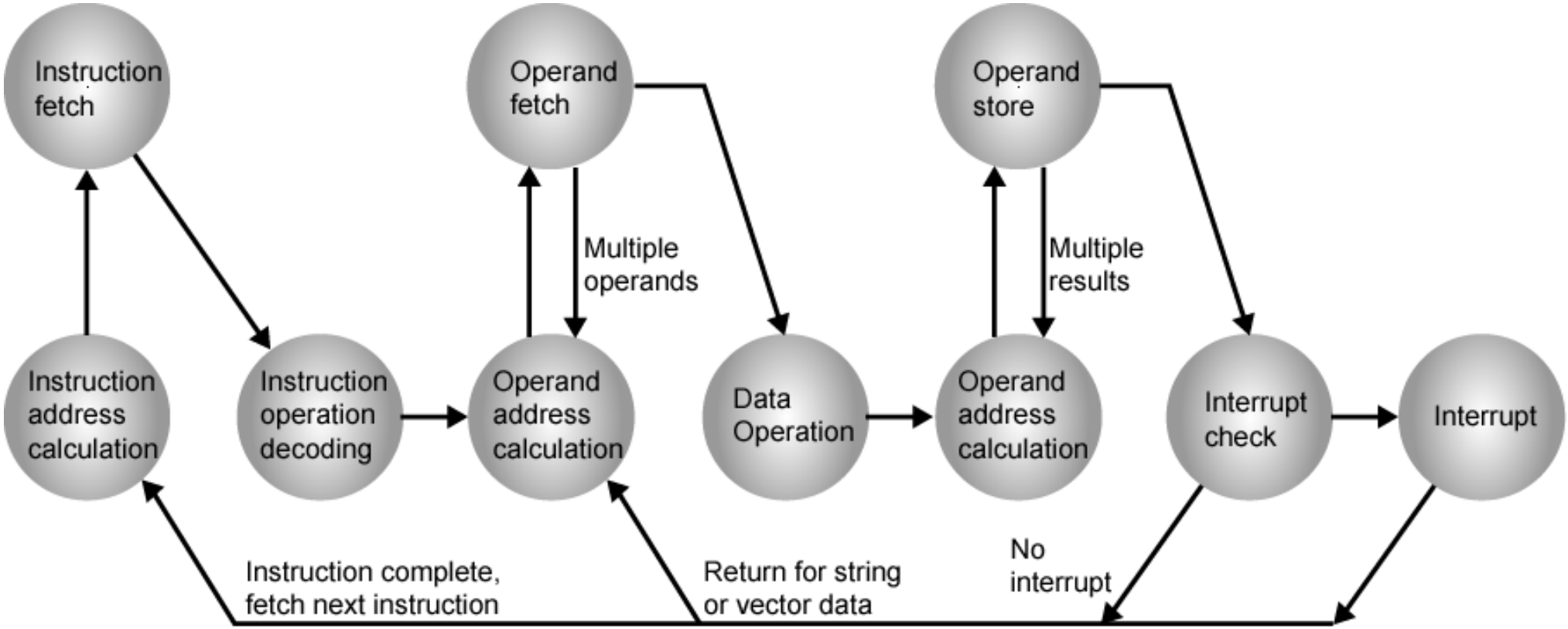


(a) Without interrupts



(b) With interrupts

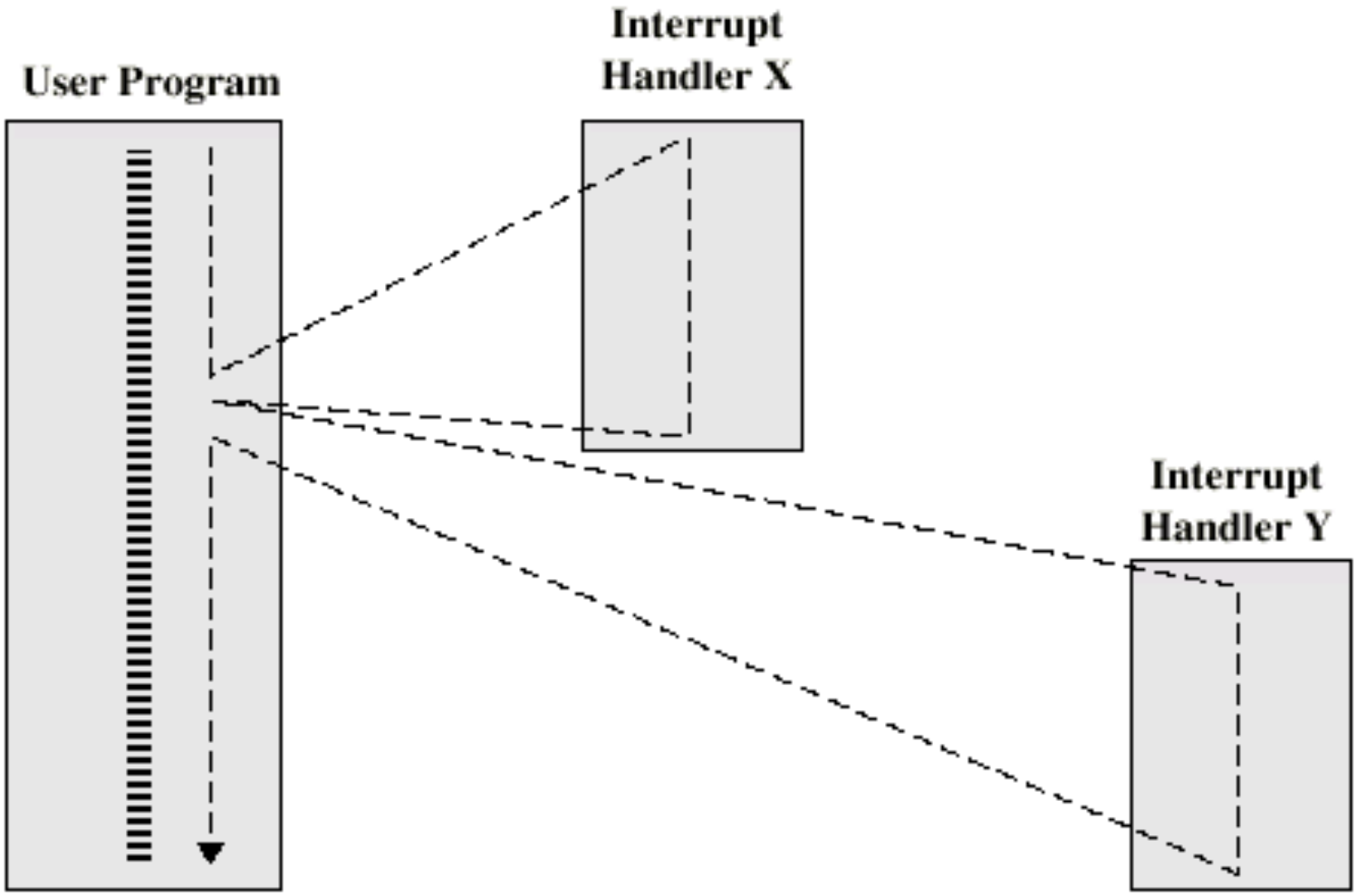
Инструкциски циклус (со прекини) - дијаграм на состојби



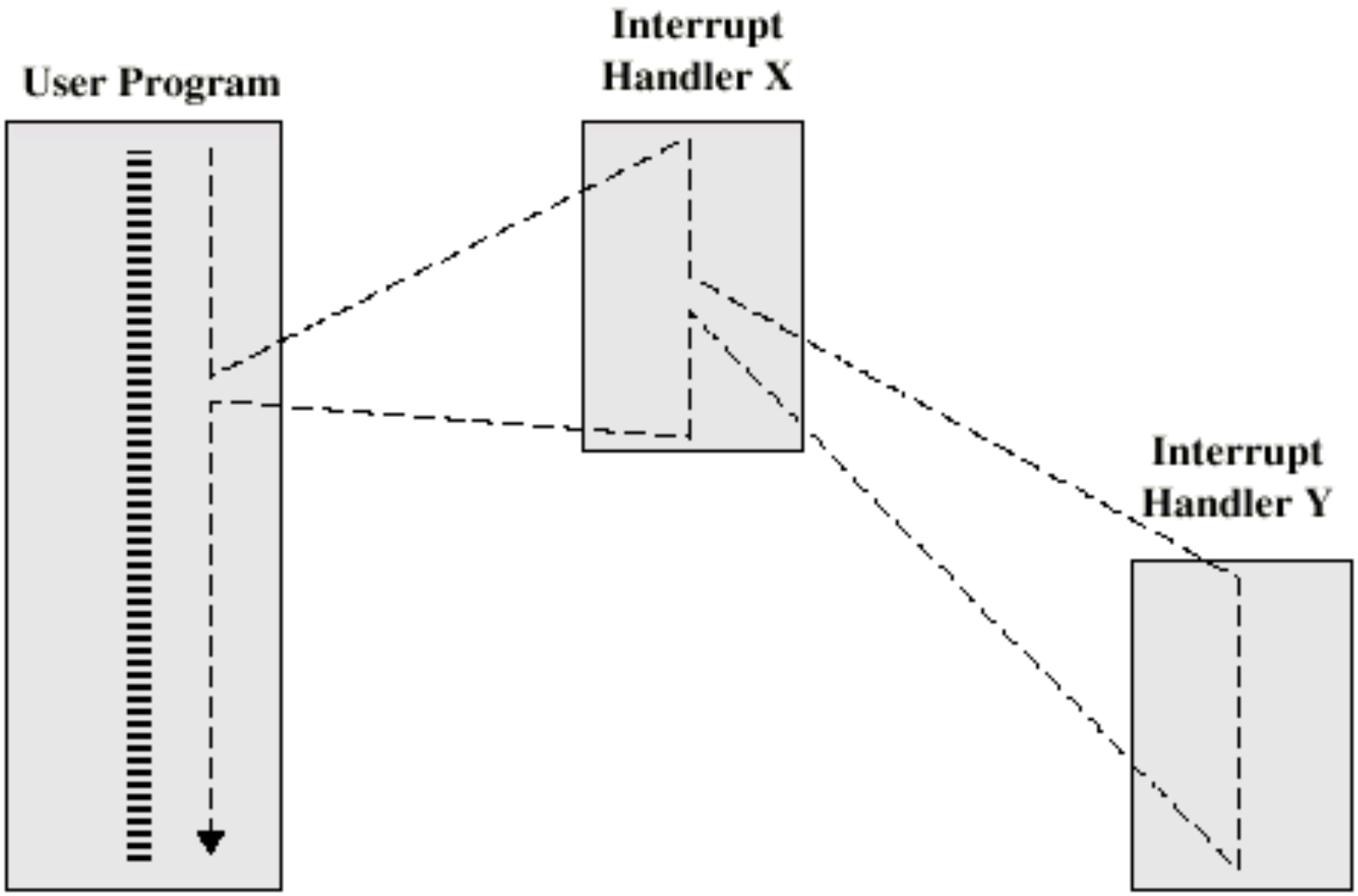
Повеќе прекини

- Оневозможување на прекини
  - Процесорот ќе игнорира други прекини додека обработува прекин
  - Прекините остануваат да чекаат додека да се заврши со обработка на првиот прекин
  - Прекините се обработуваат во низа како што се јавуваат
- Дефинирање на приоритети
  - Прекини со низок приоритет може да се прекинат од прекин со повисок приоритет
  - Кога ќе се обработи прекинот со повисок приоритет, процесорот се враќа на претходниот прекин

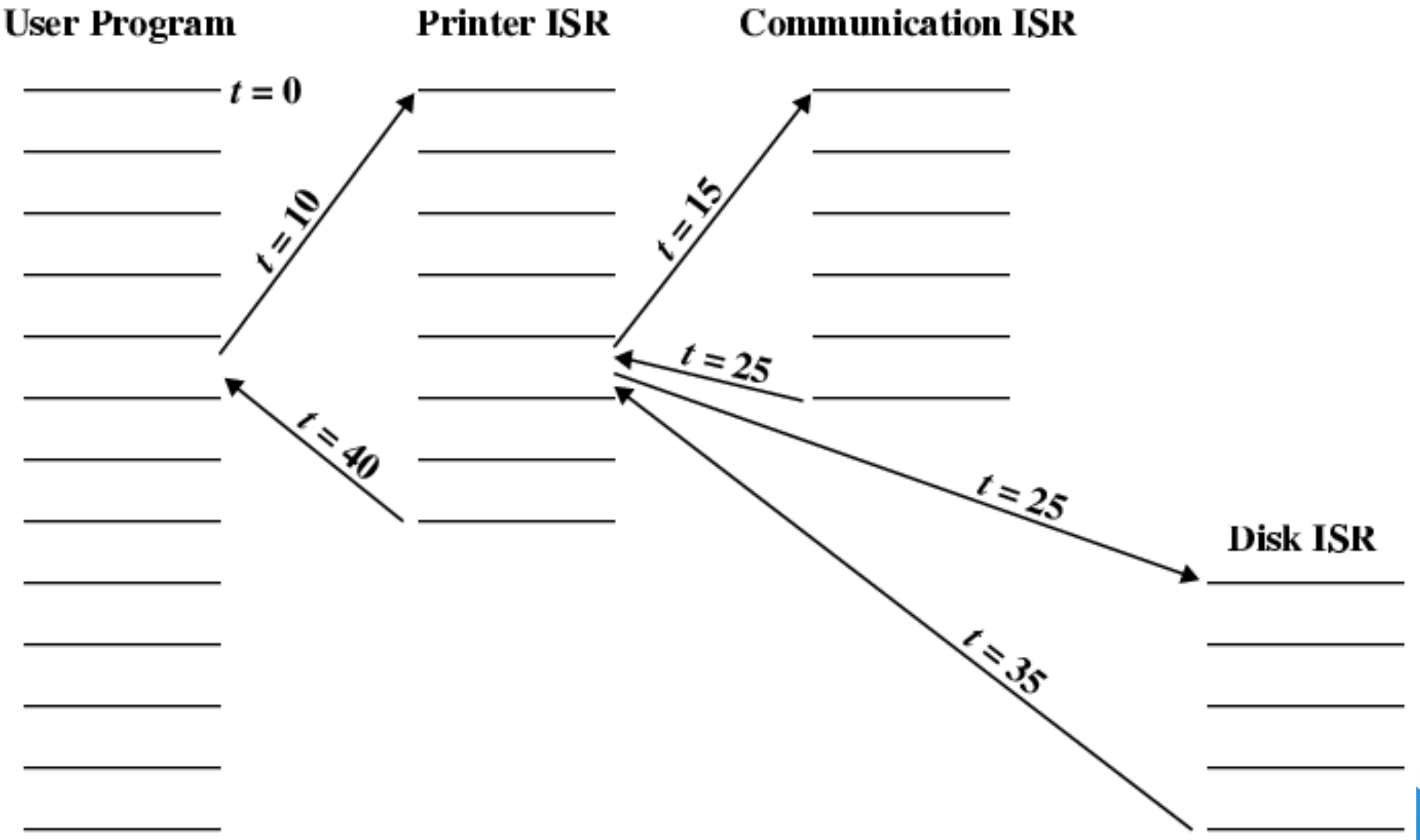
Повеќе прекини - секвенцијално



Повеќе прекини - вгнездено



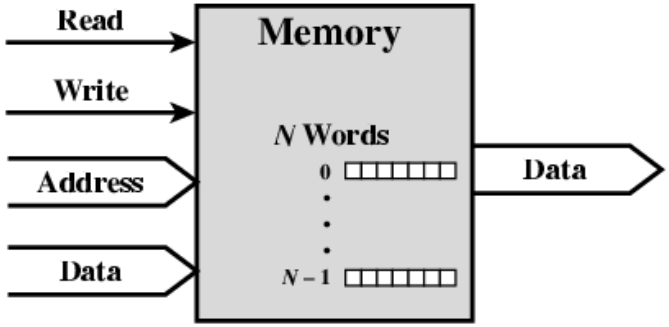
Временски распоред на повеќе прекини

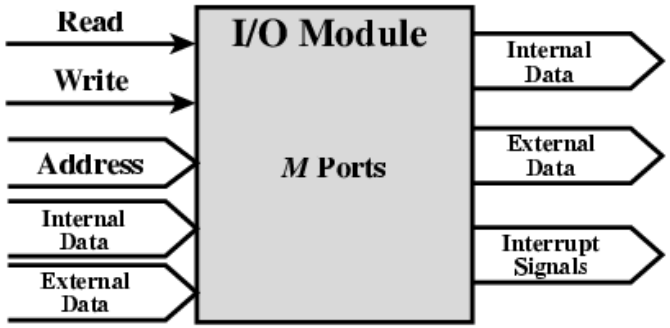


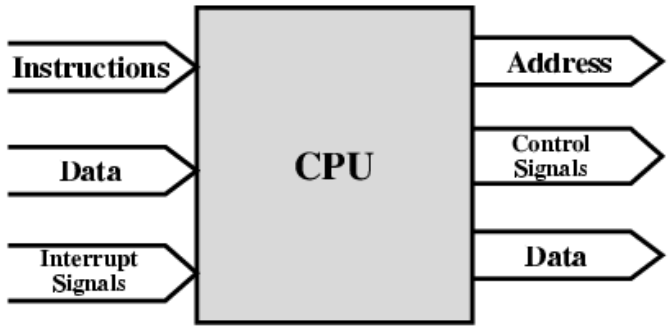
поврзување

- Сите единици мора да се поврзани
- Има различни типови на врски за различни типови единици
  - меморија
  - Влезно/излезни единици
  - CPU

Компјутерски модули







Поврзување со меморија

- Добива и испраќа податоци
- Добива адреси (на локации)
- Добива контролни сигнали
  - Read
  - Write
  - Timing

В/И поврзување (1)

- Слично на меморија од гледна точка на компјутерот
- излез
  - Прими податок од компјутер
  - Испрати податок на периферија
- влез
  - Прими податок од периферија
  - Испрати податок кон компјутер

В/И поврзување (2)

- Прими контролни сигнали од компјутер
- Испрати контролни сигнали кон периферија
- Прими адреса од компјутер
  - пр. Број на порта за идентификација на периферијата
- Испрати сигнал за прекин (контрола)

CPU поврзување

- Прима инструкции и податоци
- Запишува податоци (по обработка)
- Испраќа контролни сигнали кон други единици
- Прима (и реагира на) прекини

Магистрали

- Има поголем број на можни системи за поврзување
- Најчести се единечна и повеќекратна магистрала – BUS
- Системска магистрала
  - Магистрала која ги поврзува главните компјутерски компоненти
- пр. Контролна/адресна/податочна (PC)
- пр. Unibus (DEC-PDP)

Што е магистрала?

- Комуникациска патека која поврзува 2 или повеќе уреди
- Споделен медиум за пренос
- Вообичаено е broadcast
- Често е групирана
  - Број на канали/линии во една магистрала
  - пр. 32 битна податочна магистрала има 32 одвоени канали со по 1 бит
- Линиите за напојувања не мора да се прикажани

Системска магистрала

- 50-100тици одвоени линии
- Секоја со свое значење и улога

- Типично групирање во:
  - Податочна магистрала +
  - Адресна магистрала +
  - Контролна магистрала

Податочна магистрала

- Пренесува податоци
  - Нема разлика помеѓу “податок” и “инструкција” на ова ниво
- Ширината е клучна за одредување на перформансите
  - 8, 16, 32, 64 bit

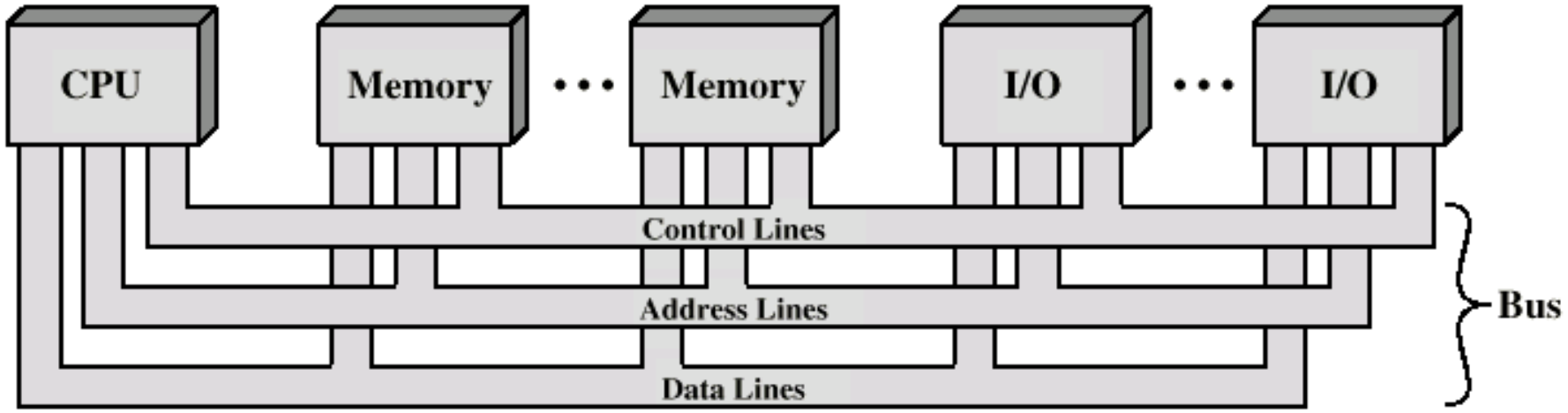
Адресна магистрала

- Го одредува изворот или одредиштето на податокот
- пр. CPU треба да прочита инструкција (податок) од дадена локација во меморијата
- Ширината на магистралата го одредува максималниот мемориски капацитет на системот
  - пр. 8080 има 16 bit адресна магистрала што значи 64k адресен простор
- Адресирање на В/И порти

Контролна магистрала

- Информација за контрола и време
  - Меморија читај/запиши сигнал
  - В/И читај/запиши
  - Барање за пристап до магистрала
  - Барање за прекин
  - Такт сигнали – Clock
  - Ресет

Шема на поврзување со магистрали

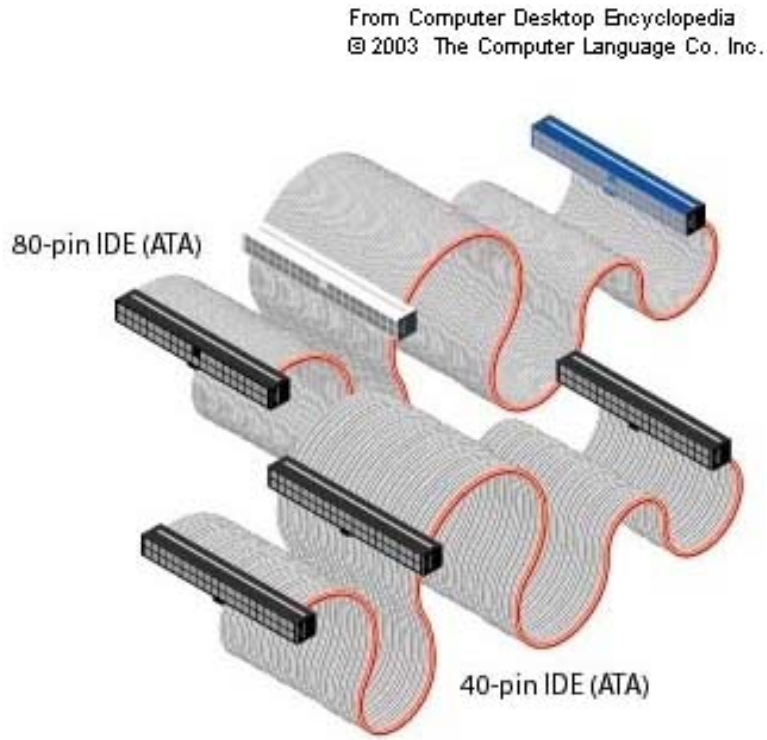


Работа на магистрала

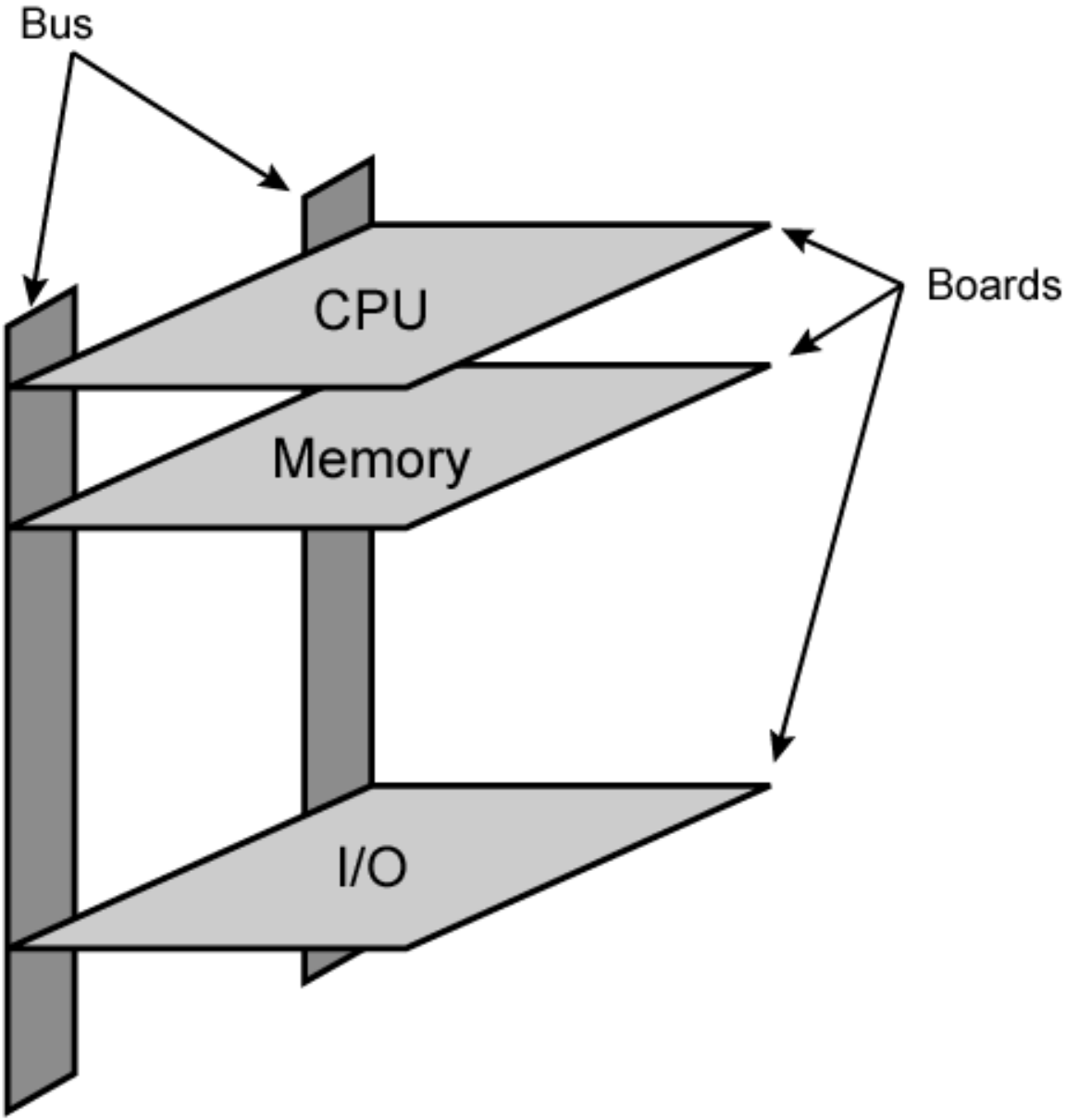
- Ако модул сака да испрати податоци до друг модул мора да направи 2 работи
  - Да добие право на користење на магистралата
  - Да ги пренесе податоците преку магистралата
- Ако модул сака да добие податоци од друг модул мора да направи 3 работи
  - Да добие право на користење на магистралата
  - Да го пренесе барањето за податоци преку магистралата
  - Потоа чека вториот модул да ги прати податоците

Како изгледаат магистралите?

- Паралелни линии на електронските плочки
- Плоснати лентести кабли - Ribbon cables
- Strip конектори на матичната плоча
  - пр. PCI
- Множество од жици



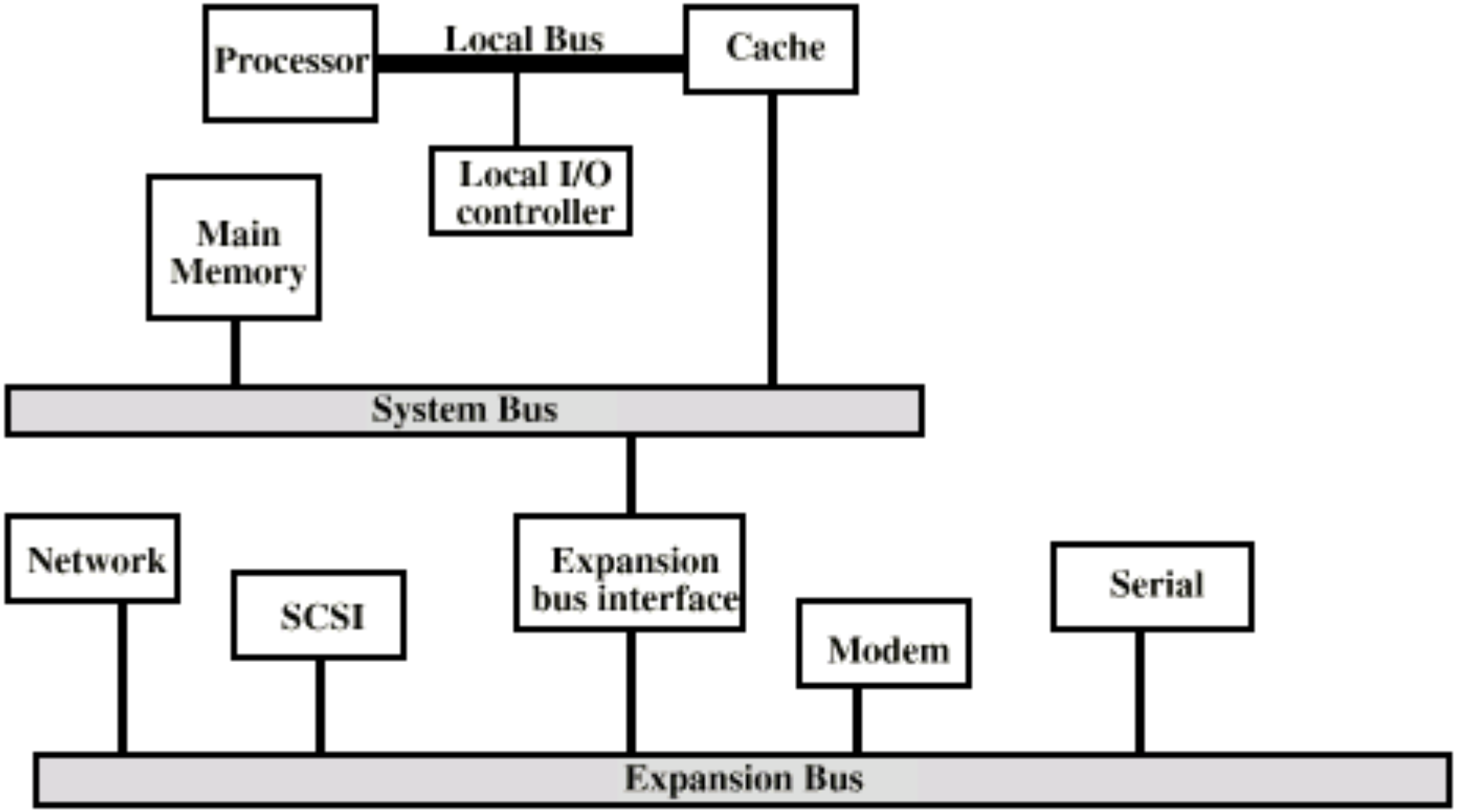
Физичка реализација на магистрала



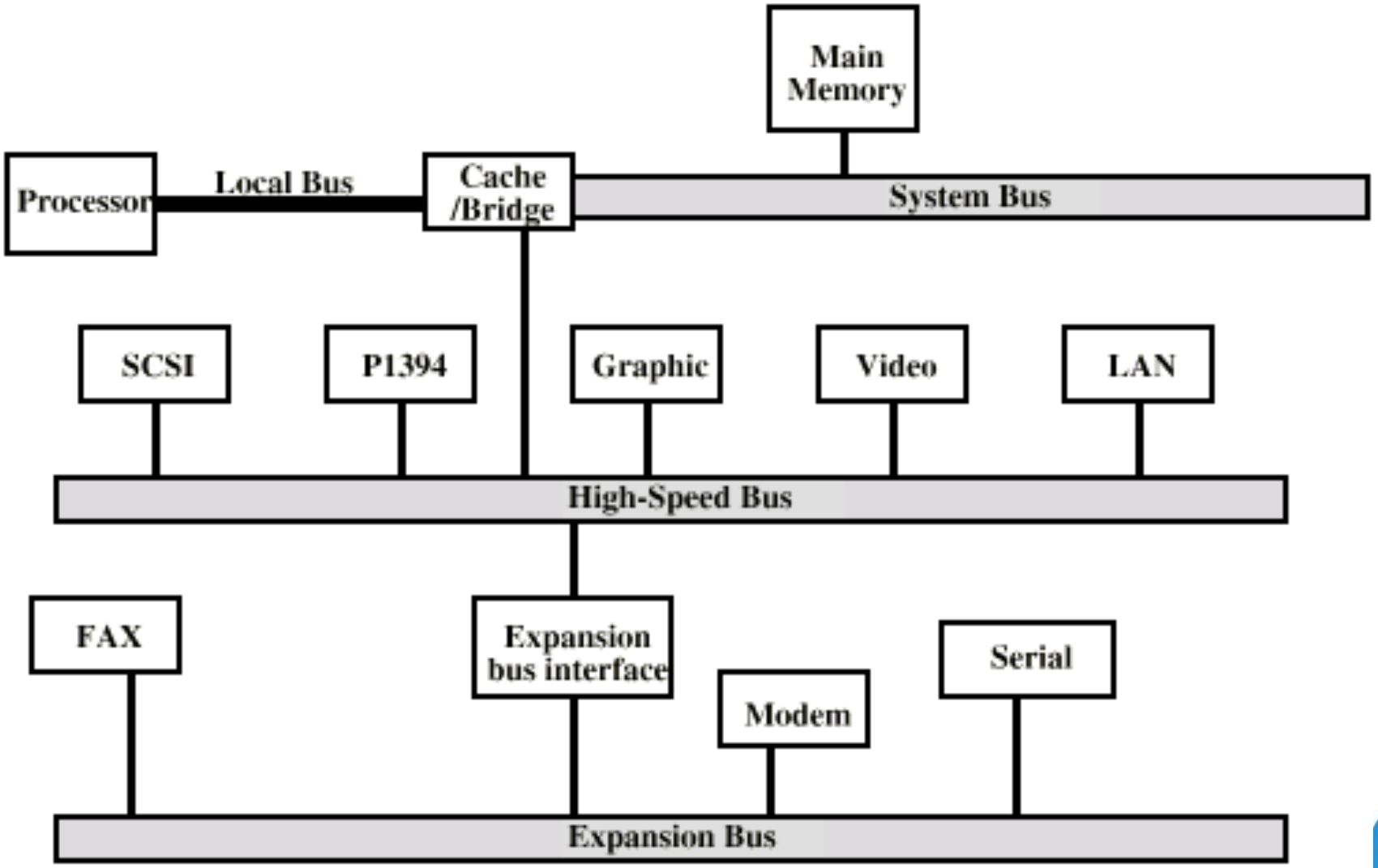
Проблеми со единечна магистрала

- Многу уреди на магистралата води кон:
  - Пропагациско доцнење
    - Долгите податочни патеки значат координација на магистралата што лошо влијае врз перформансите
    - Насобраниот податочен трансфер се приближува до капацитетот на магистралата
- Повеќето системи користат повеќе магистрали за да ги надминат овие проблеми

Традиционално (ISA) (со кеш)



Магистрала со високи перформанси мезанин архитектура



Типови на магистрали

- Специјално наменета - Dedicated
  - Одвоени податочни и адресни линии
- мултиплексирана
  - Споделени линии
  - Контролна линија која кажува дали е адреса или податок
  - Предност – помалку линии
  - Недостатоци
    - Покомплексна контрола
    - Перформанси

Арбитрација на магистралата

- Повеќе од еден модул ја контролира магистралата
  - Пр. CPU и DMA контролер
- Само еден модул може да ја контролира магистралата во единица време
- Арбитрацијата може да биде централизирана и дистрибуирана

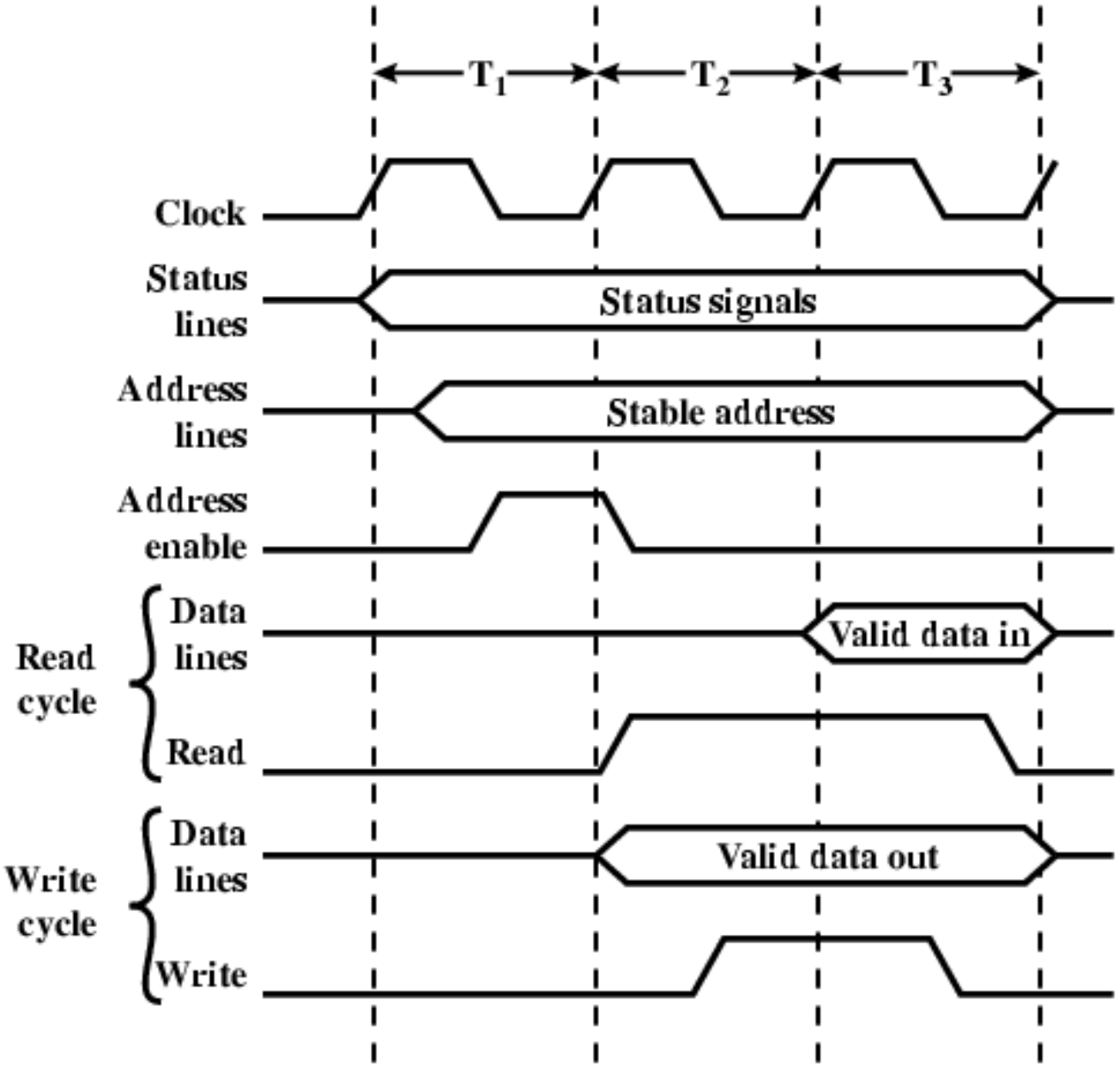
Централизирана или дистрибуирана арбитрација

- централизирана
  - Еден хардверски уред го контролира пристапот до магистралата
    - Контролер на магистралата
    - Арбитер
  - Може да е дел од CPU или одвоен
- дистрибуирана
  - Секој модул може да ја присвои магистралата
  - Контролна логика на сите модули

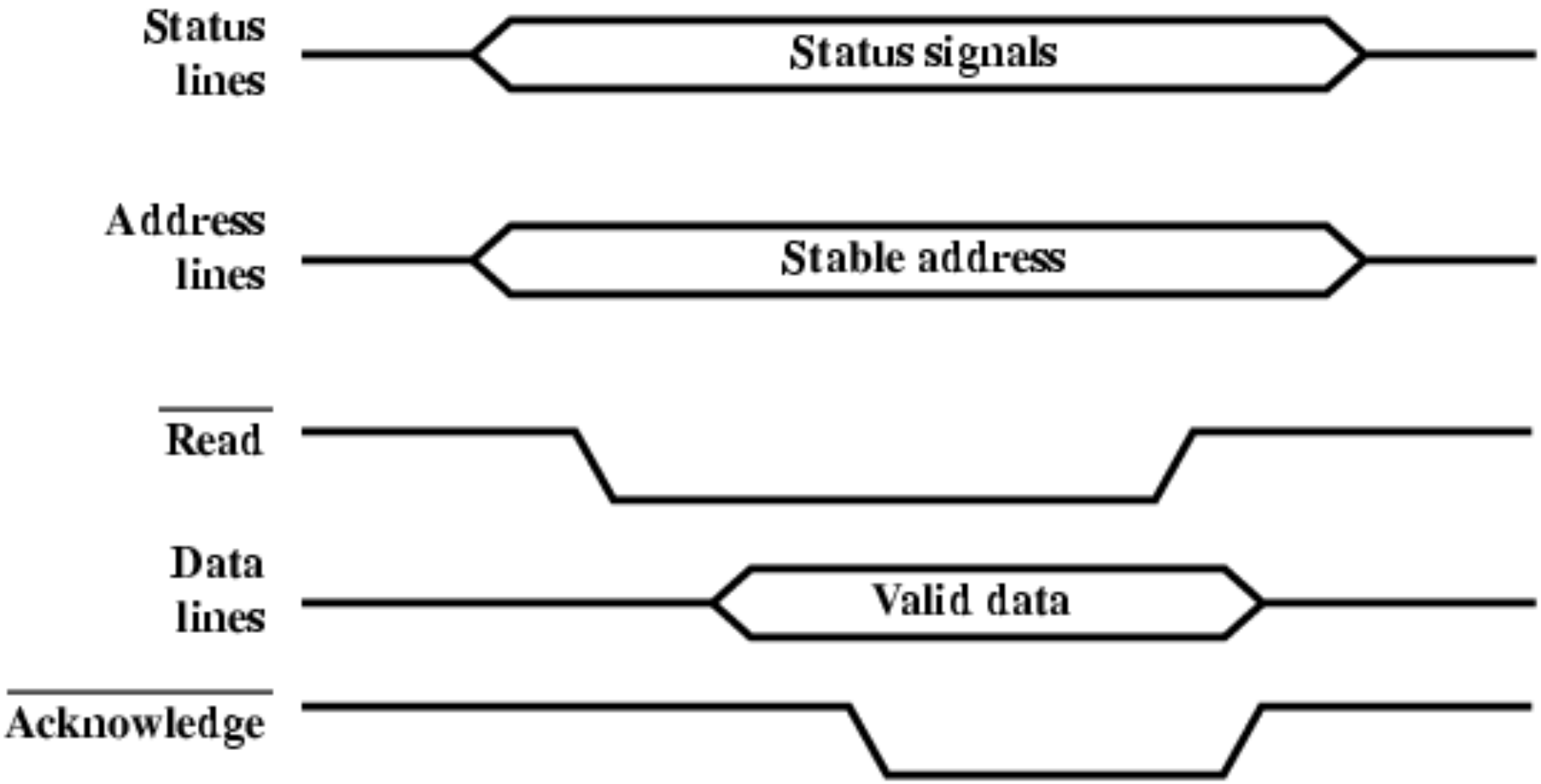
Timing

- Координација на настаните на магистралата
- синхроно
  - Настаните се одредени според такт сигналите
  - Контролната магистрала ја вклучува и clock линијата
  - едно 1-0 е циклус на магистралата
  - Сите уреди ја читаат clock линијата
  - Вообичаено синхронизирани на растечка ивица
  - Обично еден циклус по настан

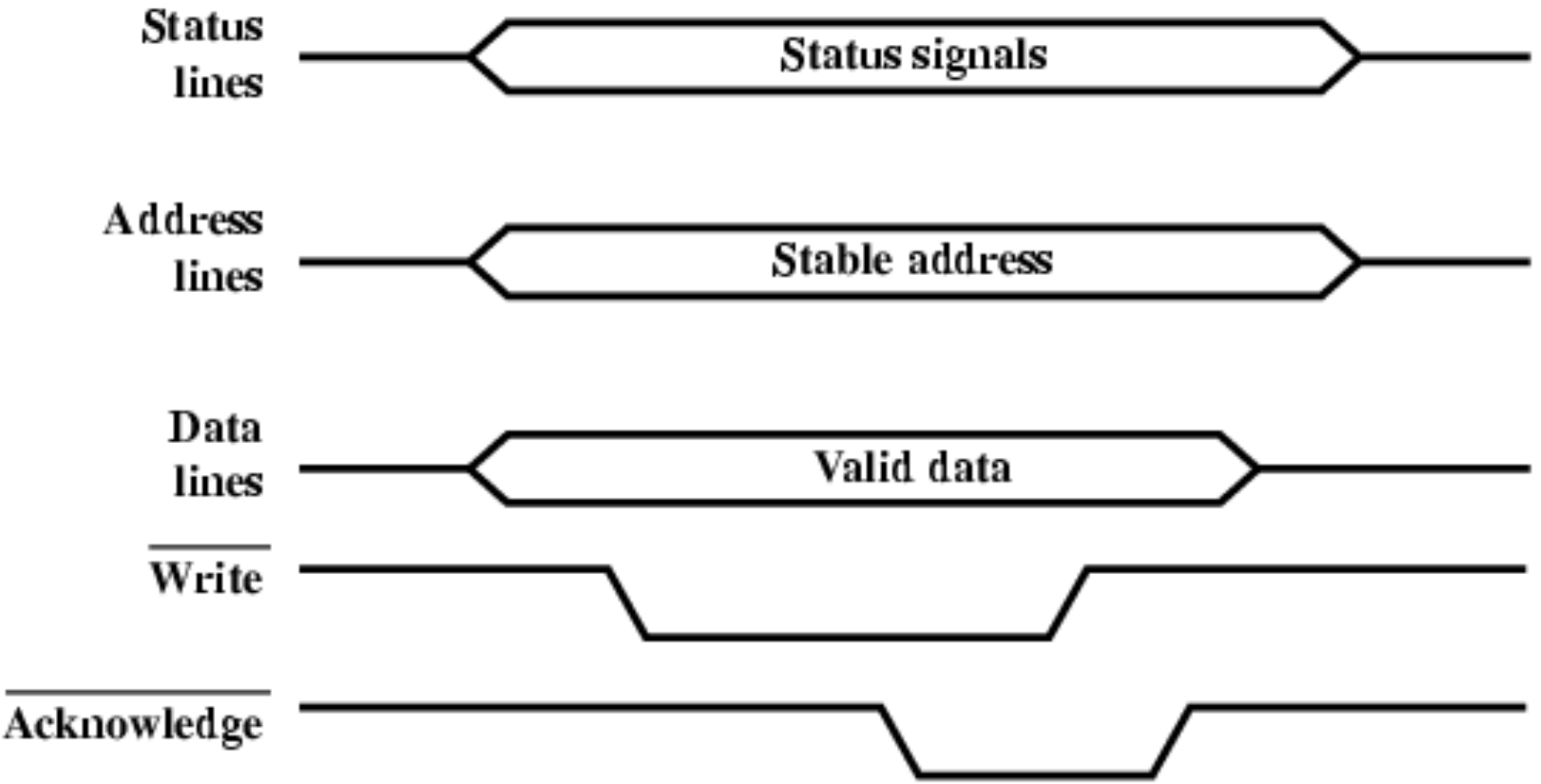
Синхрон временски дијаграм



Асинхрон временски дијаграм за читање

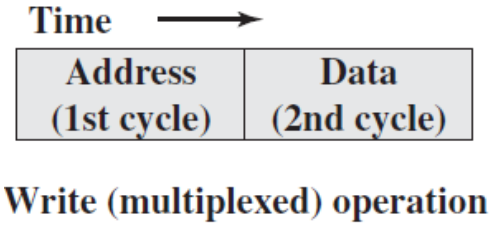


Асинхрон временски дијаграм за запишување

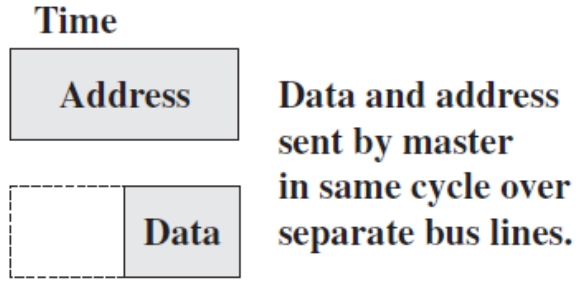


Тип на трансфер на податоци

- Read
- Write
- Read-modify-write
- Read-after-write
- Block

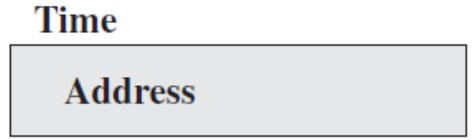


Write (multiplexed) operation



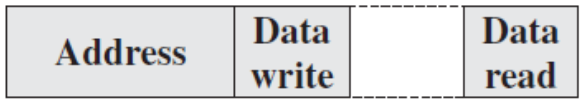
Write (non-multiplexed) operation

Read (multiplexed) operation

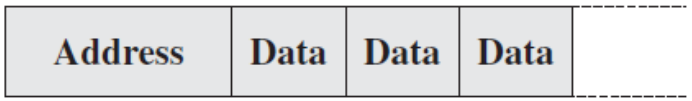


Read (non-multiplexed) operation

Read-modify-write operation



Read-after-write operation



Block data transfer

Во CPU

Инструкциско множество – карактеристики и функции

Chapter 10

Instruction Sets:

Characteristics and Functions

Што е инструкциско множество ?

- Комплетната колекција од инструкции кои ги разбира еден CPU
- Машински код
- Бинарен
- Обично претставен преку асемблер

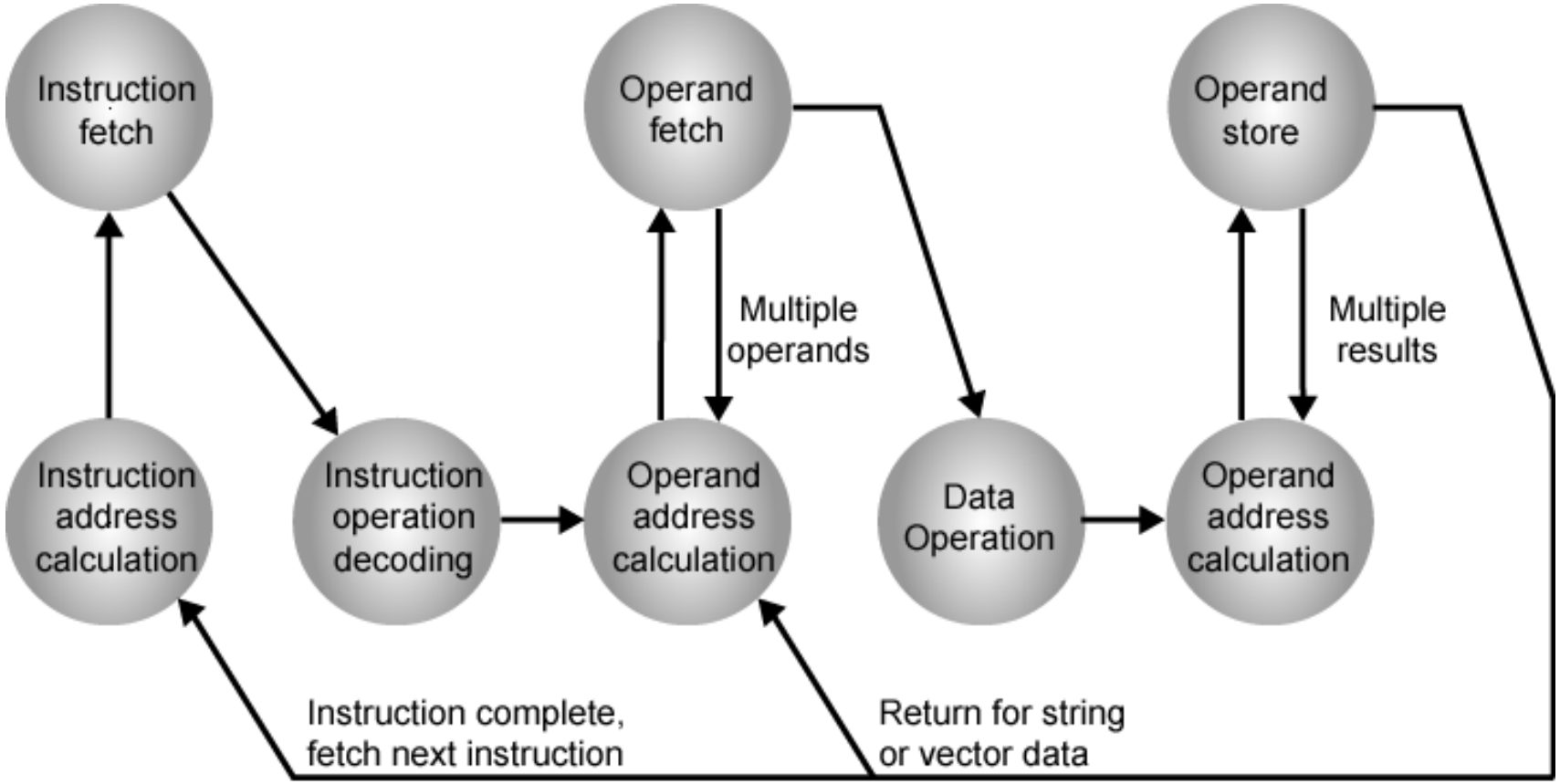
Елементи на инструкција

- Операциски код (Op code)
  - Направи го ова
- Референца кон изворен операнд
  - На ова
- Референца кон резултантен операнд
  - Стави го одговорот тука
- Референца кон следна инструкција
  - Откако го направи тоа, направи го ова...

Каде се операндите?

- Главна меморија (или виртуелна или кеш)
- CPU регистри
- Непосредни
- В/И уреди

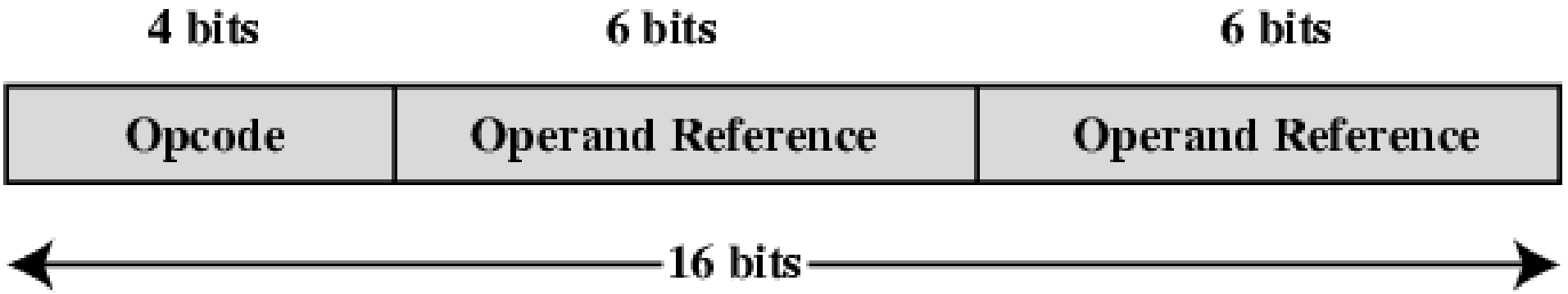
потсетување



Претставување на инструкциите

- Во машинскиот код секоја инструкција има уникатна низа од битови
- За човечко разбирање (барем за програмерите ☺) се користи симболичко претставување
  - пр. ADD, SUB, LOAD
- И операндите може да се претстават на овој начин
  - ADD A,B

Едноставен формат на инструкција



Типови на инструкции

- Обработка на податоци
  - Аритметички и логички операции
- Складирање на податоци
  - Пренос на податоци од/во регистри и главна меморија
- Пренос на податоци
  - В/И инструкции
- Контрола на текот на програмата
  - Тест и гранање

Број на адреси (а)

- 3 адреси
  - Операнд 1, операнд 2, резултат
  - a = b + c;
  - Може да има четврт – следна инструкција (обично е имплицитен)
  - Не е вообичаено
  - Потребни се многу долги зборови за да се зачува се што е потребно

Број на адреси (b)

- 2 адреси
  - Една адреса претставува операнд и резултат
  - a = a + b
  - Ја намалува должината на инструкцијата
  - Потребна е дополнителна работа
    - Привремено складиште за дел од резултатите

Број на адреси (c)

- 1 адреса
  - Имплицитна втора адреса
  - Обично е регистер (акумулатор)
  - Вообичаено за раните машини

Instruction		Comment
SUB	Y, A, B	Y ← A − B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

$$Y = \frac{A - B}{C + (D \times E)}$$

Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y − B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC − B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) One-address instructions

Број на адреси (d)

- 0 (нула) адреси
  - Сите адреси се имплицитни
  - Користи магацин (stack)
  - пр. push a
  - push b
  - add
  - pop c

  - c = a + b

Колку адреси

- Повеќе адреси
  - покомплексни (помоќни?) инструкции
  - Повеќе регистри
    - Операциите меѓу регистри се побрзи
  - Помалку инструкции во програма
- Помалку адреси
  - поедноставни (моќни?) инструкции
  - Повеќе инструкции по програма
  - Побрзо земи/изврши на инструкциите

Одлуки при дизајн(1)

- Репертоар на операции
  - Колку операции?
  - Што може да направат?
  - Колку се комплексни?
- Податочни типови
- Инструкциски формати
  - Должина на полето за opcode
  - Број на адреси

Одлуки при дизајн (2)

- регистри
  - Број на достапни регистри во CPU
  - Кои операции може да се изведат со кои регистри?
- Режими на адресирање (подоцна…)

- RISC v CISC

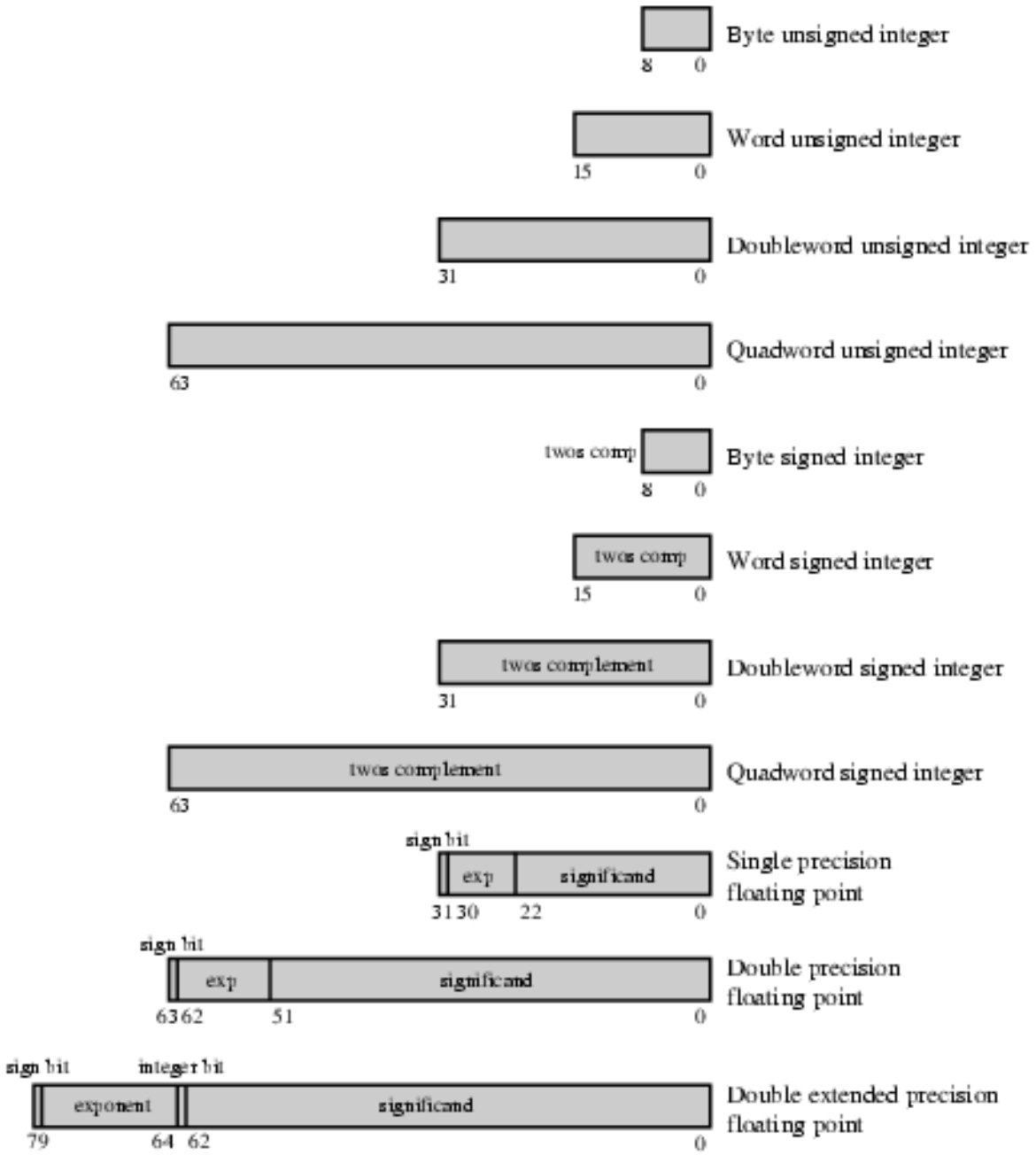
Типови на операнди

- адреси
- броеви
  - цели/со подвижна запирка
- знаци
  - ASCII и друго
- логички
  - Битови или знаменца
- (фуснота: дали има разлика помеѓу броеви и знаци?
  С програмирање!)

x86 податочни типови

- 8 bit Byte
- 16 bit word
- 32 bit double word
- 64 bit quad word
- 128 bit double quadword
- Адресирањето е преку 8 bit единица
- Зборовите не мора да се подредени на парни адреси
- Податоците до кои се пристапува преку 32 bit магистрала се во единици од double word на адреси делливи со 4
- Little endian

x86 нумерички податочни формати

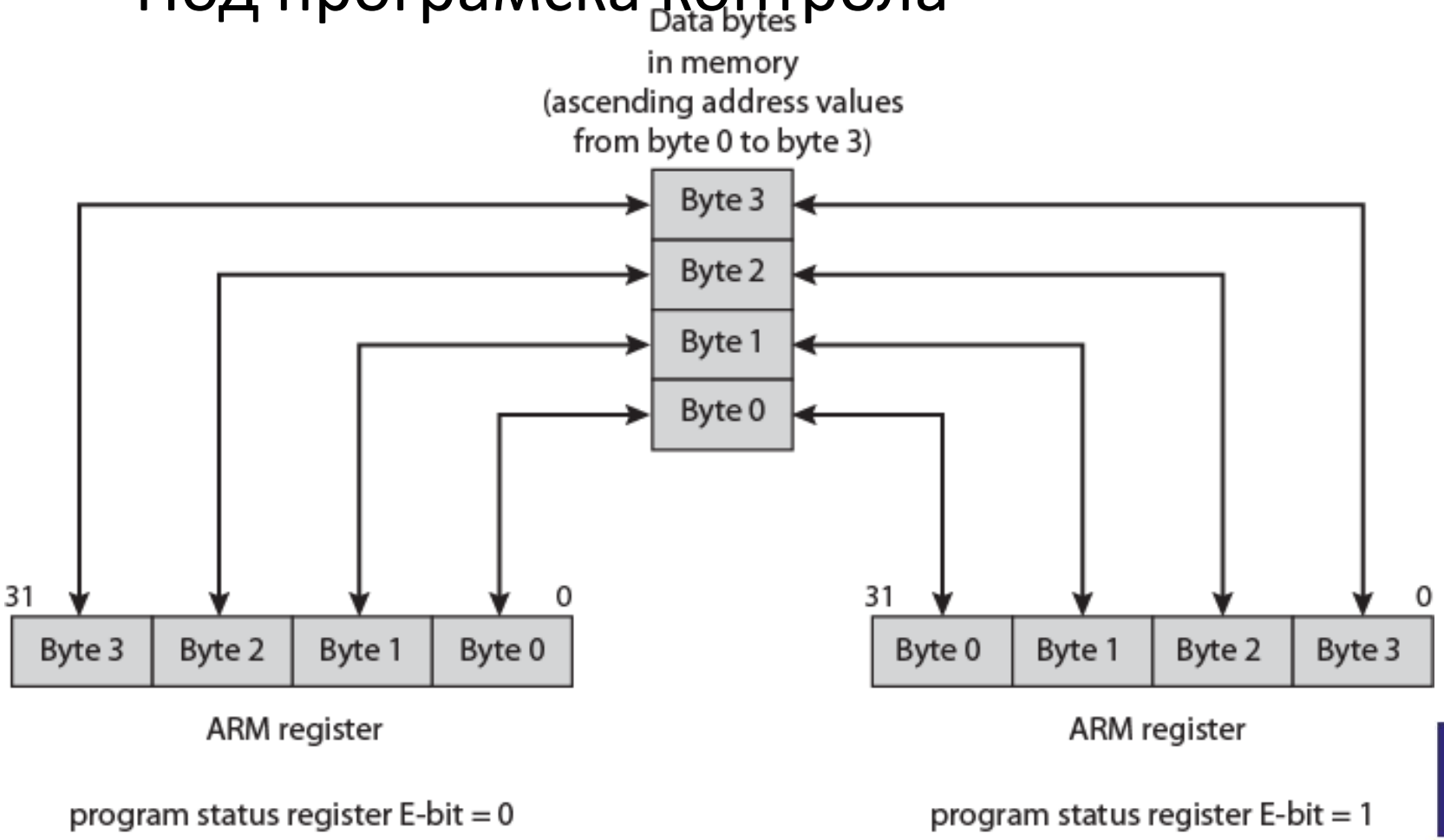


ARM податочни типови

- 8 (byte), 16 (halfword), 32 (word) bits
- Halfword и word пристапите треба да се подредени
- Постои можност и за неподреден пристап
- За сите типови е поддржана интерпретација на
  - цел број без знак
  - цел број со знак во двоен комплемент
- Повеќето имплементации немаат хардвер за броеви со подвижна запирка
  - Штеди батерија и простор
  - Софтверски се имплементирани
  - Има опционален ко-процесор за оваа намена кој работи со IEEE 754 со единечна и двојна прецизност

ARM Endian поддршка

- E-bit во системскиот контролен регистер
- Под програмска контрола



Типови на операции

- Пренос на податоци
- Аритметички
- Логички
- Претворба
- В/И
- Системска контрола
- Контрола на пренос

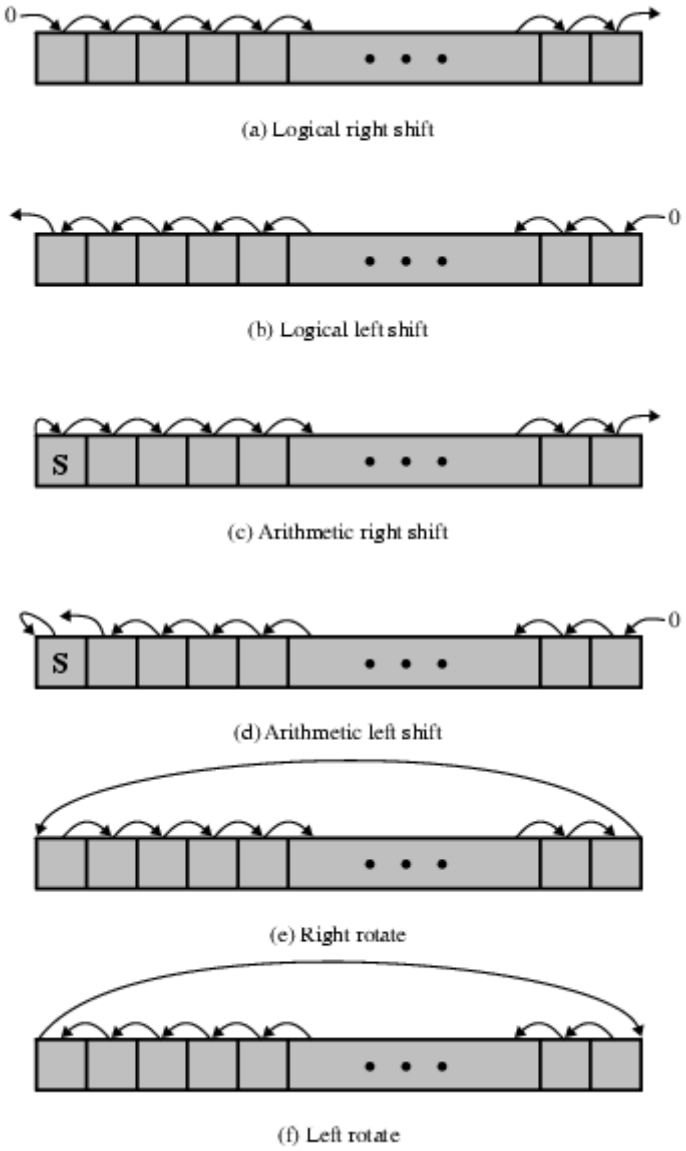
Пренос на податоци

- дефинирај
  - Извор
  - Одредиште
  - Количество податоци
- Може да има различни инструкции за различни преноси
  - пр. IBM 370
- Или една инструкција и различни адреси
  - пр. VAX

Аритметички

- Додади, одземи, помножи, подели
- Цел број со знак
- Број со подвижна запирка ?
- Може да вклучува
  - инкрементација (a++)
  - декрементација (a--)
  - негација (-a)

Операции за поместување и ротација



Логички

- Операции врз битовите
- AND, OR, NOT

Претворба

- пр. бинарно во декадно

Влезно/излезни

- Може да бидат специјални инструкции
- Може да се користат инструкциите за пренос на податоци
  - Т.н. Мемориско мапирање
- Може да се прави со одвоен контролер (DMA)

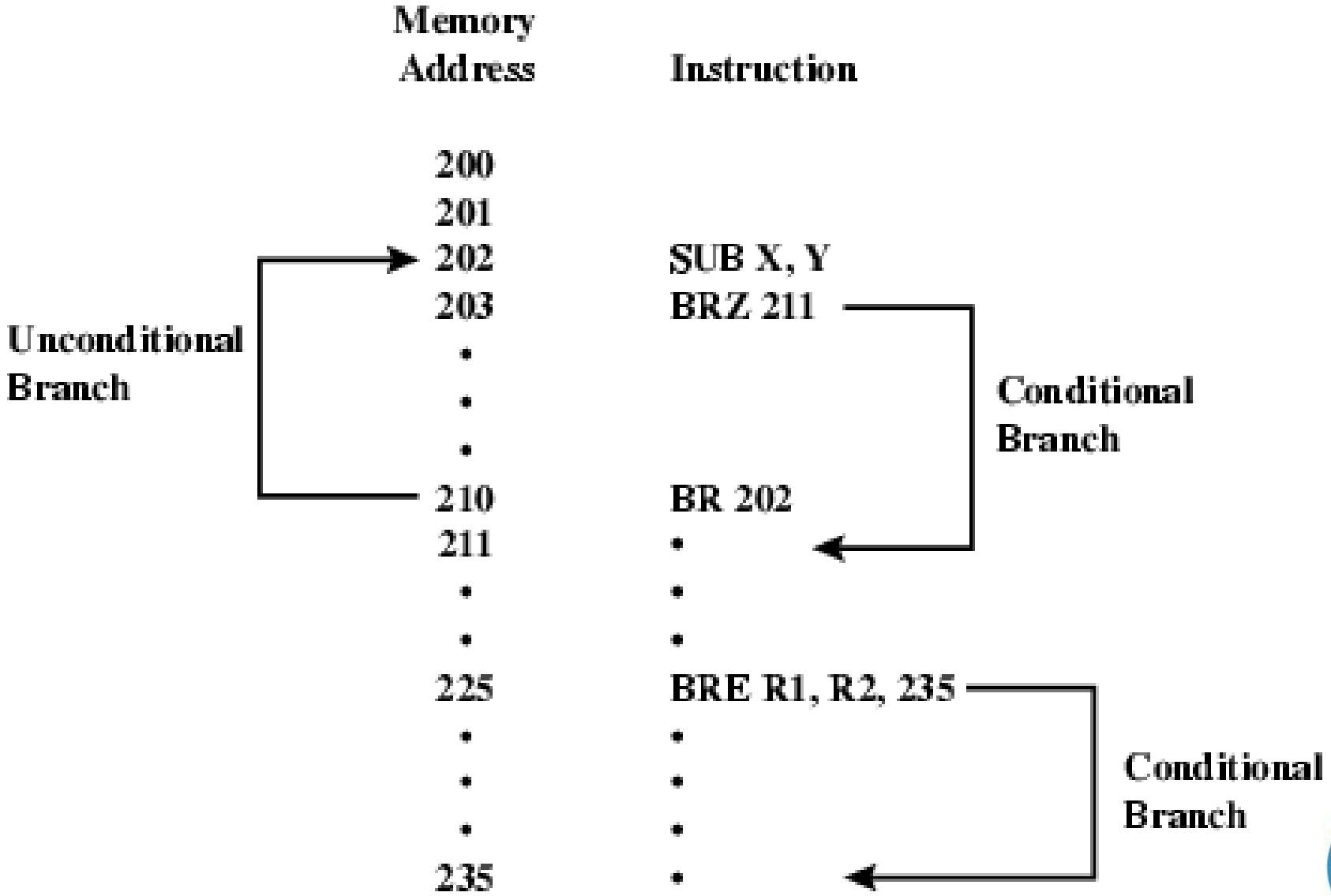
Системска контрола

- Привилегирани инструкции
- CPU треба да е во специјална состојба
  - Ring 0 на 80386+
  - Kernel режим
- Резервирани за употреба од страна на оперативниот систем

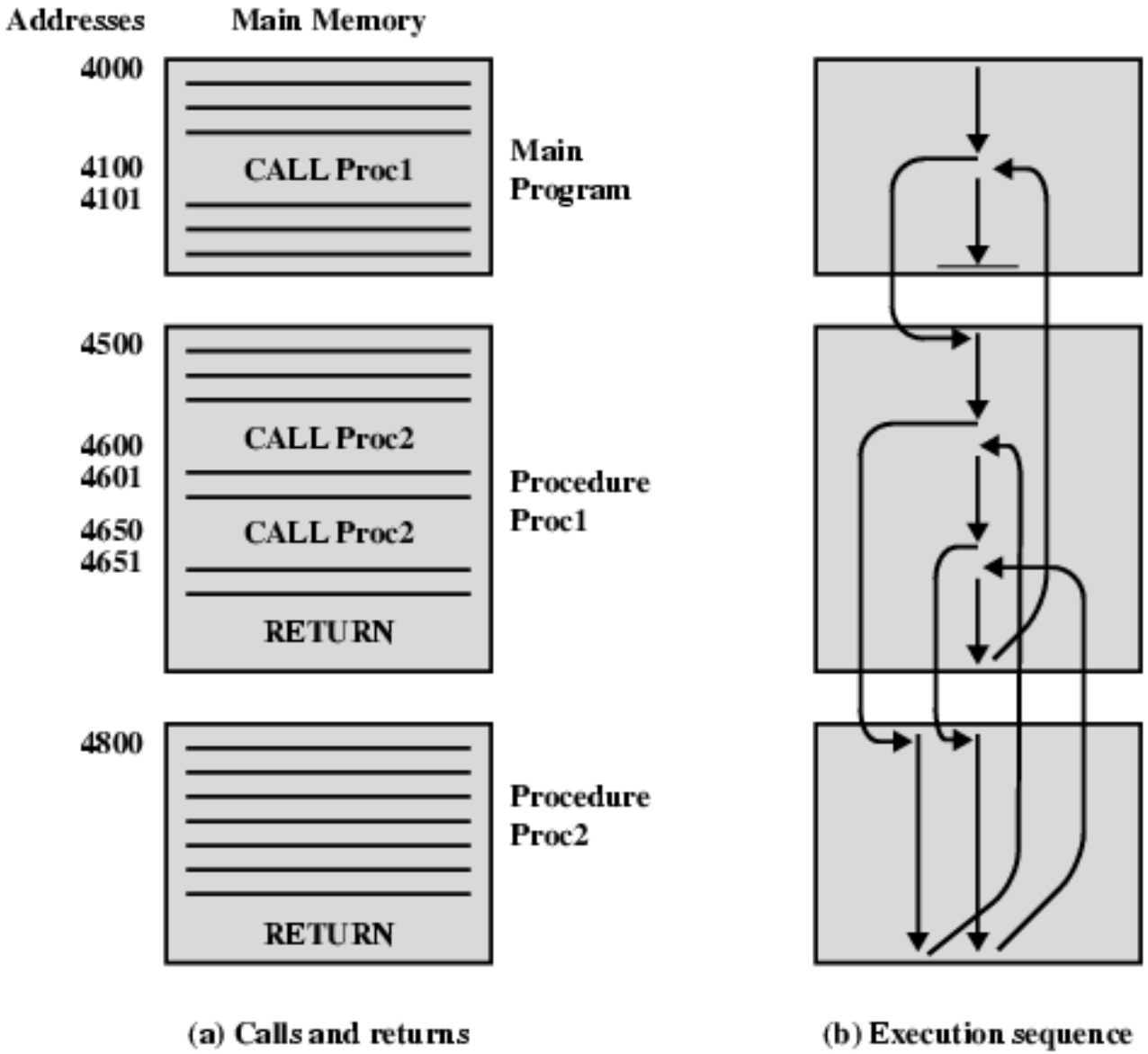
Пренос на контрола

- гранање
  - пр. скокни на x ако резултатот е нула
  - Условно и без условно
- скокни
  - пр. инкрементирај и скокни ако е нула
  - ISZ Register1
  - BR(anch) xxxx
  - Обично за правење јамки
- Повик на процедура
  - Повик на обработка на прекин

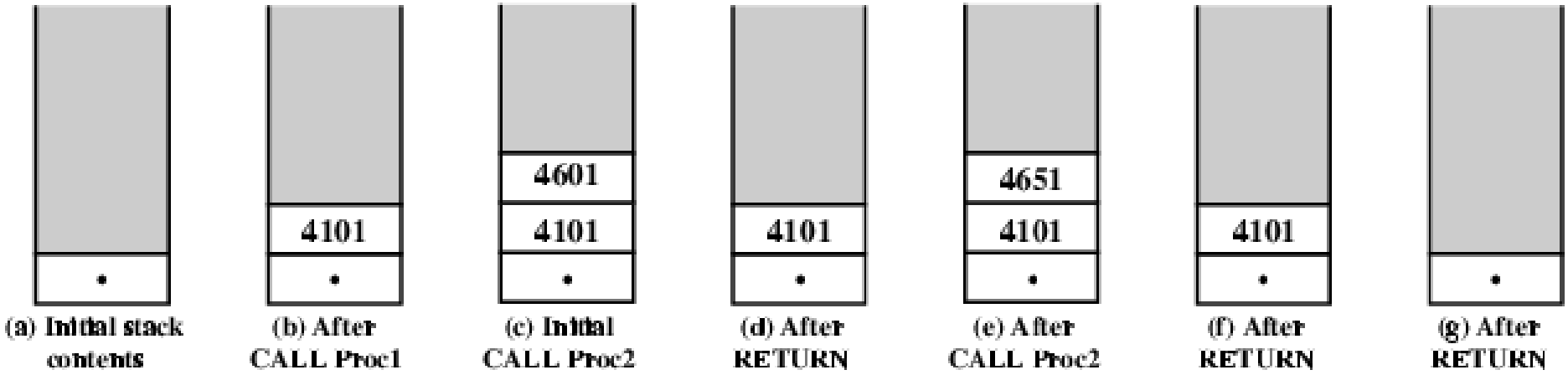
Инструкции за гранање



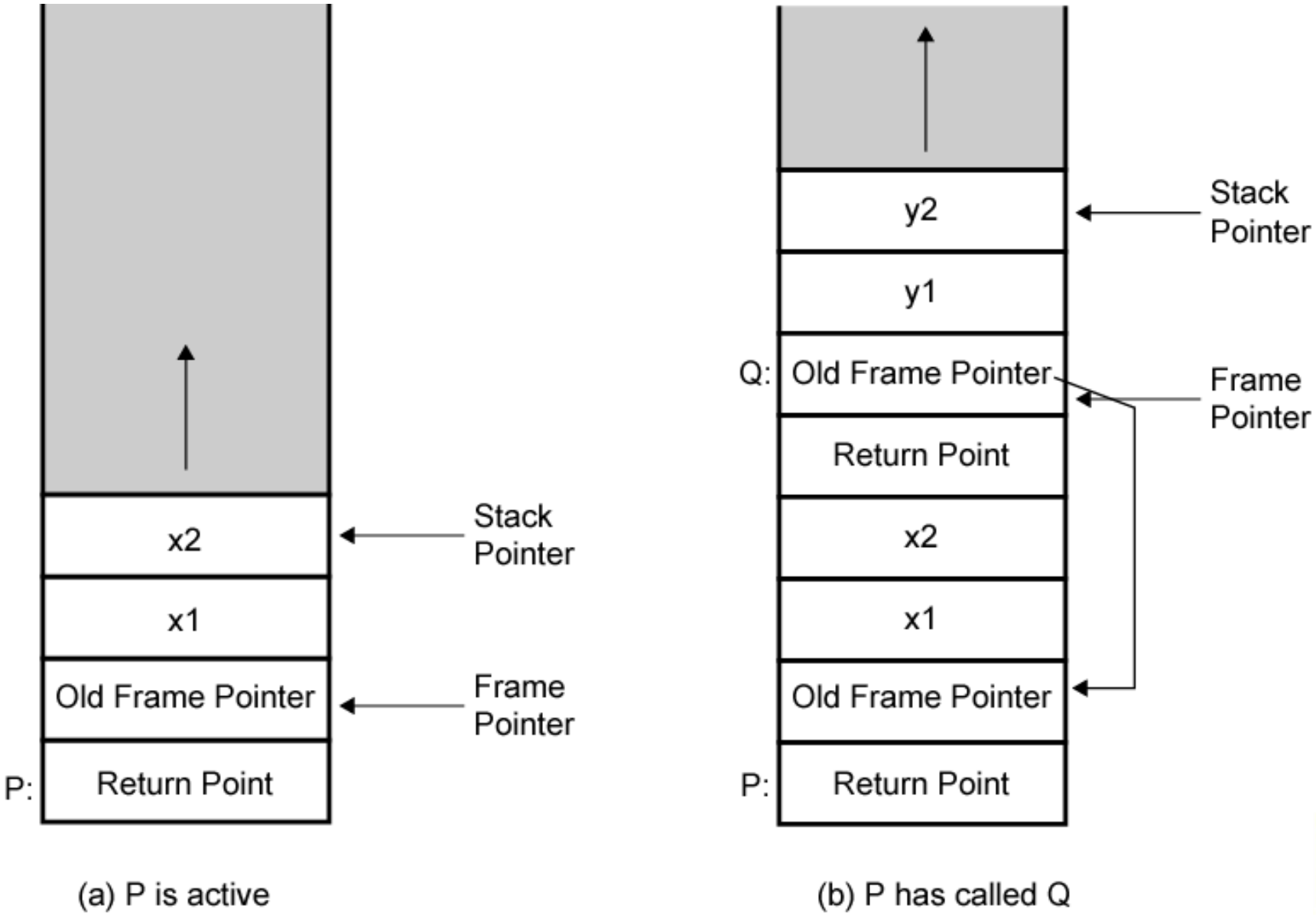
Вгнездени повици на процедури



Користење магацин



Растење на рамката на магацинот со пример



Редослед на бајтите

- Во кој редослед ги читаме броевите кои зафаќаат повеќе од 1 бајт
- пр. (броеви во hex за полесно читање)
- 12345678 може да се запише во 4 x 8 bit локации на следните начини

Редослед на бајтите (пример)

адреса	вредност(1)	вредност(2)
184	12	78
185	34	56
186	56	34
187	78	12

- Значи се чита од горе надолу или од доле нагоре?

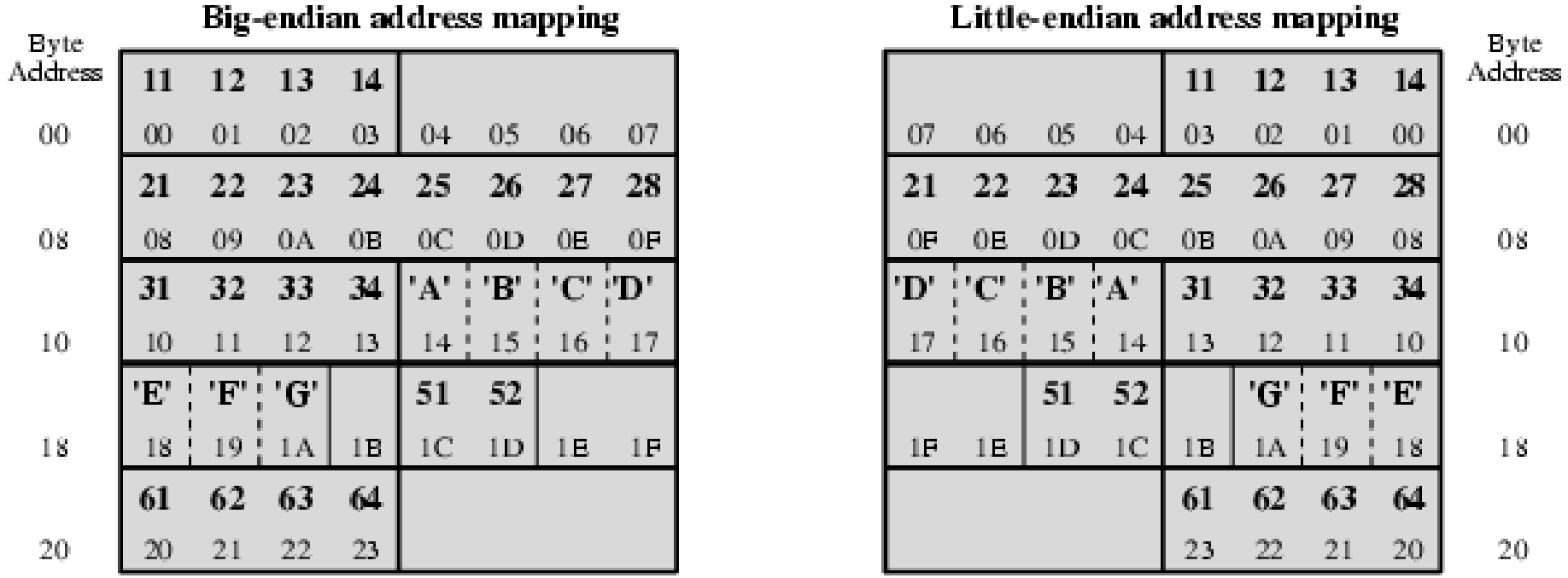
Имиња на редоследот на бајтите

- Проблемот е наречен Endian
- Кај системот лево најзначајниот бајт е на најмалата адреса
- Ова се вика big-endian

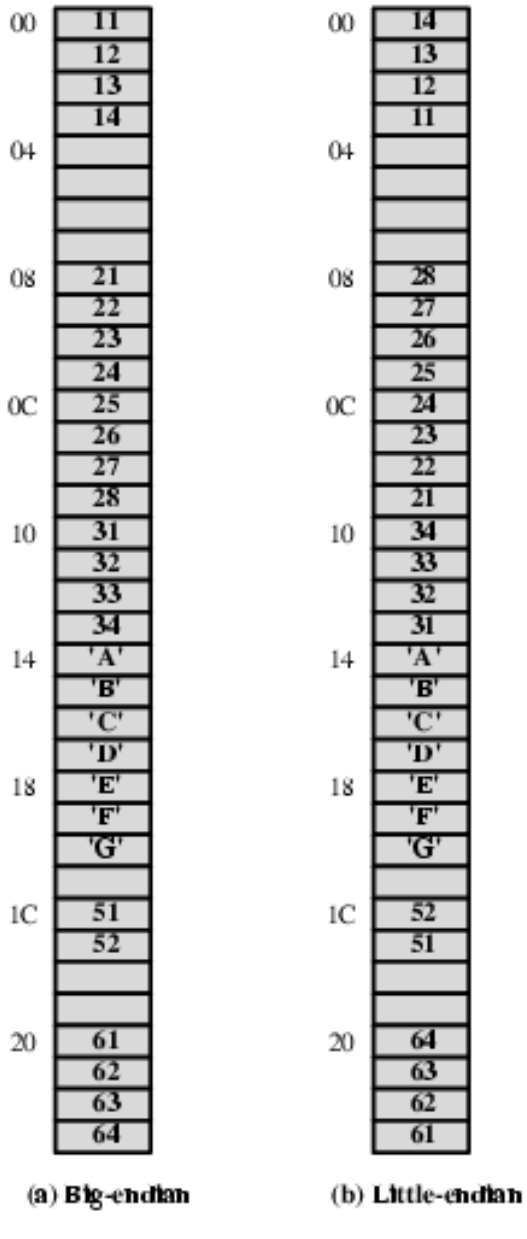
- Кај системот десно најмалку значајниот бајт е на најмалата адреса
- Ова се вика little-endian

Пример на С податочна структура

```
struct{
   int     a;     //0x1112_1314                    word
   int     pad;   //
   double  b;     //0x2122_2324_2526_2728          doubleword
   char*   c;     //0x3132_3334                    word
   char    d[7];  //'A','B','C','D','E','F','G'   byte array
   short   e;     //0x5152                         halfword
   int     f;     //0x6161_6364                    word
} s;
```



Алтернативен поглед на мемориската мапа



Стандард…Кој стандард?

- Pentium (x86), VAX се little-endian
- IBM 370, Motorola 680x0 (Mac), и повеќето RISC се big-endian
- Internet е big-endian
  - Пишувањето на Internet програми на PC е малку незгодно!
  - WinSock обезбедува htoi и itoh (Host to Internet & Internet to Host) функции за претворба

Инструкциско множество: режими на адресирање и формати

Chapter 11
Instruction Sets: Addressing Modes and Formats

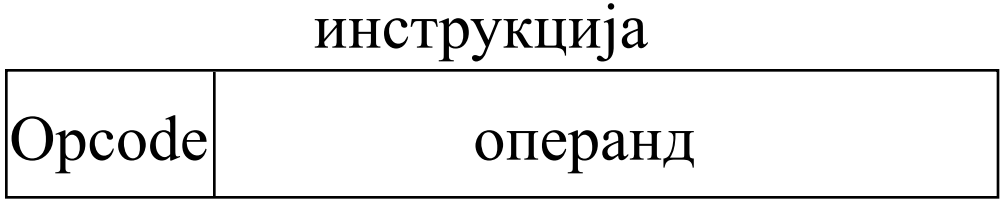
Режими на адресирање

- Непосредно
- Директно
- Индиректно
- Регистерско
- Регистерско индиректно
- Индексирано
- Магацински

Непосредно адресирање

- Операндот е дел од инструкцијата
- операнд = адресно поле
- пр. ADD 5
  - додади 5 на содржината на акумулаторот
  - 5 е операндот
- Нема мемориска референца за земање податок
- Брзо
- Ограничен опсег

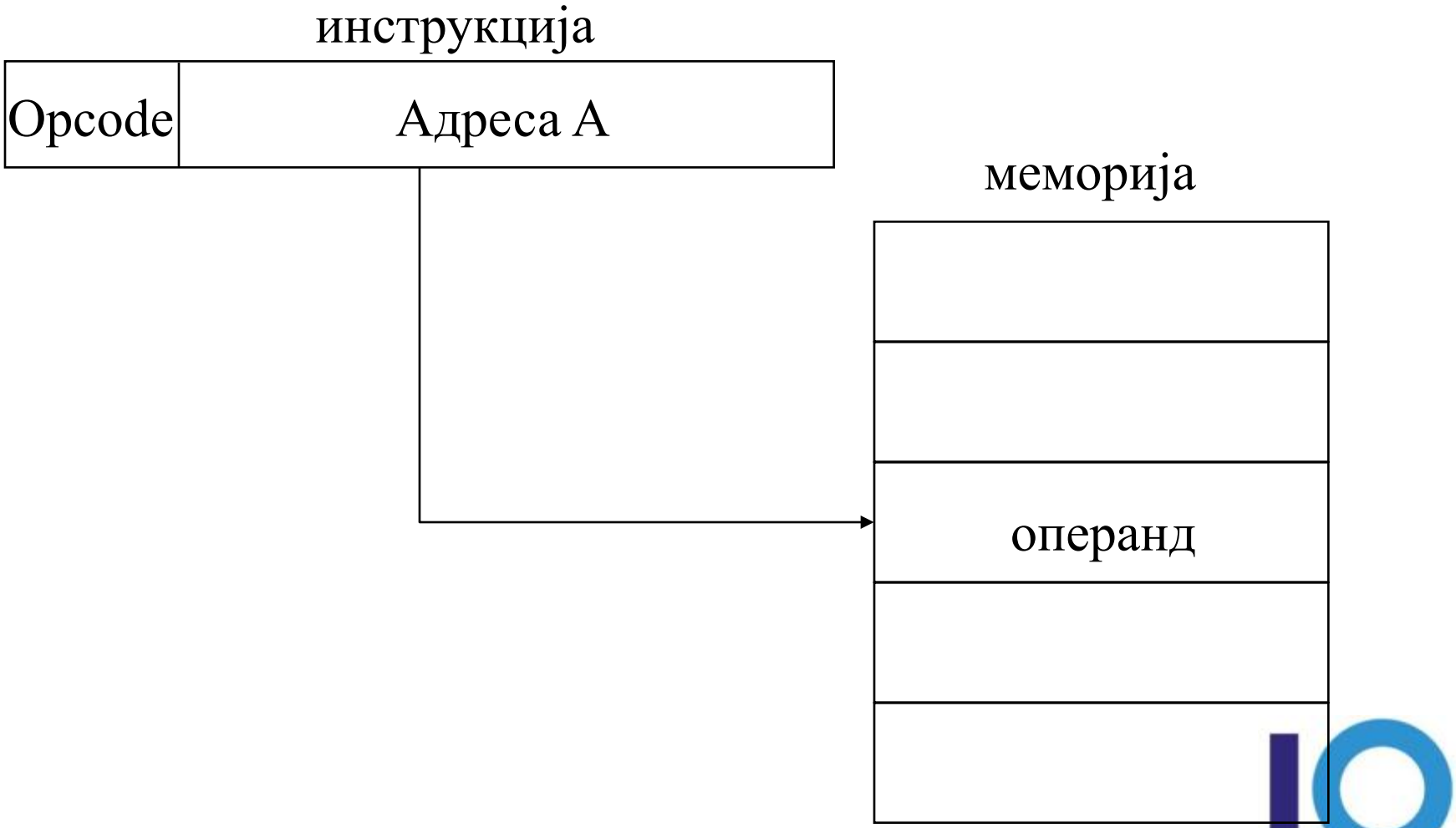
Дијаграм на непосредно адресирање



Директно адресирање

- Адресното поле содржи адреса на операндот
- Ефективна адреса (EA) = адресно поле (A)
- пр. ADD A
  - Додади ја содржината на ќелијата A на акумулаторот
  - Види во меморија на адреса A за да го најдеш операндот
- Една мемориска референца за пристап до податок
- Нема дополнителни пресметки за да се пресмета EA
- Ограничен адресен простор

Дијаграм на директно адресирање



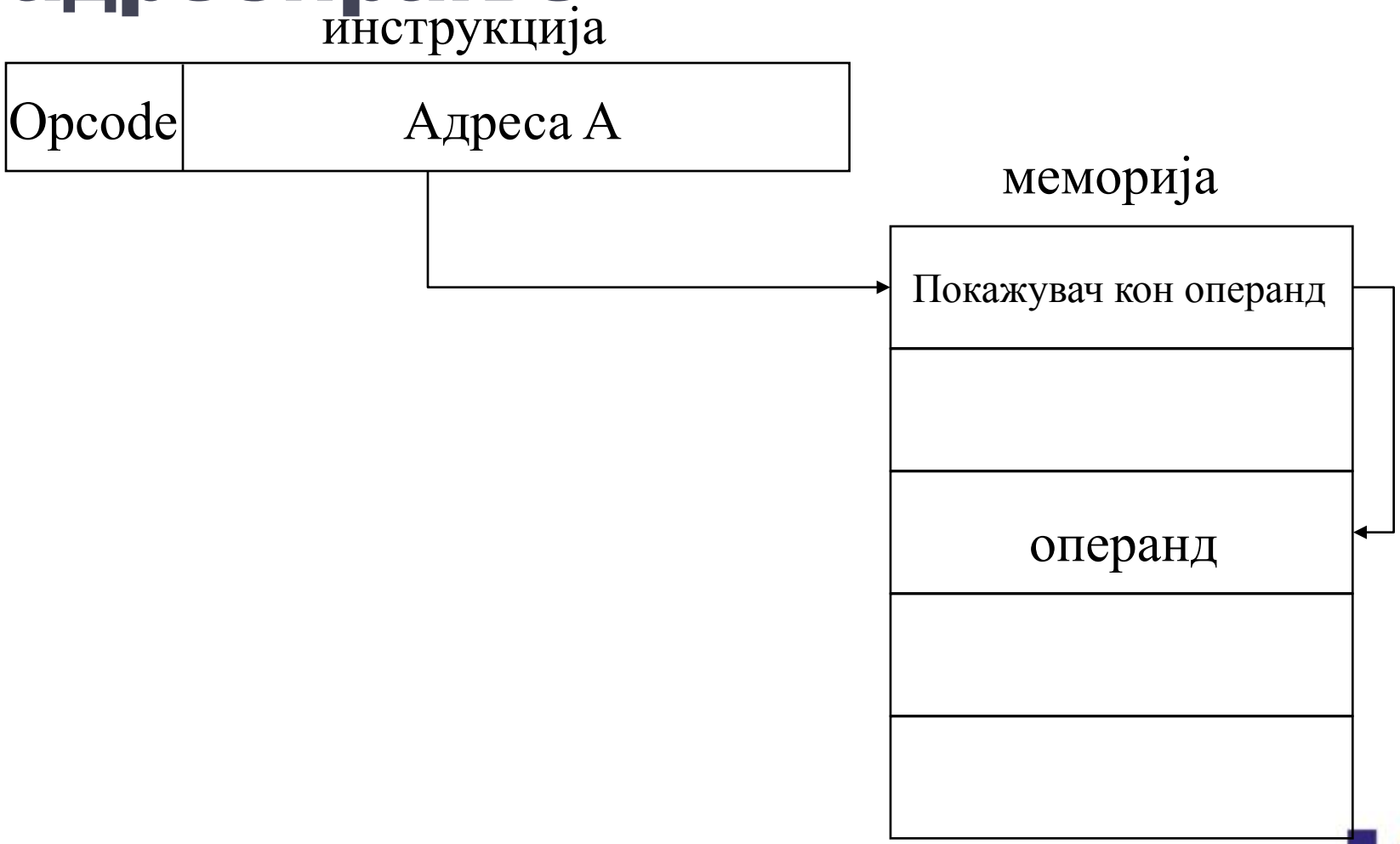
Индиректно адресирање(1)

- Мемориска ќелија кон која покажува адресното поле чија содржина е адресата на (покажувач кон) операндот
- EA = (A)
  - Види во A, најди ја адресата (A) и види таму за операндот
- пр. ADD (A)
  - Додади ја содржината на ќелијата кон која покажува содржината на A на акумулаторот

Индиректно адресирање(2)

- Голем адресен простор
- 2^n каде n = должина на зборот
- Може да е вгнездено, со повеќе нивоа, каскадно
  - пр. EA = (((A)))
- Повеќе мемориски пристапи за да се најде операндот
- Значи побавно

Дијаграм на индиректно адресирање



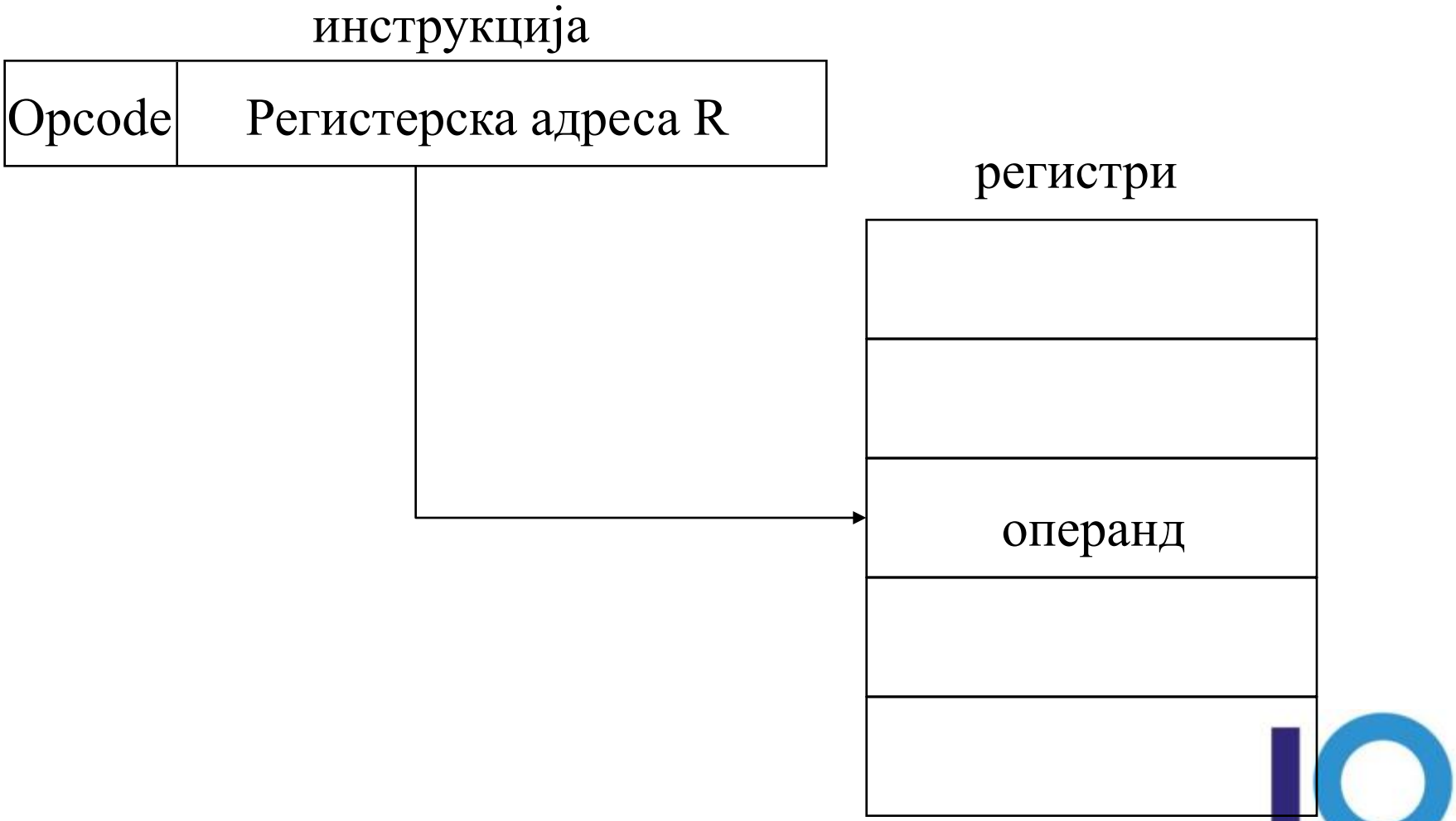
Регистерско адресирање (1)

- Операндот се чува во регистер кој се именува во адресното поле
- EA = R
- Ограничен број на регистри
- Потребно е мало адресно поле
  - Пократки инструкции
  - Побрзо земање инструкција

Регистерско адресирање(2)

- Нема мемориски пристап
- Многу брзо извршување
- Многу ограничен адресен простор
- Повеќе регистри даваат подобри перформанси
  - Бара добро асемблерско програмирање или компајлер
  - N.B. C програмирање
    - register int a;

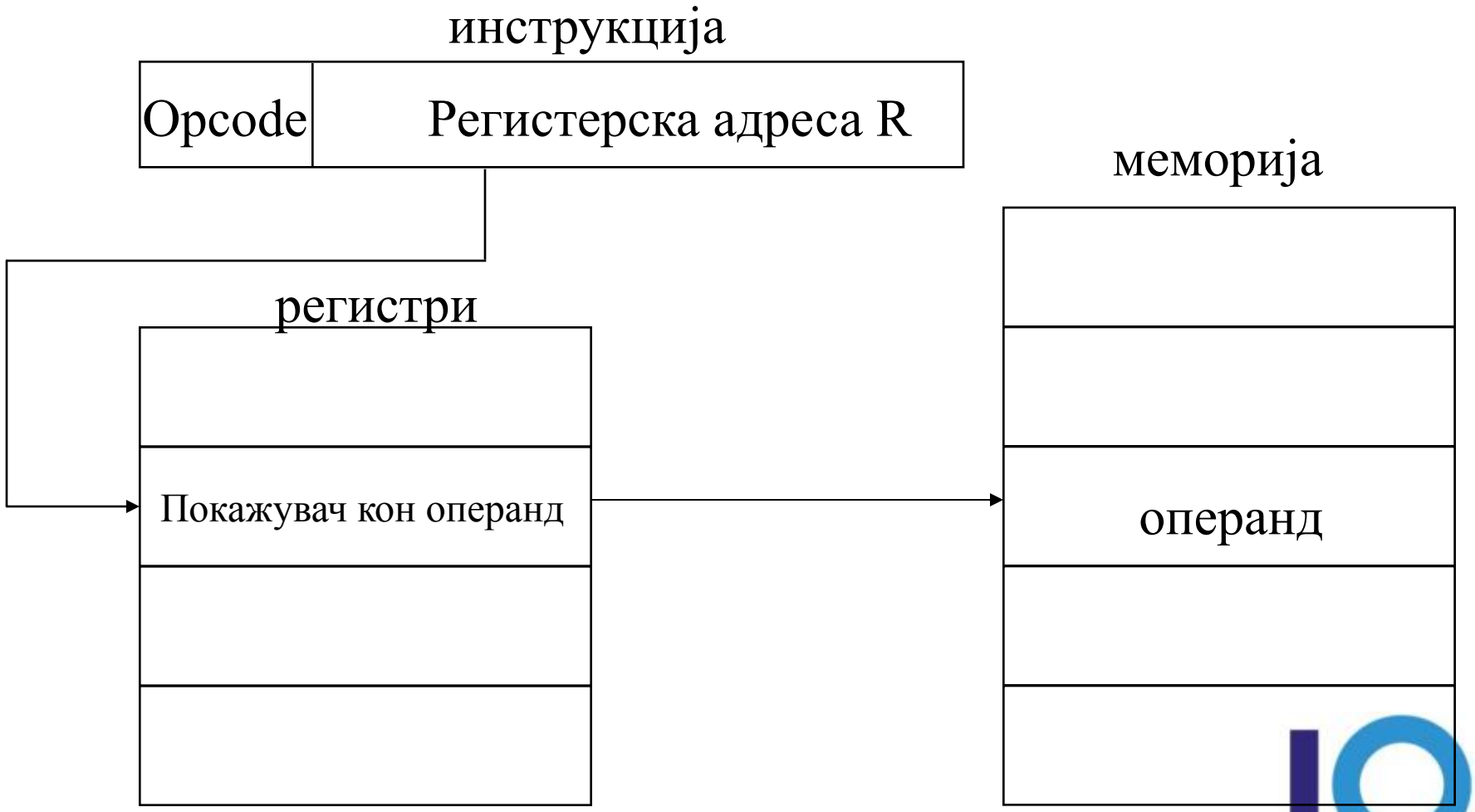
Дијаграм на регистерско адресирање



Регистерско индиректно адресирање

- EA = (R)
- Операндот е во мемориска ќелија кон која покажува содржината на регистерот R
- Голем адресен простор (2^n)
- -1 мемориски пристап во споредба со индиректното адресирање

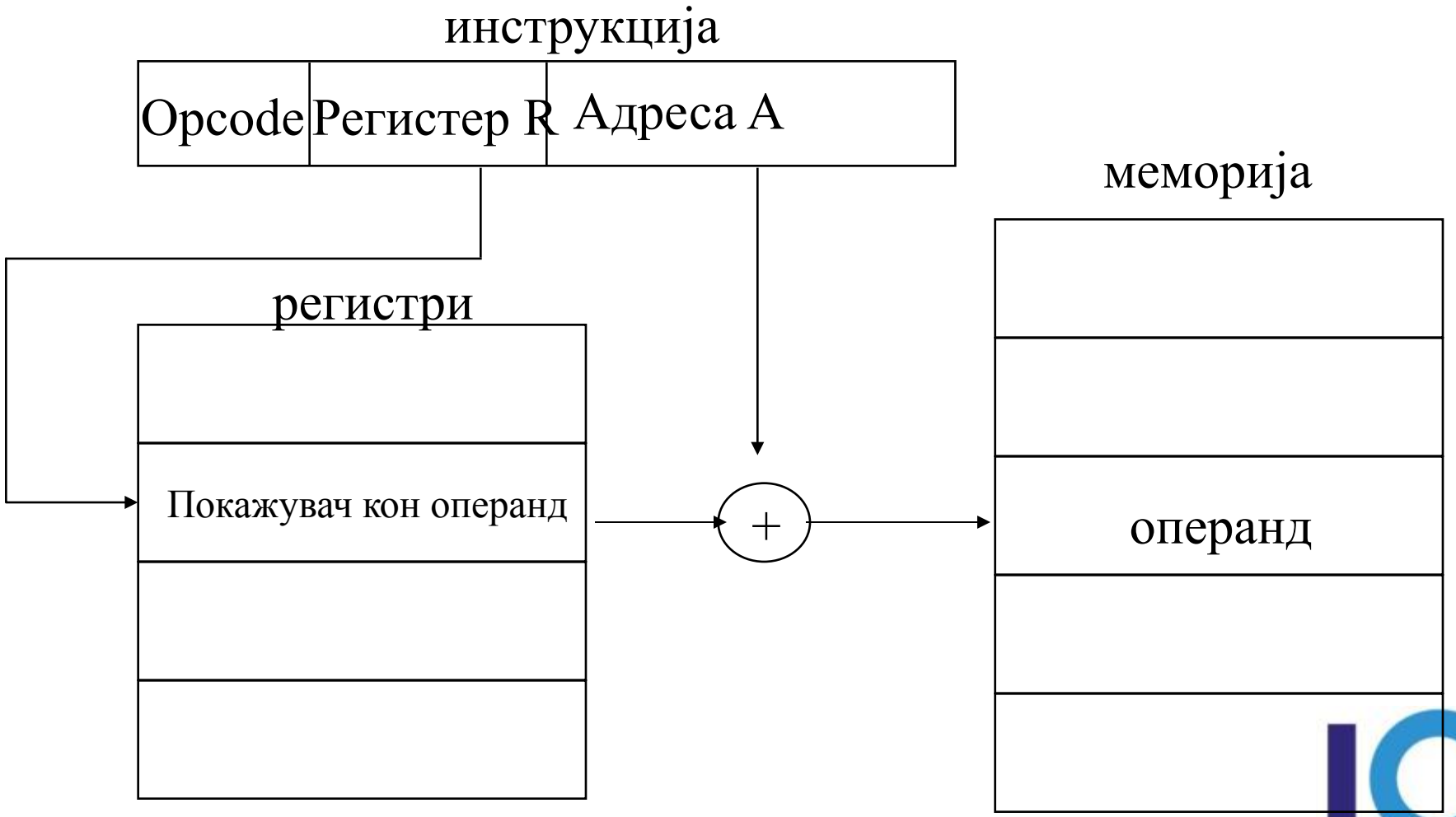
Дијаграм на регистерско индиректно адресирање



Адресирање со поместување

- EA = A + (R)
- Адресното поле има 2 вредности
  - A = основна вредност - база
  - R = регистер кој го содржи поместувањето
  - Или обратно

Дијаграм на адресирање со поместување



Релативно адресирање

- Верзија на адресирање со поместување
- R = програмски бројач, PC
- EA = A + (PC)
- На пр. земи операнд од A-тата ќелија почнувајќи од локацијата каде моментално покажува PC

Базно-регистерско адресирање

- A го чува поместувањето
- R го чува покажувачот кон базната адреса
- R може да е експлицитна или имплицитна
- пр. сегментни регистри во 80x86

Индексно адресирање

- A = база
- R = поместување
- EA = A + R
- Добро за пристап до поле
  - EA = A + R
  - R++

комбинации

- Postindex
- EA = (A) + (R)

- Preindex
- EA = (A+(R))

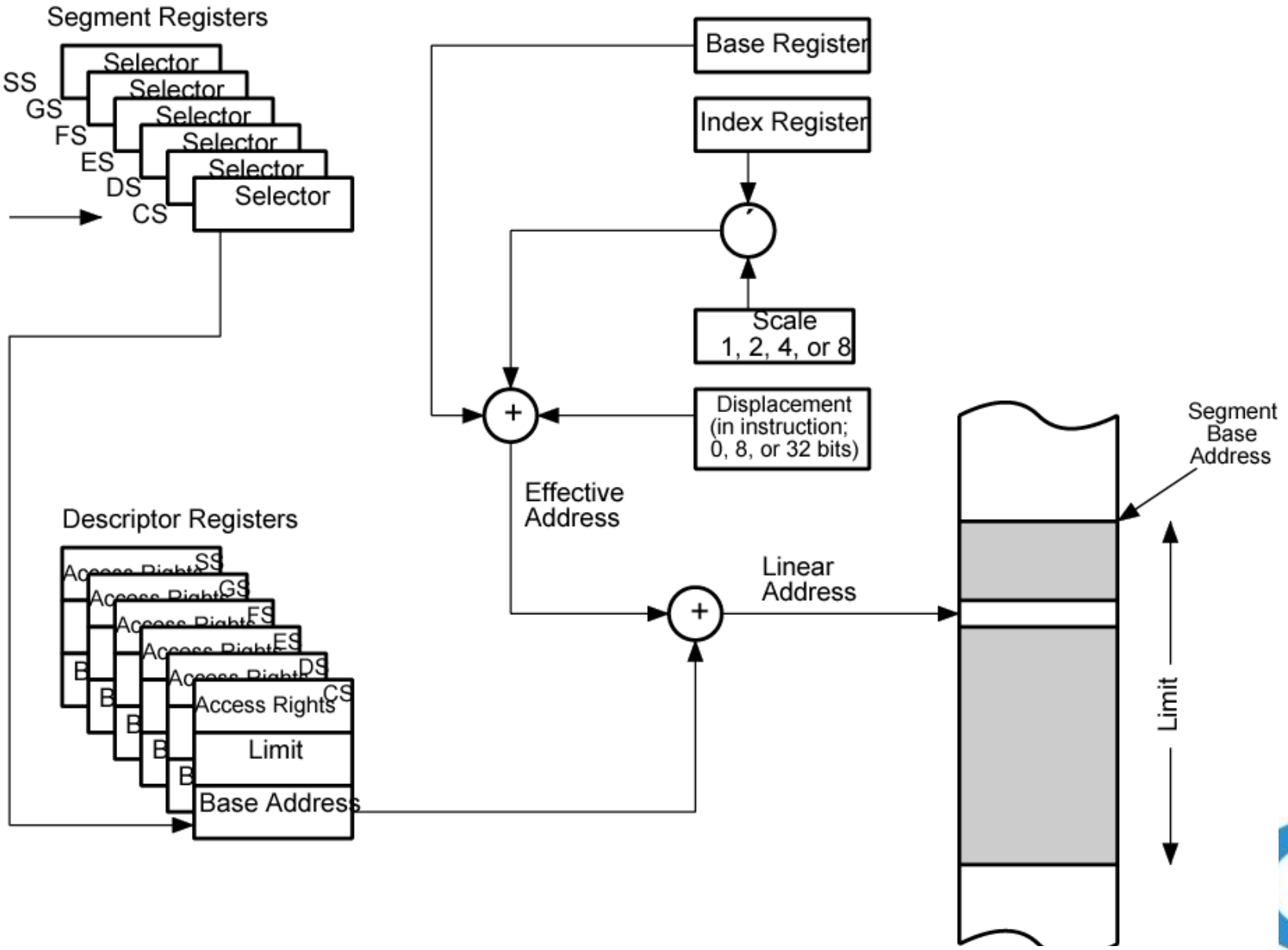
Магацинско адресирање

- Операндот е (имплицитно) на врвот на магацинот
- пр.
  - ADD земи ги двата први елементи од магацинот и собери ги

x86 режими на адресирање

- Виртуелната или ефективната адреса е поместување во сегмент
  - Стартна адреса плус поместување = линеарна адреса
- 12 режими на адресирање
  - Непосредно
  - Регистерски операнд
  - Поместување
  - Базно
  - Базно со поместување
  - Скалирано индексно со поместување
  - Базно со индекс и поместување
  - Базно со скалиран индекс и поместување
  - релативно

x86 пресметување на адреса



ARM режими на адресирање вчитај/запиши

- Само инструкции кои референцираат меморија
- Индиректно преку базен регистар плус поместување
- Preindex и Postindex
- Базниот регистар се однесува како индексен регистар
- Поместувањето е или непосредна вредност или друг регистер
- Ако е регистер можно е и скалирање

Инструкциски формати

- Изглед (поделба) на низата битови во инструкцијата
- вклучува opcode
- Вклучува (имплицитно или експлицитно) операнд(и)
- Вообичаено има повеќе од еден инструкциски формат во едно инструкциско множество

Должина на инструкцијата

- Под влијание на и влијае врз:
  - Големината на меморијата
  - Организацијата на меморијата
  - Структурата на магистралата
  - Комплексноста на процесорот
  - Брзината на процесорот
- Балансирање помеѓу репертоар на моќни инструкции и заштеда на простор

Алокација на битови

- Број на режими на адресирање
- Број на операнди
- Регистри наспроти меморија
- Број на регистерски множества
- Опсег на адреси
- Грануларност на адреси

асемблер

- Машината запишува и разбира бинарни инструкции
- пр. N= I + J + K initialize I=2, J=3, K=4
- Програмата започнува на локација 101
- Податоците започнуваат на 201
- код:
- Вчитај ја содржината на 201 во AC
- Додади ја содржината на 202 на AC
- Додади ја содржината на 203 на AC
- Запиши ја содржината на AC на 204
- Напорно и подложни на грешки

Подобрувања

- Се користи хекса наместо бинарно
  - Код како серија од линии
    - Хекса адреси и мемориски адреси
  - Треба да се преведе автоматски со помош на програма
- Се додаваат симболички имиња и мнемоници за инструкции
- Три полиња по линија
  - Адреса на локацијата
  - opcode со три букви
  - Ако има мемориска референца: адреса
- Потребна е покомплексна програма за превод

Програма: бинарно

Address	Contents			
101	0010	0010	0000	0001
102	0001	0010	0000	0010
103	0001	0010	0000	0011
104	0011	0010	0000	0100
201	0000	0000	0000	0010
202	0000	0000	0000	0011
203	0000	0000	0000	0100
204	0000	0000	0000	0000

хексадецимално

Address	Contents
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

Симболичко програмирање

Address	Instruction	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

Симболички адреси

- Првото поле (адреса) сега е симболичко
- Мемориската референца во третото поле сега е симболичка
- Сега имаме асемблерски јазик и потребен е асемблер за превод
- Асемблер се користи за некои видови системско програмирање
  - Компајлери
  - I/O процедури

Асемблерска програма

Label	Operation	Operand
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

Структура и функција на процесорот

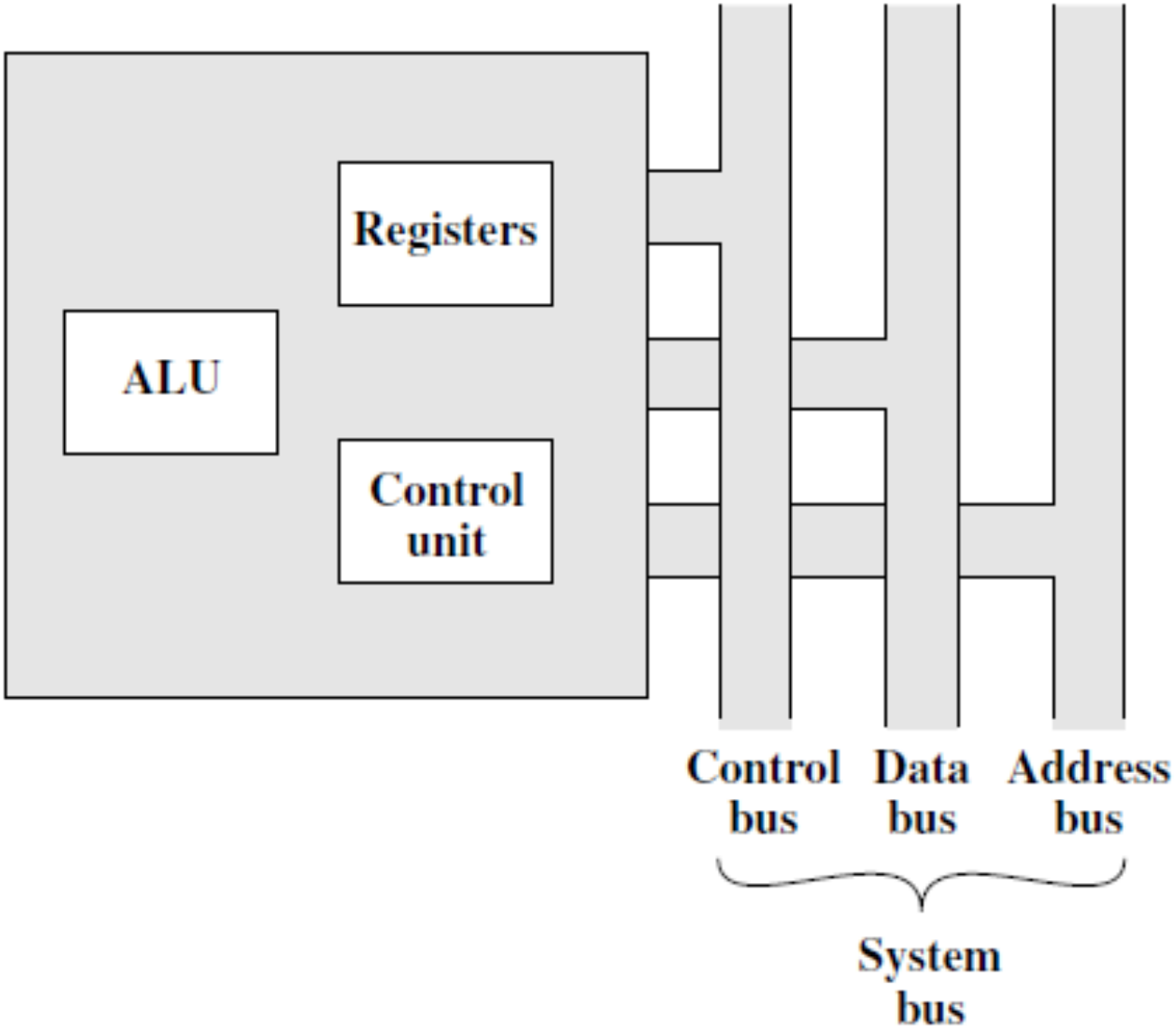
Chapter 12
PROCESSOR STRUCTURE AND FUNCTION

Што мора да прави процесорот ?

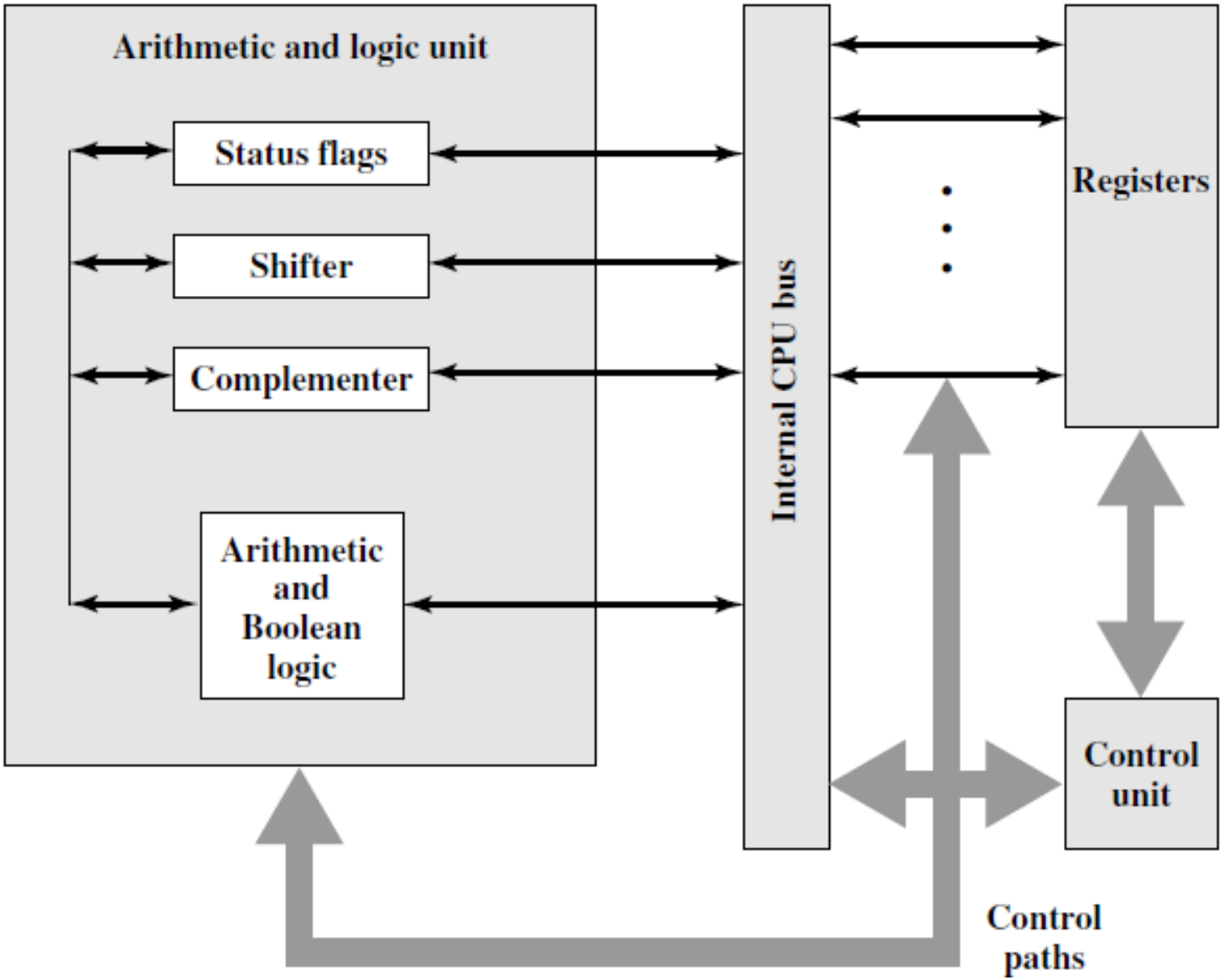
- Земи инструкција
- Интерпретирај ја инструкцијата
- Земи податок
- Обработи го податокот
- Запиши податок

- За што е потребно привремено запишување податоци – има потреба од мала внатрешна меморија

Поедноставен поглед на процесорот



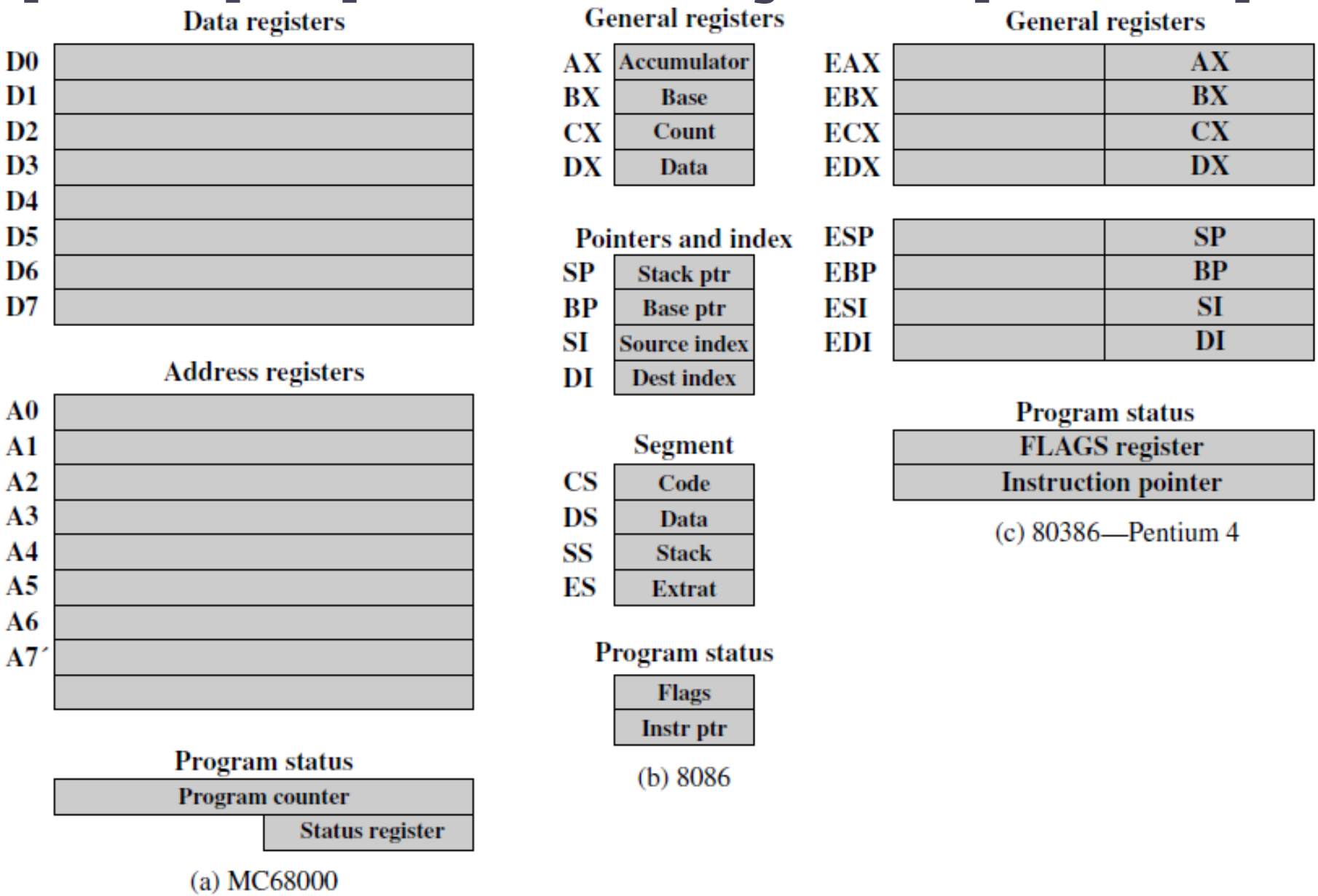
Внатрешна структура на процесорот



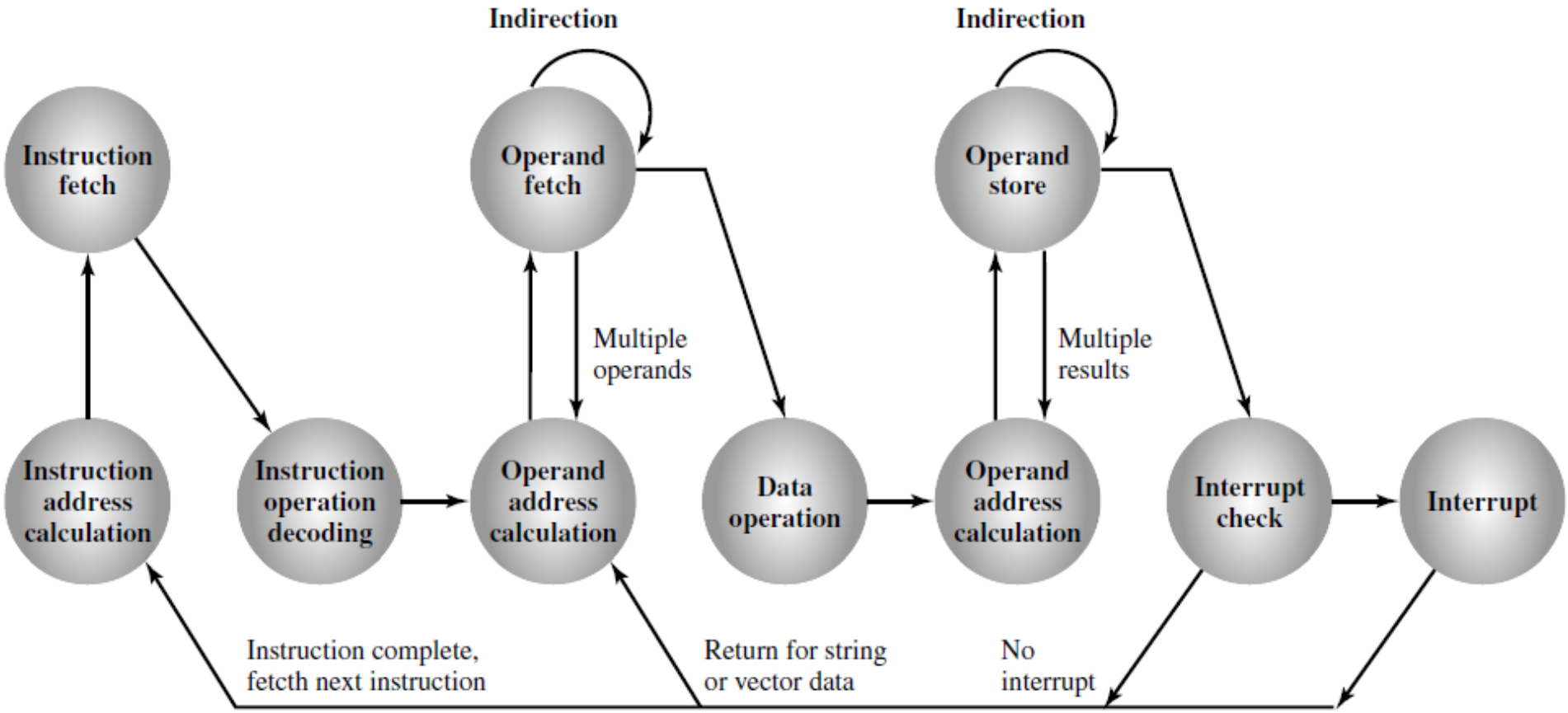
Организација на регистрите

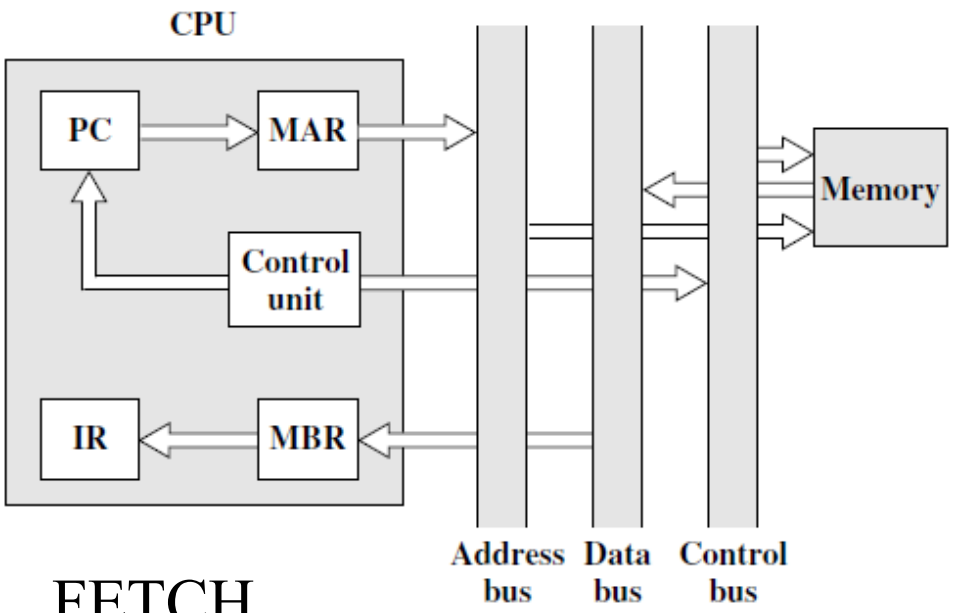
- Кориснички видливи регистри
  - Може да се референцираат во асемблер
    - За општа намена
    - Податочни
    - Адресни
      - Сегментни, индексни, покажувач кон магацин
    - Условни кодови (знаменца)
- Контролни и статусни регистри
  - Ги користи контролната единица
    - PC, IR
    - MAR – memory address register
    - MBR – memory buffer register
    - Статусни регистри за прекини
    - SP

Пример организација на регистри

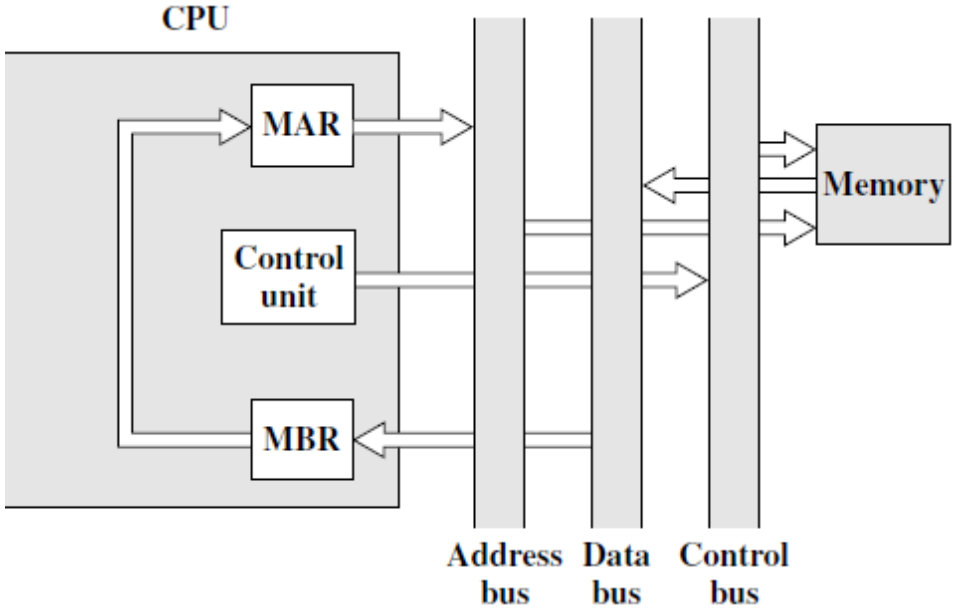


Инструкциски циклус

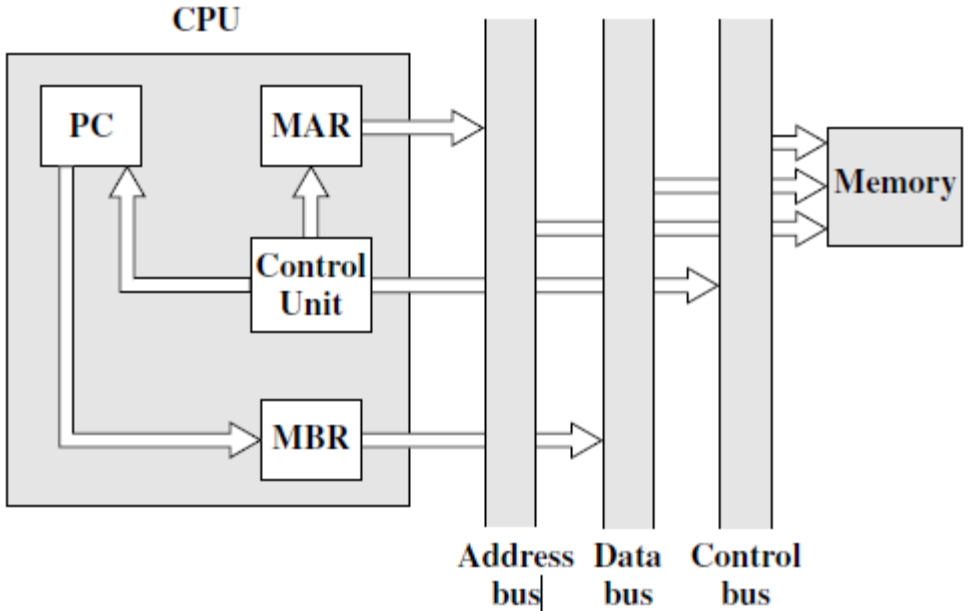




FETCH

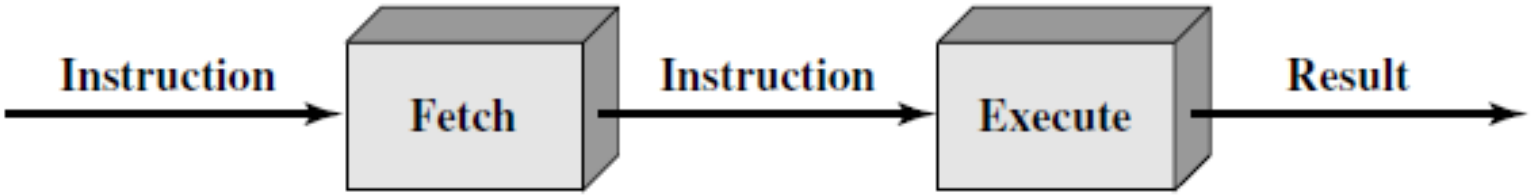


INDIRECT



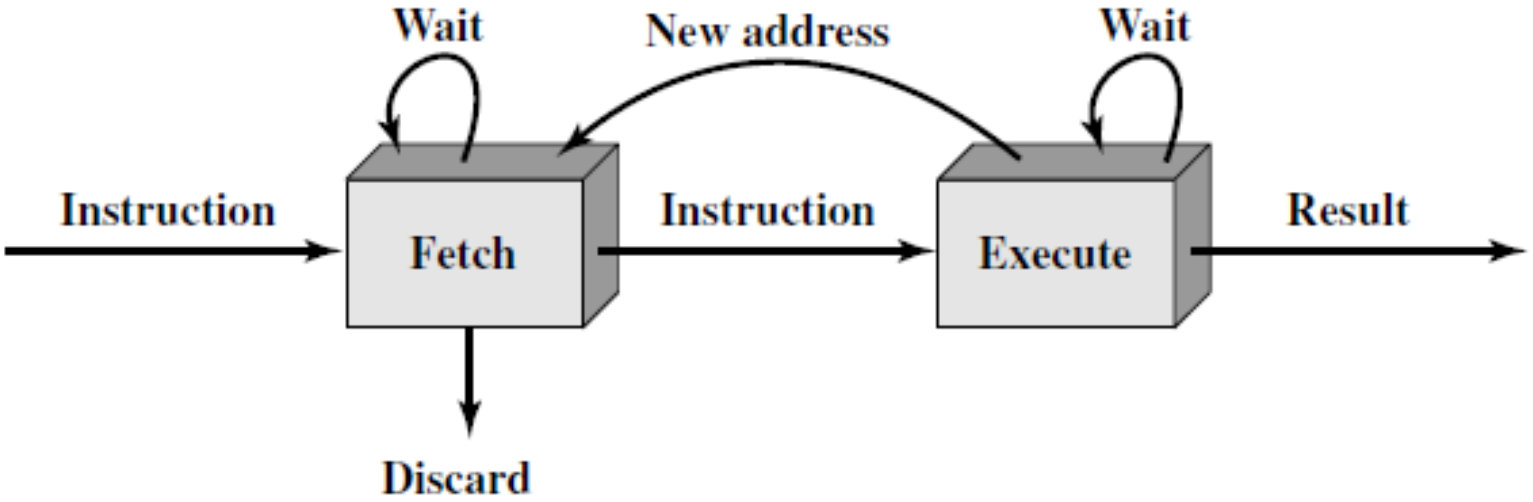
INTERRUPT

Проточност – PIPELINE на инструкциите



(a) Simplified view

instruction prefetch



(b) Expanded view

проблем

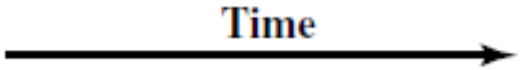
- Времето на извршување е обично подолго од времето на земање инструкција
- Извршувањето вклучува читање и запишување операнди
- Појава на условен скок
  - Секогаш се прави земање на следната инструкција па ако условот за скок не е исполнет не се губи време

Поделба на повеќе фази

- **Земи инструкција (FI)**
- **Декодирај инструкција (DI)**
- **Пресметај операнди (CO)** - одреди ја ефективната адреса на операндите
- **Земи операнди (FO)**
- **Изврши инструкција (EI)**
- **Запиши операнд (WO)**

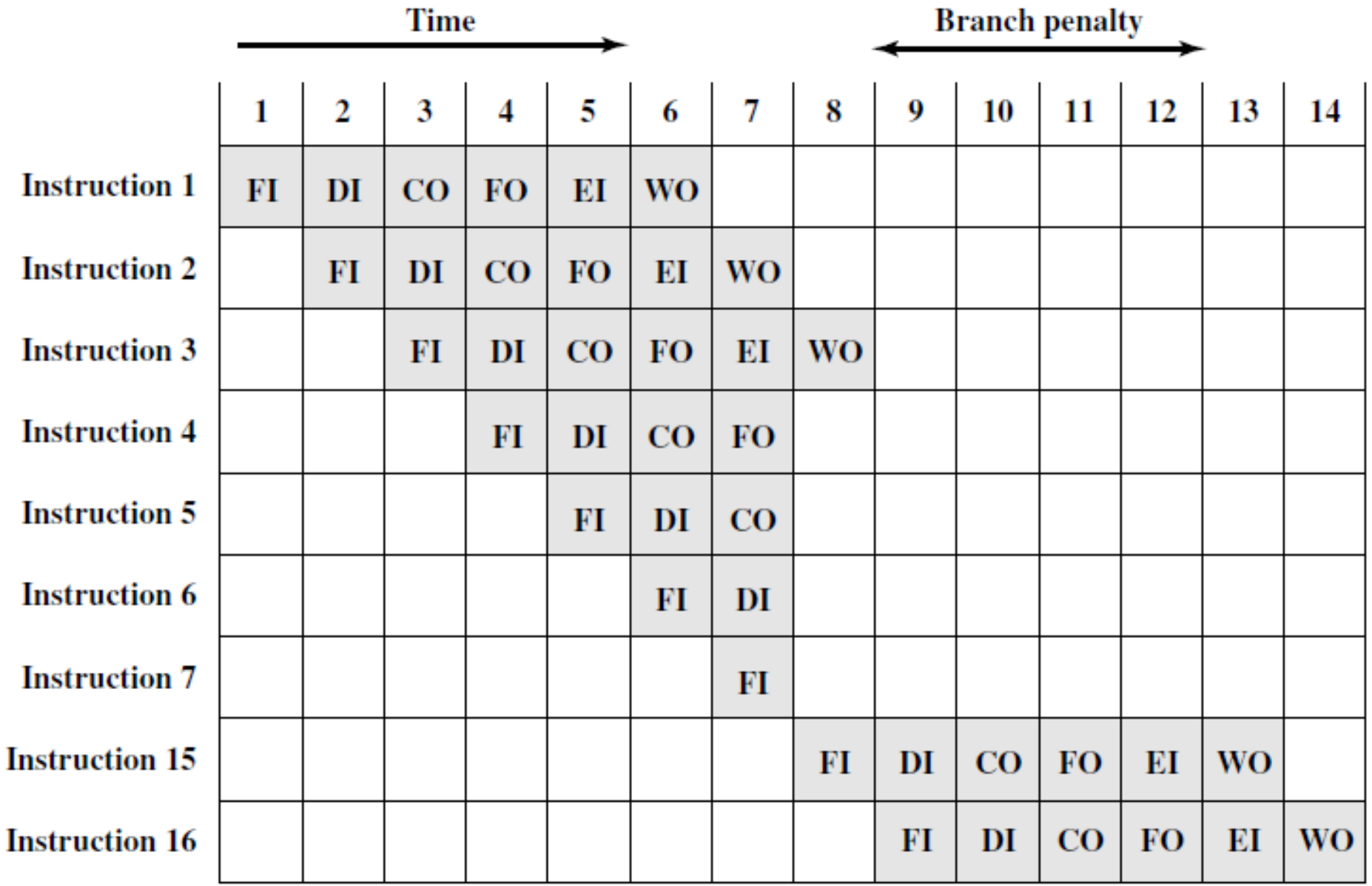
- Вака фазите траат отприлика исто време

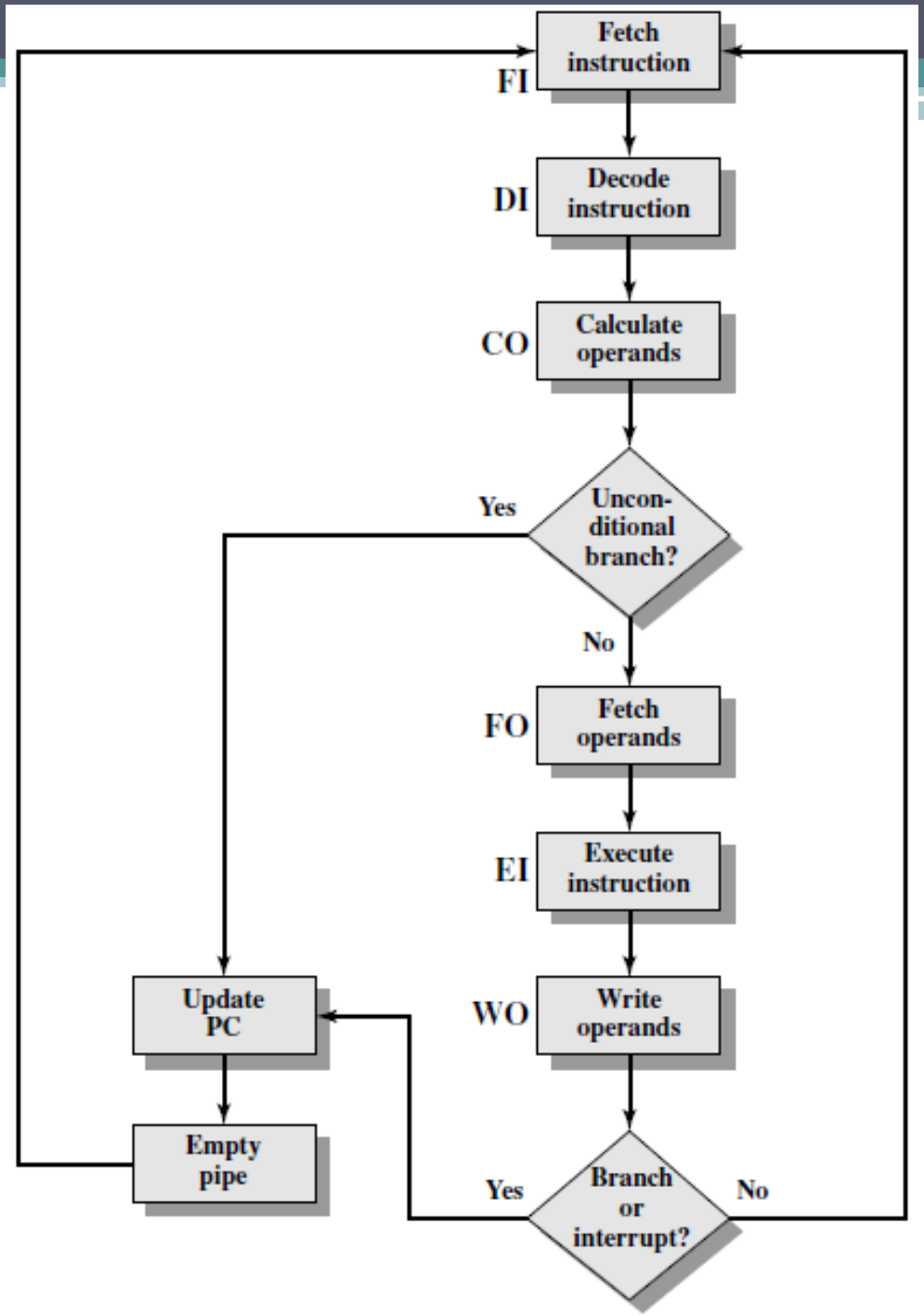
Временски дијаграм



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Кога има скок...




Fetch instruction
FI
Decode instruction
DI
Calculate operands
CO
Yes
Uncon-ditional branch?
No
Fetch operands
FO
Execute instruction
EI
Update PC
Write operands
WO
Empty pipe
Yes
Branch or interrupt?
No

Одлуки при дизајн на проточност

- Во секоја фаза има одреден overhead за преместување податоци и припрема
  - Ова може значајно да го зголеми времето потребно за извршување на инструкција
  - Посебно е значајно за секвенцијални инструкции кои се логички зависни
- Контролната логика потребна за работа со мемориските и регистерските зависности за оптимизација на проточноста енормно расте со бројот на фази
  - Ова може да доведе до ситуација кога контролната логика е покомплексна од самите фази кои ги контролира

Pipeline hazard – pipeline bubble

- Хазард на ресурси
  - Две или повеќе инструкции во цевката користат исти ресурси
- Податочен хазард
  - Конфликт во пристапот на локација на операнд
    - Read after write – вистинска зависност
    - Write after read – анти зависност
    - Write after write – излезна зависност
- Контролен хазард
  - Хазард на гранање
  - Се прави погрешна одлука при предвидување на гранањето

Справување со гранање

- Multiple streams
  - Да се земат предвид двата можни исходи во посебни протоци
- Prefetch branch target
  - Се презема и следната инструкција и целта на скокот
- Loop buffer
  - Содржи n последно земени инструкции во низа
  - Посебно добро за јамки што може да ги собере во баферот
- Branch prediction
- Delayed branch

Предвидување на гранање

- Постојат различни техники
  - Статични – не зависат од историјата
    - Predict never taken
    - Predict always taken
    - Predict by opcode
  - Динамички – врз основа на старите исходи
    - Taken/not taken switch
    - Branch history table

Компјутери со скратено инструкциско множество RISC

Chapter 13
Reduced Instruction Set Computers

Главни напредоци кај компјутерите (1)

- Концептот на фамилија
  - Ја одвојува архитектурата од имплементацијата
- Микропрограмирана контролна единица
  - Го олеснува дизајнот и имплементацијата на контролната единица
- Кеш меморија
  - Драматично зголемување на перформансите

Главни напредоци кај компјутерите (2)

- Solid State RAM
  - (повеќе кога ќе зборуваме за мемории)
- Микропроцесори
  - Intel 4004 1971
- Проточност - Pipelining
  - Воведува паралелизам во инструкцискиот циклус
- Повеќе процесори
- RISC

Следниот чекор - RISC

- Reduced Instruction Set Computer
  - Драматична промена во однос на историските трендови во процесорската архитектура
- Клучни карактеристики
  - Голем број на регистри за општа намена
  - и/или употреба на компајлер за оптимизирана употреба на регистрите
  - Ограничено и едноставно инструкциско множество
  - Нагласување на оптимизацијата на проточноста на инструкциите

Споредба на процесори

	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

Движечката сила на CISC

- Цената на софтверот далеку ја надминува цената на хардверот
- Се покомплексни јазици на високо ниво
- Семантичка одвоеност
- Води кон:
  - Големо инструкциско множество
  - Повеќе режими на адресирање
  - Хардверска имплементација на наредби од јазик од високо ниво
    - пр. CASE (switch) на VAX

Целта на CISC

- Да се олесни пишувањето на компајлер
- Да се подобри ефикасноста на извршување
  - Комплексни операции во микрокод
- Поддршка за покомплексни јазици од високо ниво

Карактеристики на извршувањето

- Извршени операции
- Користени операнди
- Редослед на извршувањата
- Испитувања направени врз основа на програми напишани во јазик на високо ниво
- Динамички испитувања и мерења за време на извршувањето на програмата

операции

- доделувања
  - Преместување на податоци
- Условни наредби (IF, LOOP)
  - Контрола на редоследот
- Повик на процедура одзема многу време
- Некои наредби во јазик од високо ниво водат до многу операции во машинскиот код

Операнди

- Главно локални скаларни (не поле или структура и слично) променливи
- Оптимизацијата треба да се насочи кон пристап до локални променливи

Повици на процедури

- Бараат многу време
- Зависи од бројот на параметри кои се пренесуваат
- Зависи од нивото на вгнезденост
- Повеќето програми не прават многу повици по кои следат многу враќања
- Повеќето променливи се локални

Импликации

- Најдобра поддршка ќе се даде со оптимизација на најкористените и временски најдолгите карактеристики
- Голем број на регистри или добар компајлер
  - Референцирање на операнди
- Внимателен дизајн на проточност
  - Предвидување на гранање и слично
- Поедноставено (скратено) инструкциско множество

Голем број пристапи до регистри
Large Register File

- Софтверско решение
  - Бара компајлер кој ќе алоцира регистри
  - Алокацијата се базира главно на најкористените променливи во дадено време
  - Има потреба од софистицирана анализа на програмата
- Хардверско решение
  - Повеќе регистри
  - На овој начин повеќе променливи ќе бидат во регистрите

Регистри за локални променливи

- Чување на локалните скаларни променливи во регистри
- Намалување на пристапот до меморија
- Секој повик на процедура (функција) ја менува локалноста
  - Мора да се пренесат параметри
  - Мора да се вратат резултати
  - По враќањето мора да се вратат старите локални променливи

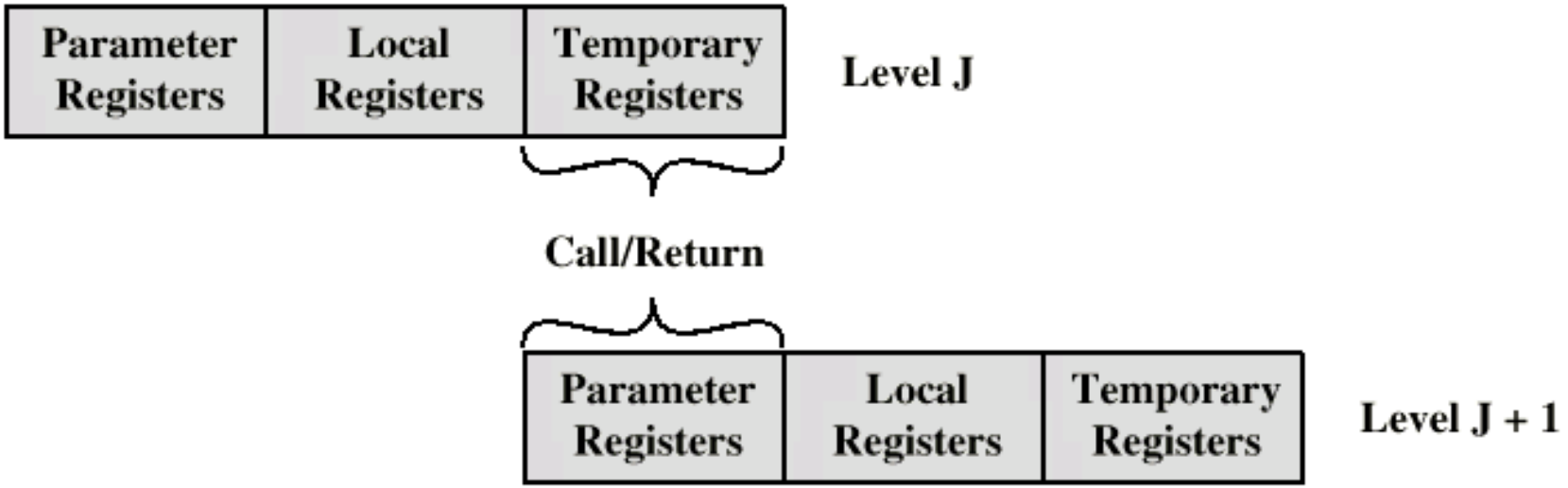
Регистарски прозорци

- Само неколку параметри
- Ограничен опсег на длабочината на повикување
- Користење повеќе мали множества регистри (регистарски прозорци)
  - Повикот користи друго множество
  - Враќањето назад враќа на старото множество
- Во еден момент се гледа само еден прозорец

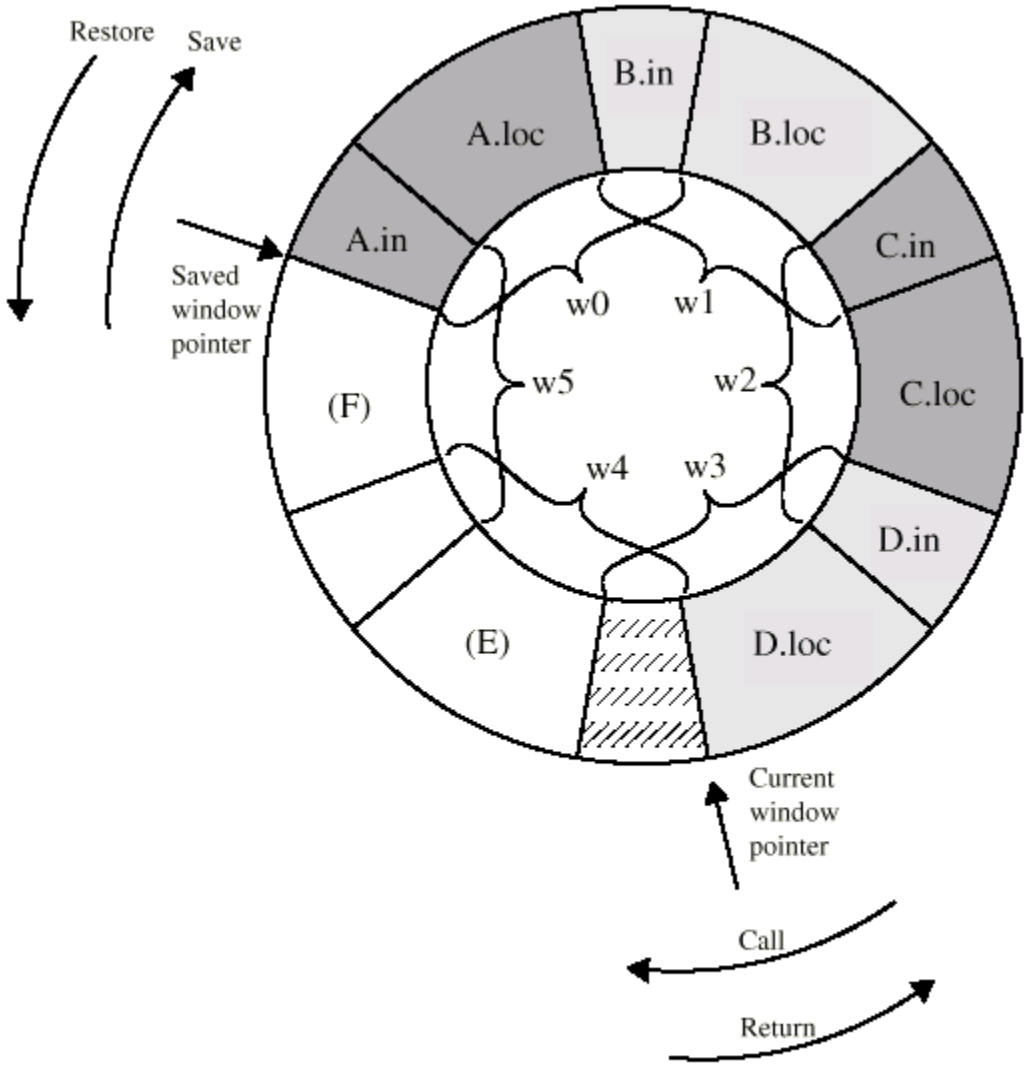
Регистарски прозорци

- Три области во рамките на прозорецот
  - Параметарски регистри
  - Локални регистри
  - Привремени регистри
  - Привремени регистри од едно множество кои се преклопуваат со параметарски регистри од следното множество
    - На овој начин се обезбедува пренесување на параметри без преместување податоци

Преклопувачки регистерски прозорци



Циркуларен бафер



Работа на циркуларниот бафер

- При повик, покажувачот на активниот прозорец се поместува за да покажува на тековно активниот прозорец
- Ако сите прозорци се во употреба, се генерира прекин и најстариот прозорец се запишува во меморија
- Покажувач кон запишаниот прозорец означува каде треба да се врати прозорецот од меморија

Глобални променливи

- Алоцирани од компајлерот во меморија
  - Неефикасно за променливи до кои често се пристапува
- Потребно е множество на регистри за глобални променливи

Зошто CISC (1)?

- Поедноставување на компајлерот?
  - Не е прифатено...
  - Комплексните машински инструкции тешко се искористуваат
  - Оптимизацијата е потешка
- Помали програми?
  - Програмата зафаќа помалку меморија но...
  - Денес меморијата е ефтина
  - Може да не зафаќа помалку битови, туку само да изгледа пократко во симболичка форма
    - Повеќе инструкции бараат подолги op-codes
    - Регистерските референци бараат помалку битови

Зошто CISC (2)?

- Побрзи програми?
  - Пристрасност кон користење на поедноставни инструкции
  - Покомплексна контролна единица
  - Поголем простор за складирање на микропрограмската контрола
  - Така поедноставни инструкции подолго се извршуваат

- Далеку е од јасно дека CISC е соодветното решение

RISC карактеристики

- Една инструкција по машински циклус
- Регистер – регистер операции
- Малку, едноставни режими на адресирање
- Малку, едноставни инструкциски формати
  - Фиксиран Hardwired дизајн (без микрокодирање)
  - Фиксен инструкциски формат
- Повеќе време и напор за компајлирање

RISC наспроти CISC

- Нема јасна граница
- Многу дизајни земаат работи од двете филозофии
- пр. PowerPC и Pentium II

Паралелизам на ниво на инструкција Суперскаларни процесори

Chapter 14
Instruction Level Parallelism
and Superscalar Processors

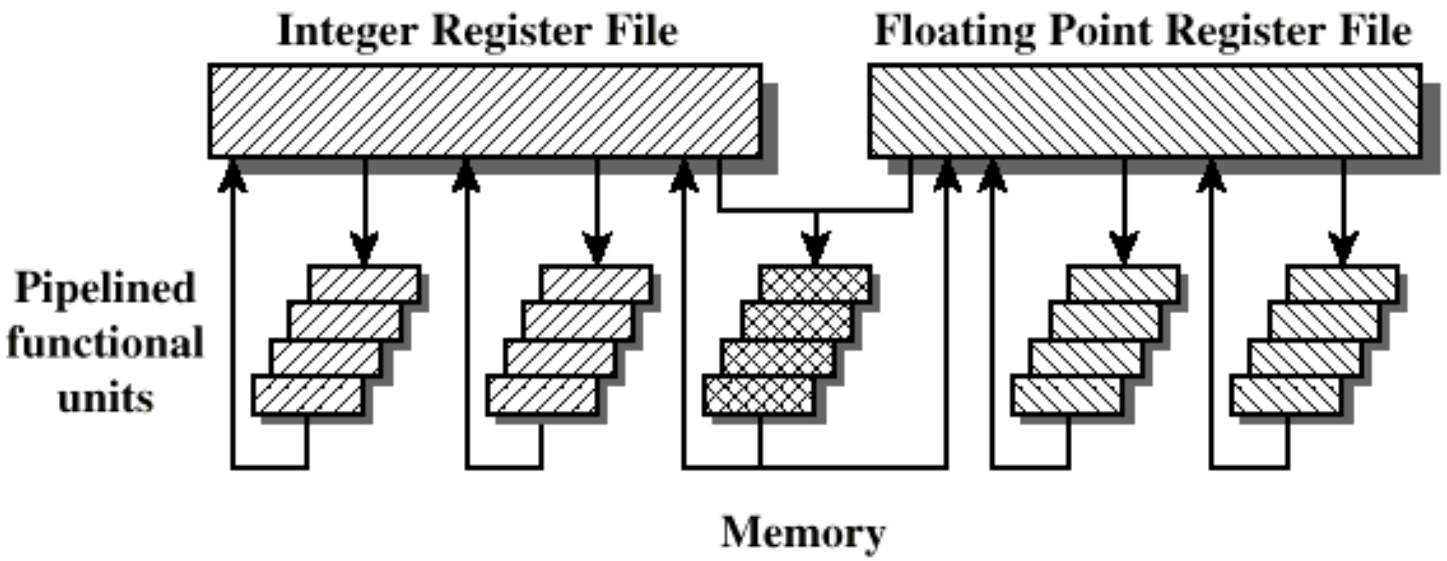
Што е Superscalar?

- Вообичаени инструкции (аритметички, прочитај/запиши, условен скок) може да се иницираат и извршат независно
- Еднакво применливо во RISC и CISC
- Во пракса обично RISC

Зошто Superscalar?

- Повеќето операции се на скаларни вредности
- Со подобрување на овие операции се добива севкупно подобрување

Општа суперскаларна организација

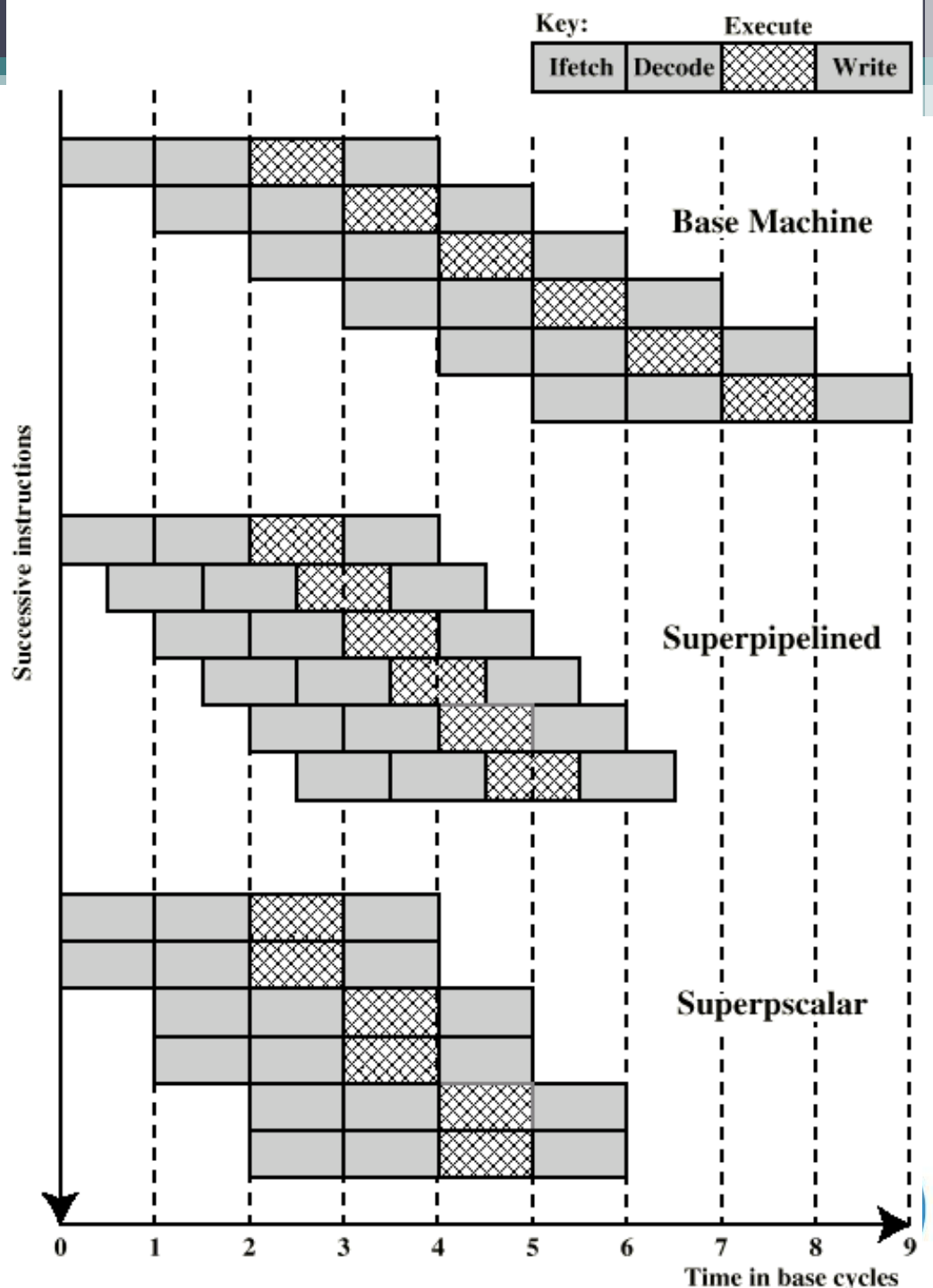


Суперпроточност - Superpipelined

- Многу фази во цевката траат помалку од половина такт
- Со дуплирање на внатрешната брзина на тактот се добиваат 2 задачи по надворешен такт

- Суперскаларите овозможуваат паралелни земи и изврши инструкција

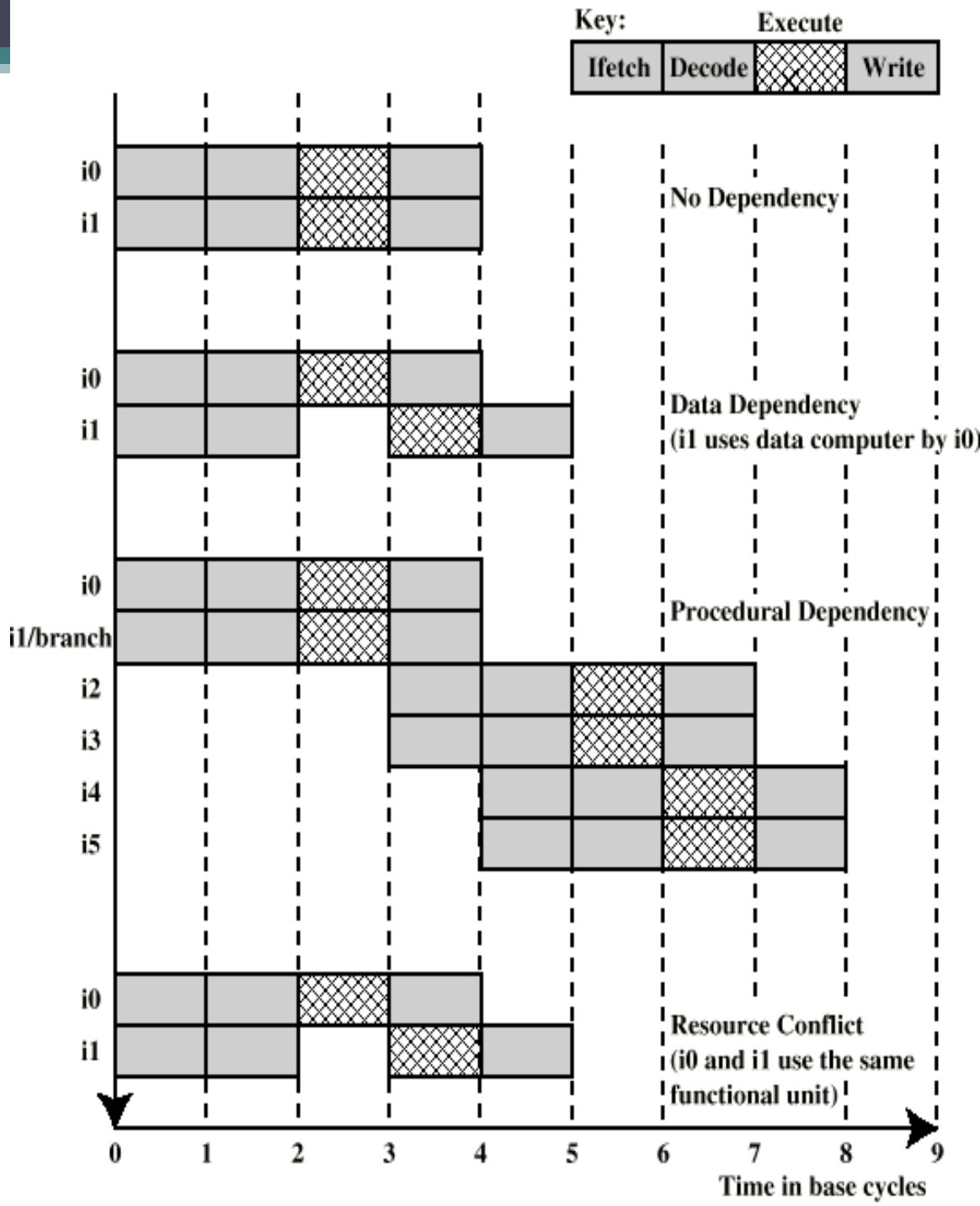
Superscalar v Superpipeline



Ограничувања

- Паралелизам на ниво на инструкција
  - Можност за извршување на повеќе инструкции паралелно
- Оптимизации на ниво на компајлер
- Хардверски техники
- Ограничени од
  - Вистинска податочна зависност WAR
  - Процедурална зависност
    - Инструкции зависни од условен скок
  - Конфликти на ресурси
    - Потреба од истиот ресурс кај 2 или повеќе инструкции
  - Излезна зависност
  - Антизависност

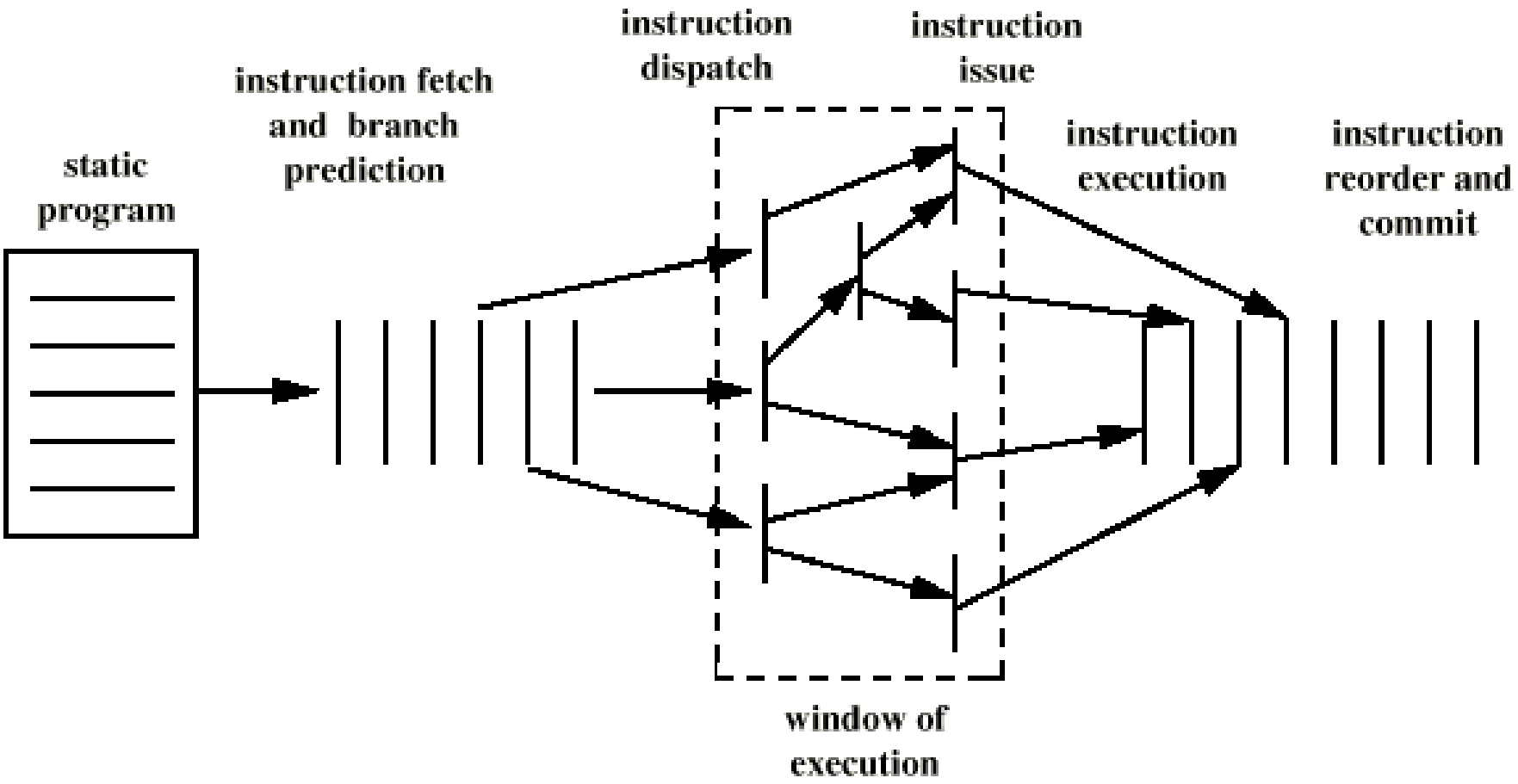
Ефект на зависностите



Дизајн

- Паралелизам на ниво на инструкција
  - Инструкциите во низа се независни
  - Може да се преклопи извршувањето
  - Под влијание на податочна и процедурална зависност
- Машински паралелизам
  - Можноста да се искористи паралелизмот на ниво на инструкција
  - Под влијание на бројот на паралелни проточни цевки

Суперскаларно извршување



Суперскаларна имплементација

- Едновремено земање повеќе инструкции
- Логика за одредување на вистинските зависности кои вклучуваат регистерски вредности
- Механизми за комуникација на овие вредности
- Механизми за иницирање на повеќе инструкции паралелно
- Ресурси за паралелно извршување на повеќе инструкции
- Механизми за извршување на процесната состојба во точен редослед