Bangladesh Govt. & UGC Approved

UNIVERSITY OF GLOBAL VILLAGE (UGV), BARISHAL.

THE FIRST SKILL BASED HI-TECH UNIVERSITY IN BANGLADESH

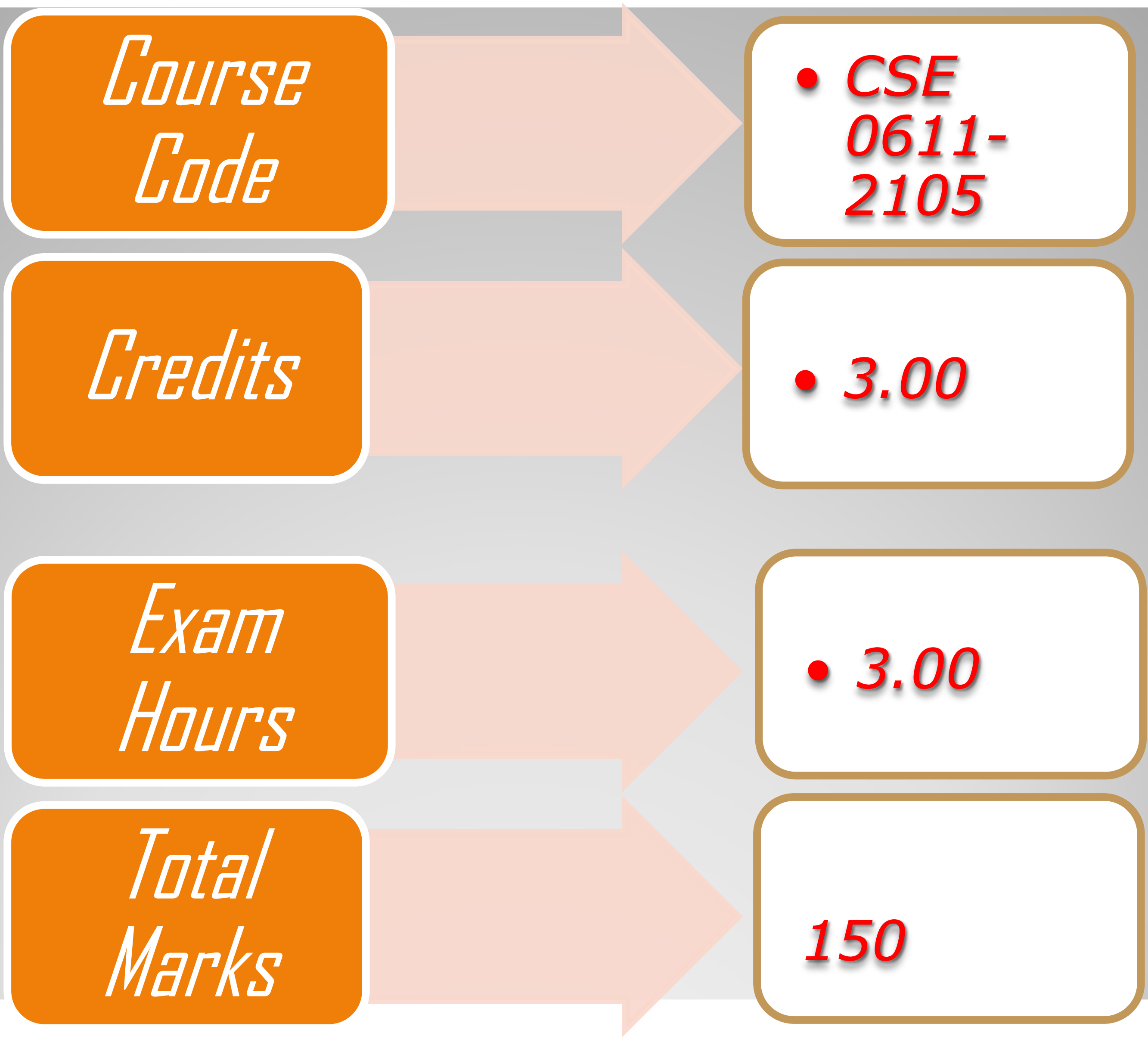
Computer Architecture

Prepared by:

Md. Abdur Razzak

Asst. Professor.

Department of CSE



Course Code

- CSE 0611-2105

Credits

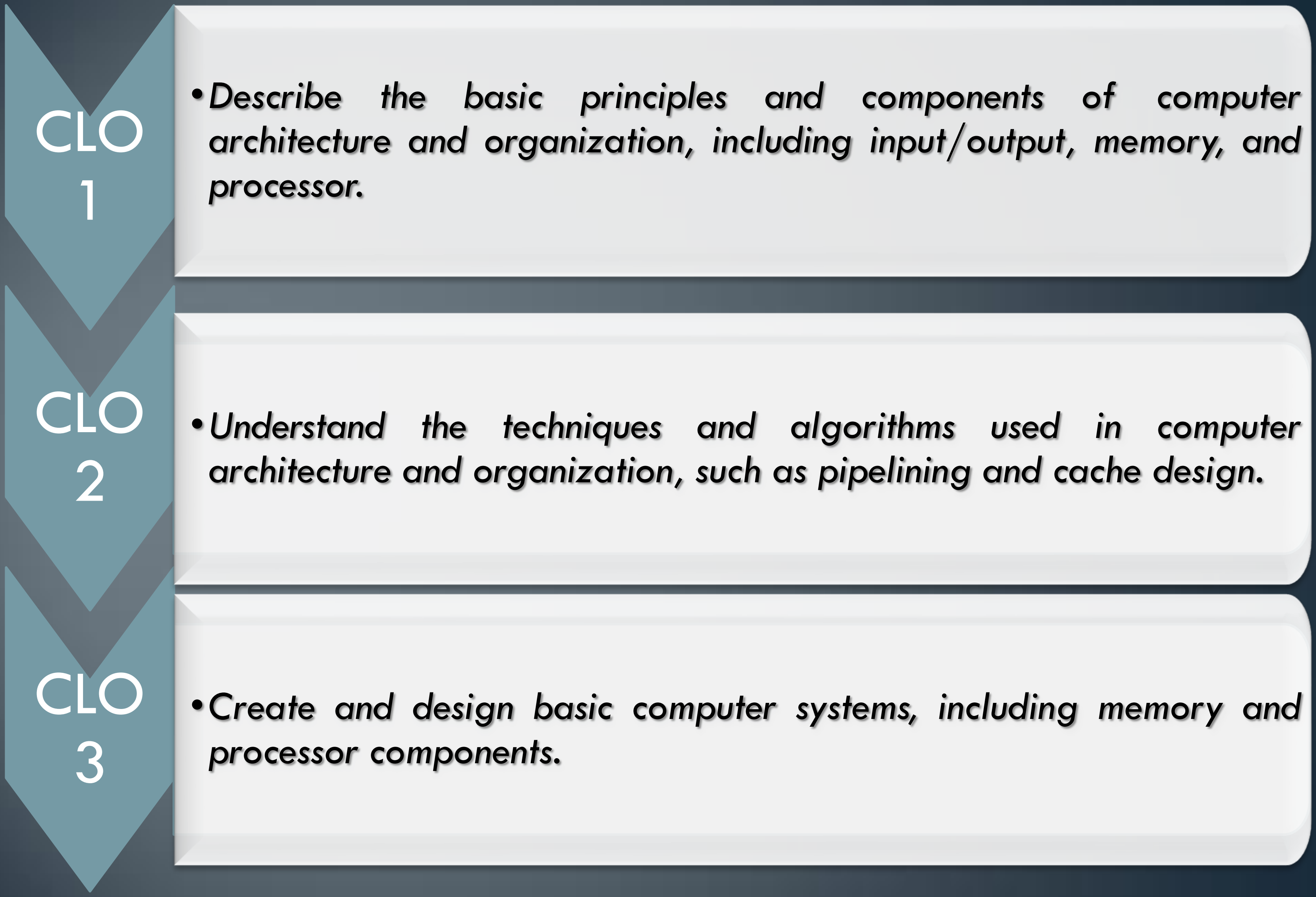
- 3.00

Exam Hours

- 3.00

Total Marks

150



CLO 1

- *Describe the basic principles and components of computer architecture and organization, including input/output, memory, and processor.*

CLO 2

- *Understand the techniques and algorithms used in computer architecture and organization, such as pipelining and cache design.*

CLO 3

- *Create and design basic computer systems, including memory and processor components.*

CLO 4

- *Apply principles of computer architecture and organization to analyse and solve problems related to performance, power consumption, and security.*

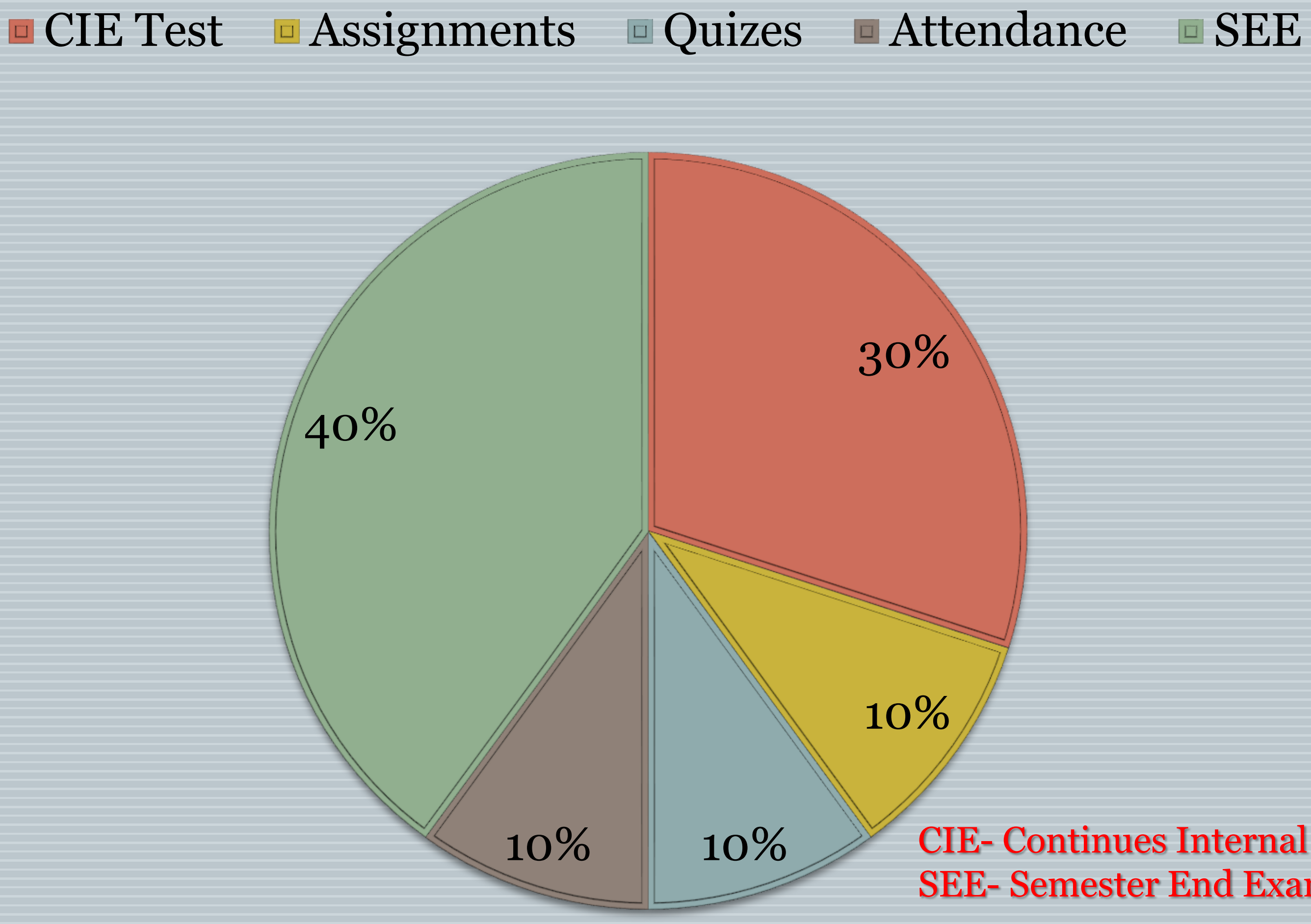
CLO 5

- *Identify emerging trends and technologies in computer architecture and organization and evaluate their potential impact on computer systems.*

Recommended Books:

- "Computer Organization and Design: The Hardware/Software Interface" by David A. Patterson and John L. Hennessy
- "Structured Computer Organization" by Andrew S. Tanenbaum
- "Computer Architecture: A Quantitative Approach" by John L. Hennessy and David A. Patterson

Assessment Pattern



CIE- Continues Internal Evaluation
SEE- Semester End Examination

Detailed Assessment Pattern

CIE- Continuous Internal Evaluation (90 Marks)

Bloom's Category Marks (out of 90)	Tests (45)	Assignments (15)	Quizzes (15)	Attendance (15)
Remember	5	03		
Understand	5	04	05	
Apply	15	05	05	
Analyze	10			
Evaluate	5	03	05	
Create	5			

SEE- Semester End Examination (60 Marks)

Bloom's Category	Test
Remember	7
Understand	7
Apply	20
Analyze	15
Evaluate	6
Create	5

Week No	Topics	Teaching Learning Strategy(s)	Assessment Strategy(s)	Alignment to CLO
1	Introduction to Architecture & Organization	- Interactive discussion - Lecture with multimedia aids - Group discussion - Practical setup exercises	- Feedback through Q&A - Assessment of practical setup exercises	**CLO1**
2	Von Neumann Architectures ENIAC - background	Lecture, Reading Assignments	Quiz, Assignment	**CLO1, CLO2**
3	Instruction Set	Lecture, Case Studies	Quiz, Assignment	**CLO2**
4	**CPU Structure**	**Lecture, Hands-on Labs**	**Quiz, Lab Reports**	**CLO2, CLO3**

5	**Memory Hierarchy**	**Lecture, Case Studies**	**Quiz, Case Study Reports**	**CLO4**
6	Cache Memory Design	Lecture, Problem Solving	Quiz, Problem Sets	**CLO4**
7	Virtual Memory	Lecture, Problem Solving	Quiz, Problem Sets	**CLO4**
8	Input/Output Systems	Lecture, Discussions	Quiz, Assignment	**CLO4**
9	**Interrupt Driven I/O and Interfacing**	**Lecture, Guest Lectures**	**Quiz, Presentation**	**CLO5**

10	**Multiprocessor and Multicomputer**	**Lecture, Case Studies**	**Quiz, Case Study Reports**	**CLO5**
11	Parallel Processing	Lecture, Hands-on Labs	Quiz, Lab Reports	**CLO3**
12	**Processor Organization and Classification**	**Lecture, Simulation Exercises**	**Quiz, Simulation Reports**	**CLO3**

13	**Performance Evaluation**	**Lecture, Problem Solving**	**Quiz, Problem Sets**	**CLO4**
14	Operating System Support-1	Lecture, Discussions	Quiz, Assignment	**CLO5**
15	Operating System Support-2	Lecture, Research Papers	Quiz, Research Paper	**CLO5**
16	Micro Programmed Controlled Control Unit Organization	Lecture, Research Papers	Quiz, Research Paper	**CLO4, CLO5**
17	**Arithmetic & Logic Unit**	**Review Sessions, Practice Tests**	**Final Exam**	**CLO4, CLO5**

Week 1

Introduction to Architecture & Organization

- Architecture is those attributes visible to the programmer
  - Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques.
  - e.g. Is there a multiply instruction?
- Organization is how features are implemented
  - Control signals, interfaces, memory technology.
  - e.g. Is there a hardware multiply unit or is it done by repeated addition?

ARCHITECTURE & ORGANIZATION 2

All Intel x86 family share the same basic architecture

The IBM System/370 family share the same basic architecture

This gives code compatibility

- At least backwards

Organization differs between different versions

Structure & Function

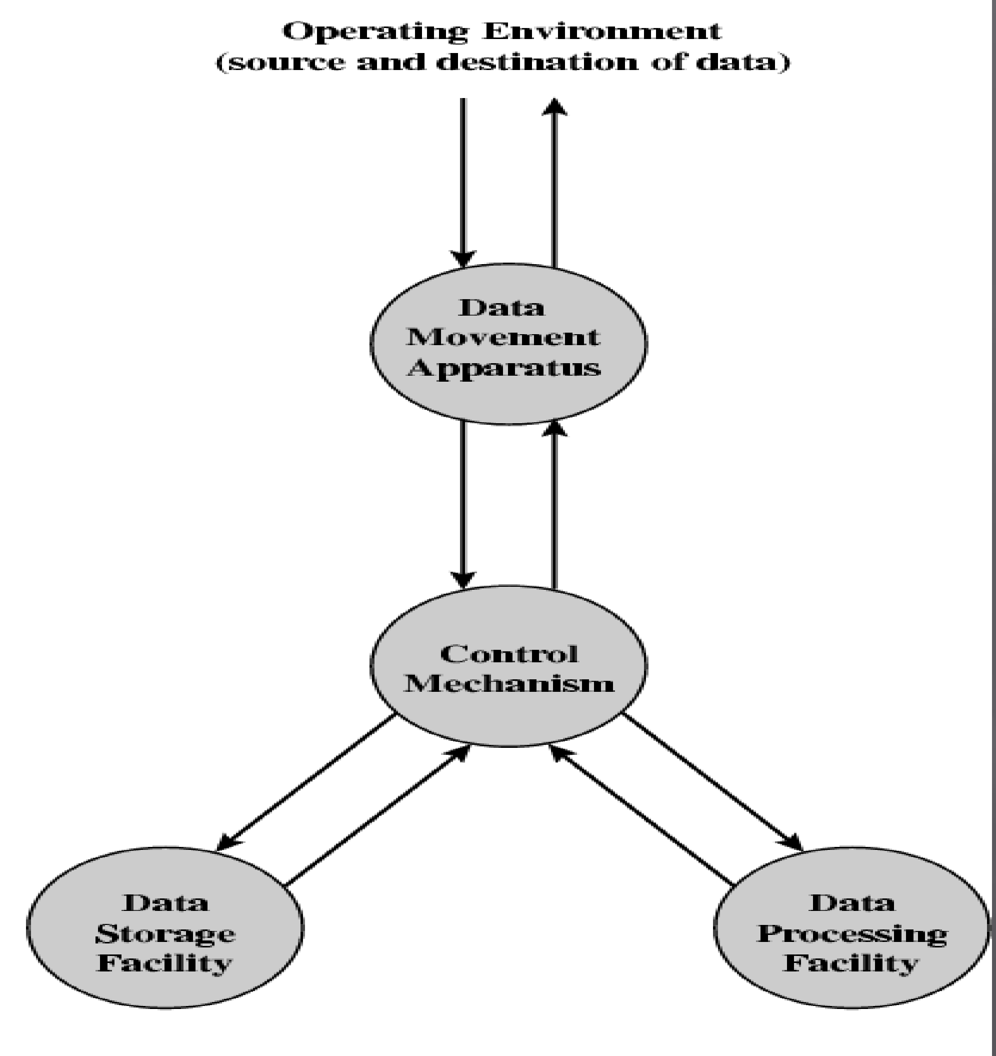
- Structure is the way in which components relate to each other
- Function is the operation of individual components as part of the structure

FUNCTION

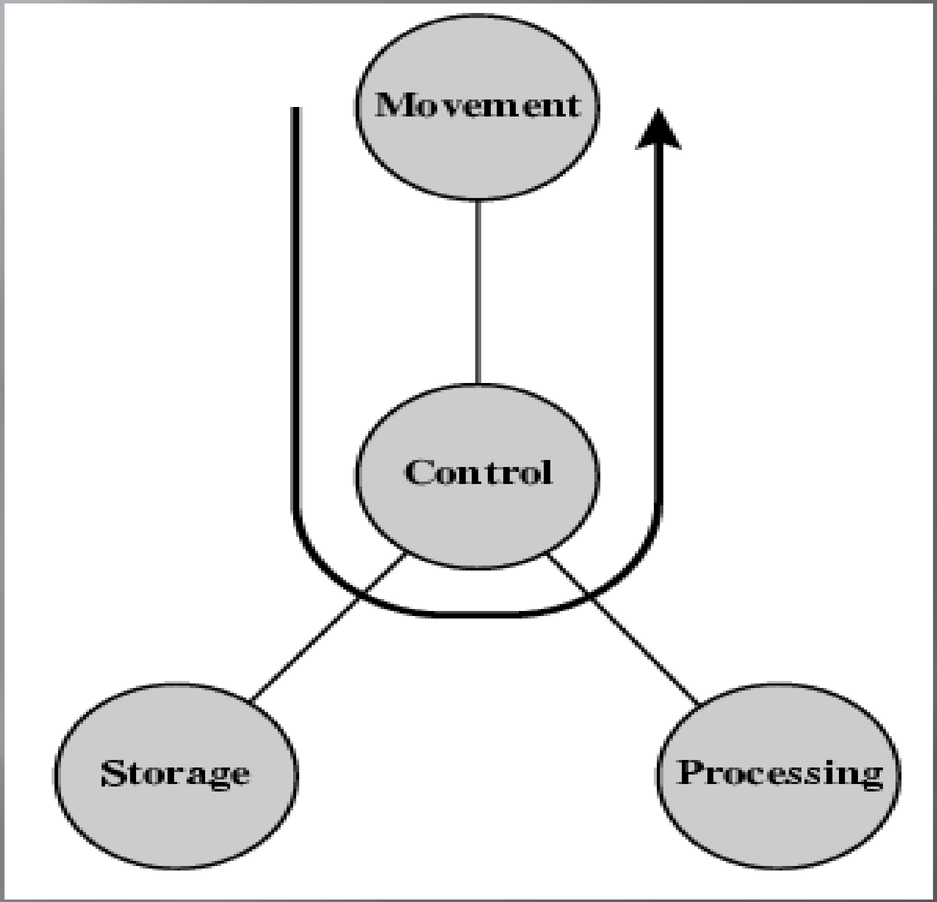
All computer functions are:

- Data processing
- Data storage
- Data movement
- Control

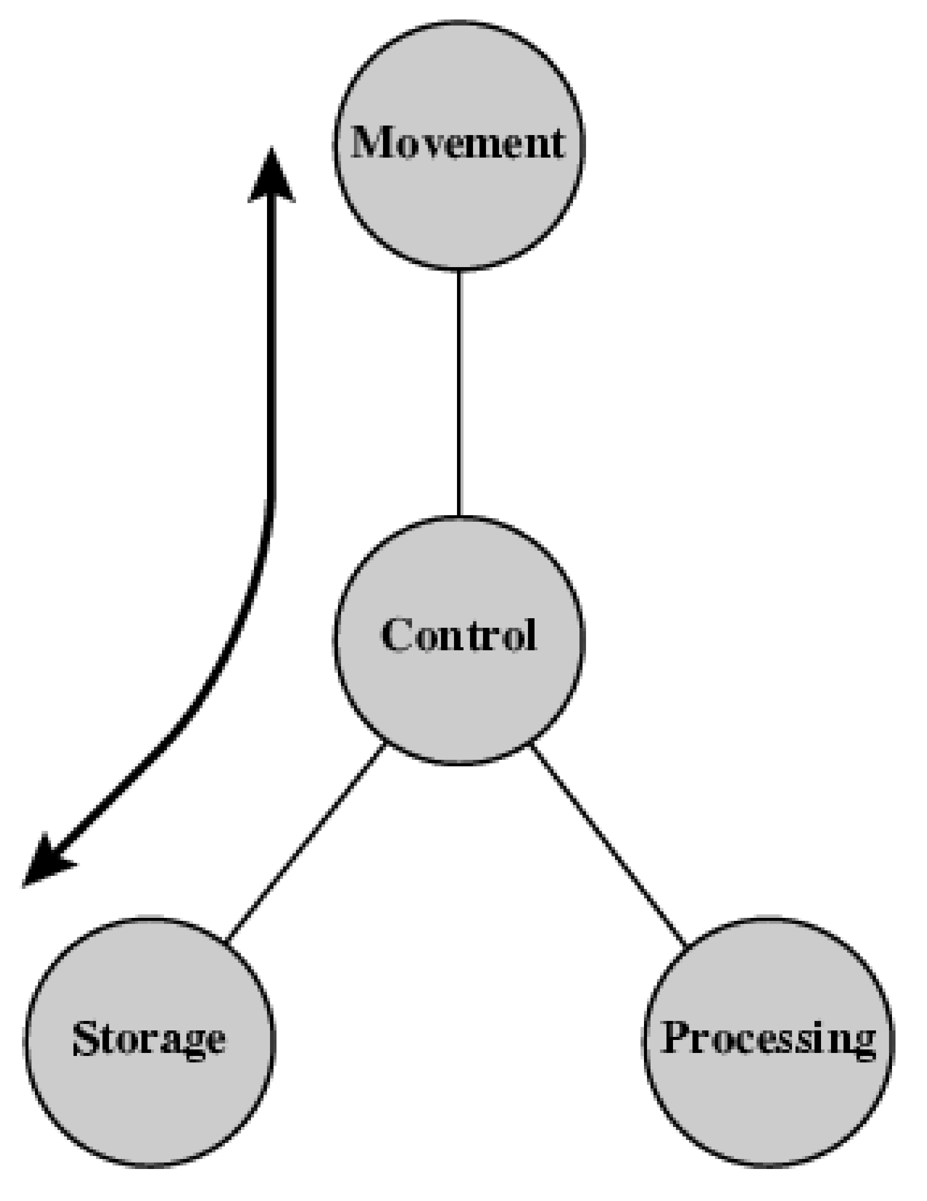
Functional View



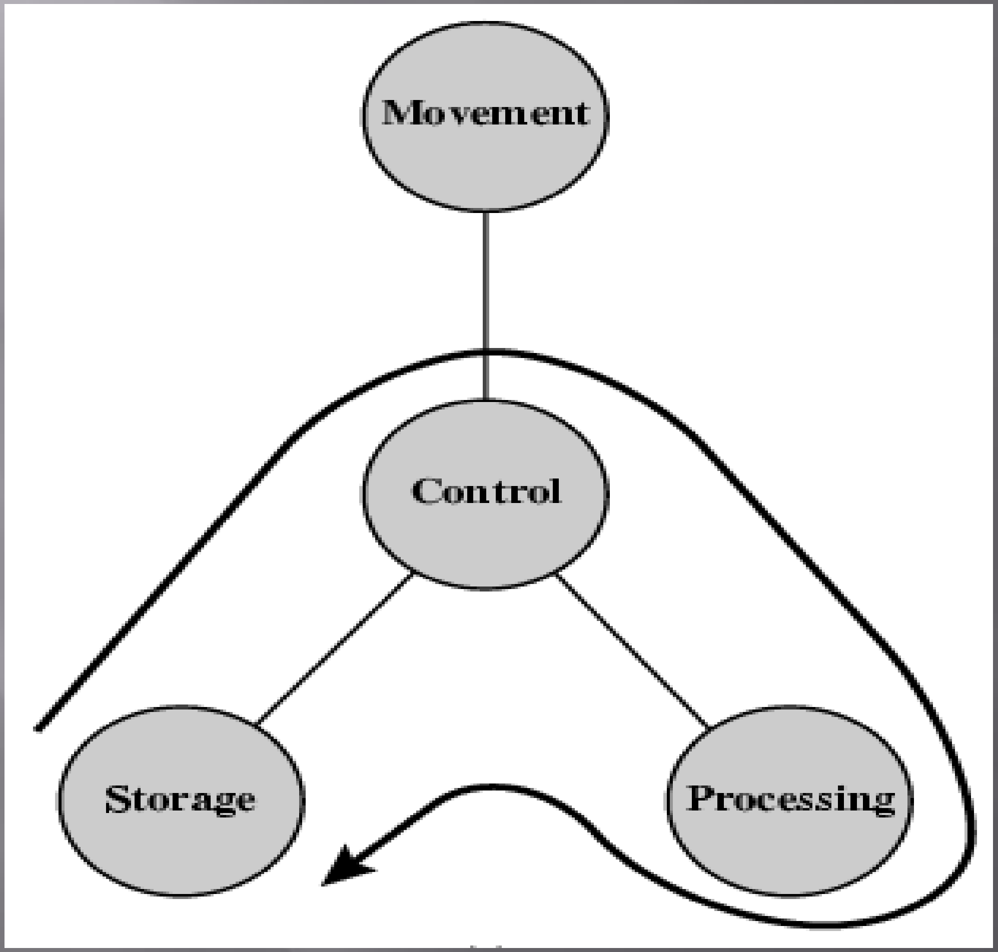
Operations (a) Data movement



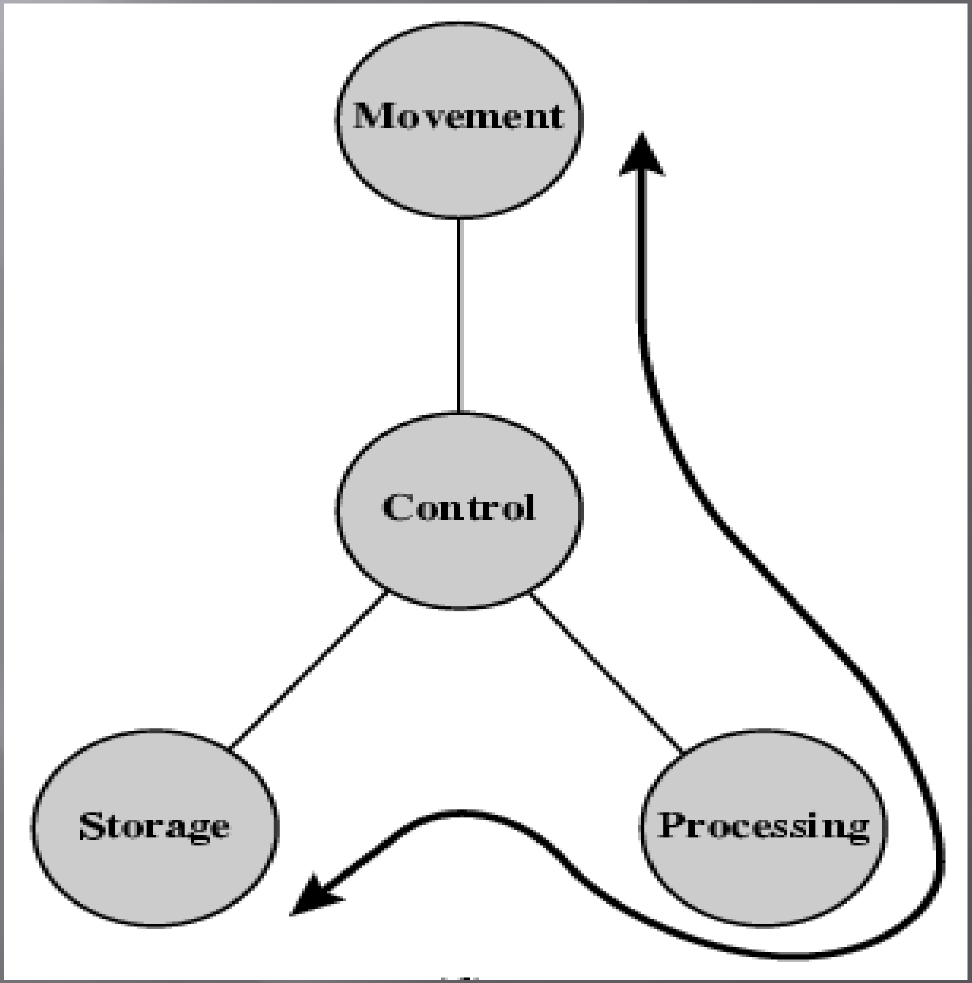
Operations (b) Storage



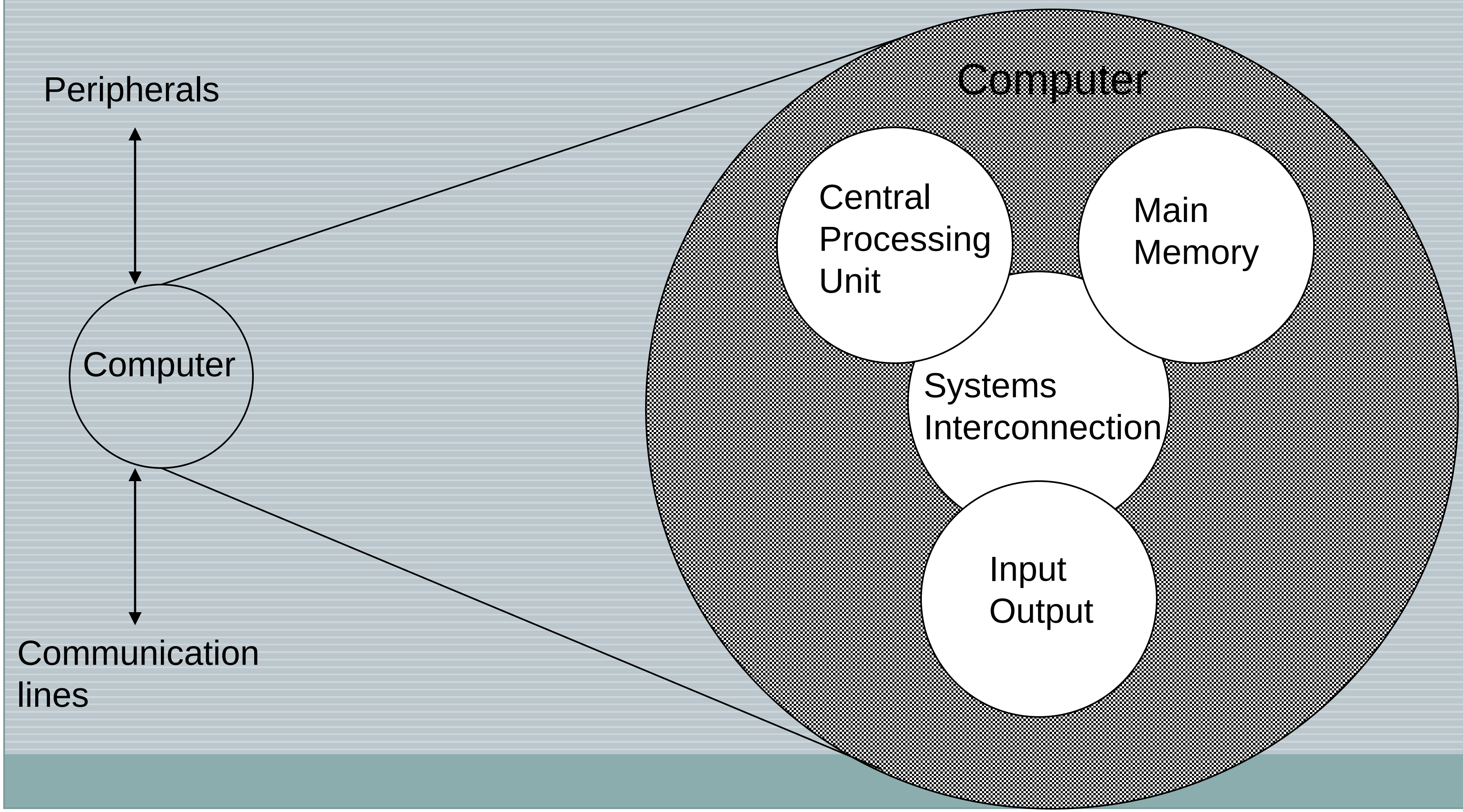
Operation (c) Processing from/to storage



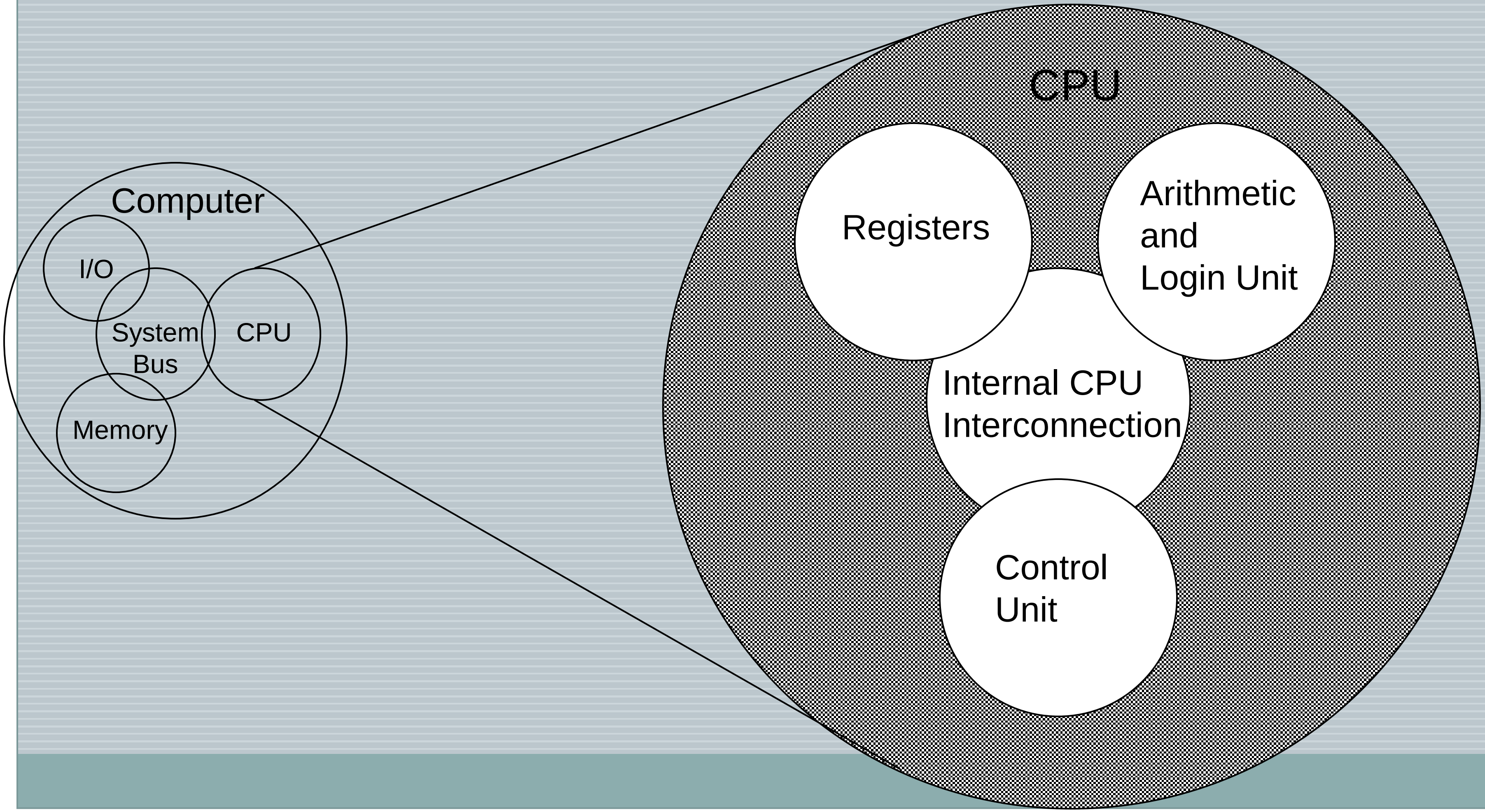
Operation (d)
Processing from storage to I/O



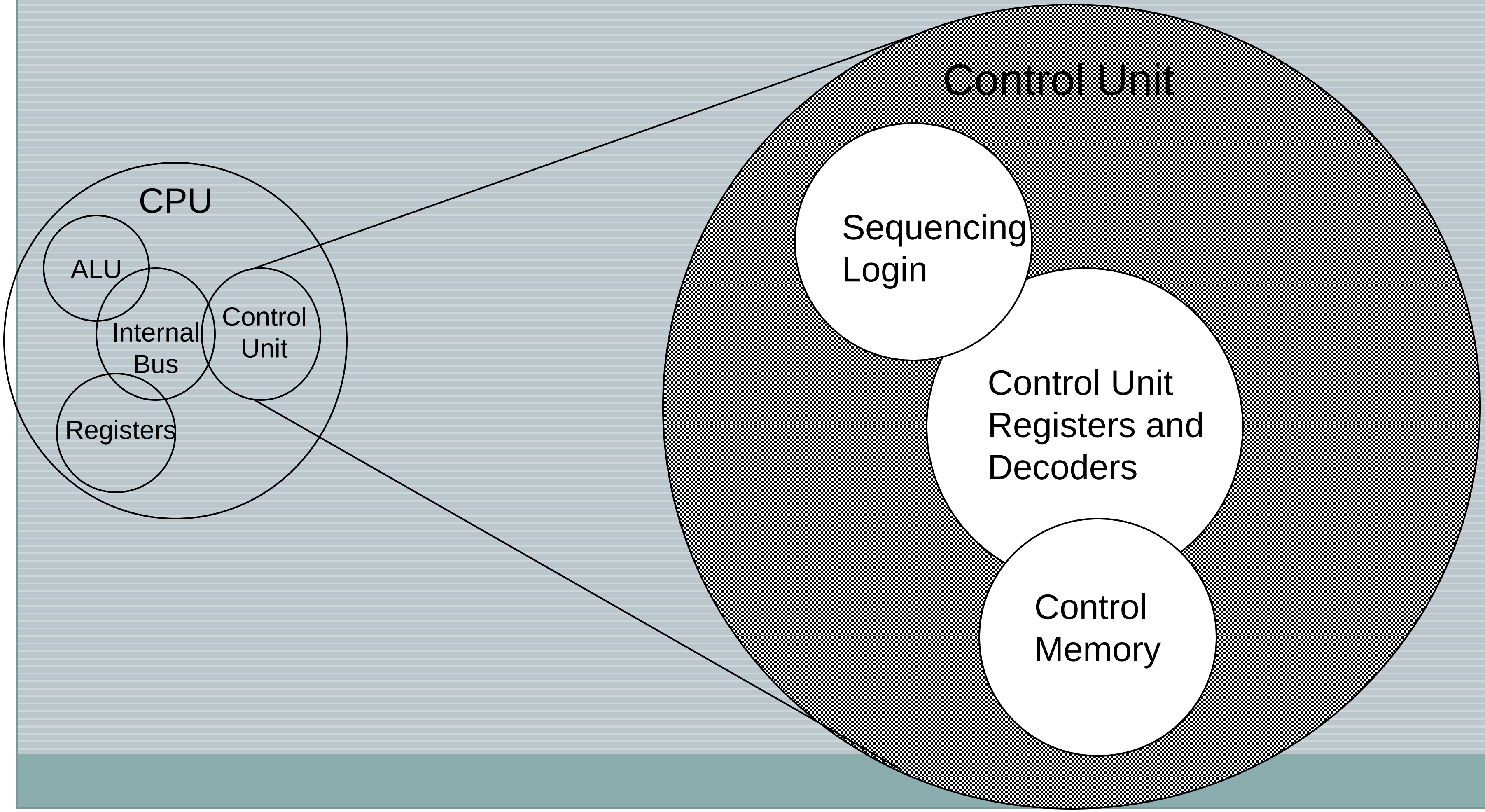
Structure - Top Level



Structure - The CPU



Structure - The Control Unit



Outline of the Book (1)

- Computer Evolution and Performance
- Computer Interconnection Structures
- Internal Memory
- External Memory
- Input/Output
- Operating Systems Support
- Computer Arithmetic
- Instruction Sets

Outline of the Book (2)

- CPU Structure and Function
- Reduced Instruction Set Computers
- Superscalar Processors
- Control Unit Operation
- Microprogrammed Control
- Multiprocessors and Vector Processing
- Digital Logic (Appendix)

Internet Resources
- Web site for book

- http://WilliamStallings.com/COA/COA7e.html
  - links to sites of interest
  - links to sites for courses that use the book
  - errata list for book
  - information on other books by W. Stallings
- http://WilliamStallings.com/StudentSupport.html
  - Math
  - How-to
  - Research resources
  - Misc

Internet Resources
- Web sites to look for

- WWW Computer Architecture Home Page
- CPU Info Center
- Processor Emporium
- ACM Special Interest Group on Computer Architecture
- IEEE Technical Committee on Computer Architecture
- Intel Technology Journal
- Manufacturer’s sites
  - Intel, IBM, etc.

Internet Resources
- Usenet News Groups

- comp.arch
- comp.arch.arithmetic
- comp.arch.storage
- comp.parallel

Week 2

VON NEUMANN ARCHITECTURES
ENIAC - BACKGROUND

Electronic Numerical Integrator And Computer

Eckert and Mauchly

University of Pennsylvania

Trajectory tables for weapons

Started 1943

Finished 1946

- Too late for war effort

Used until 1955

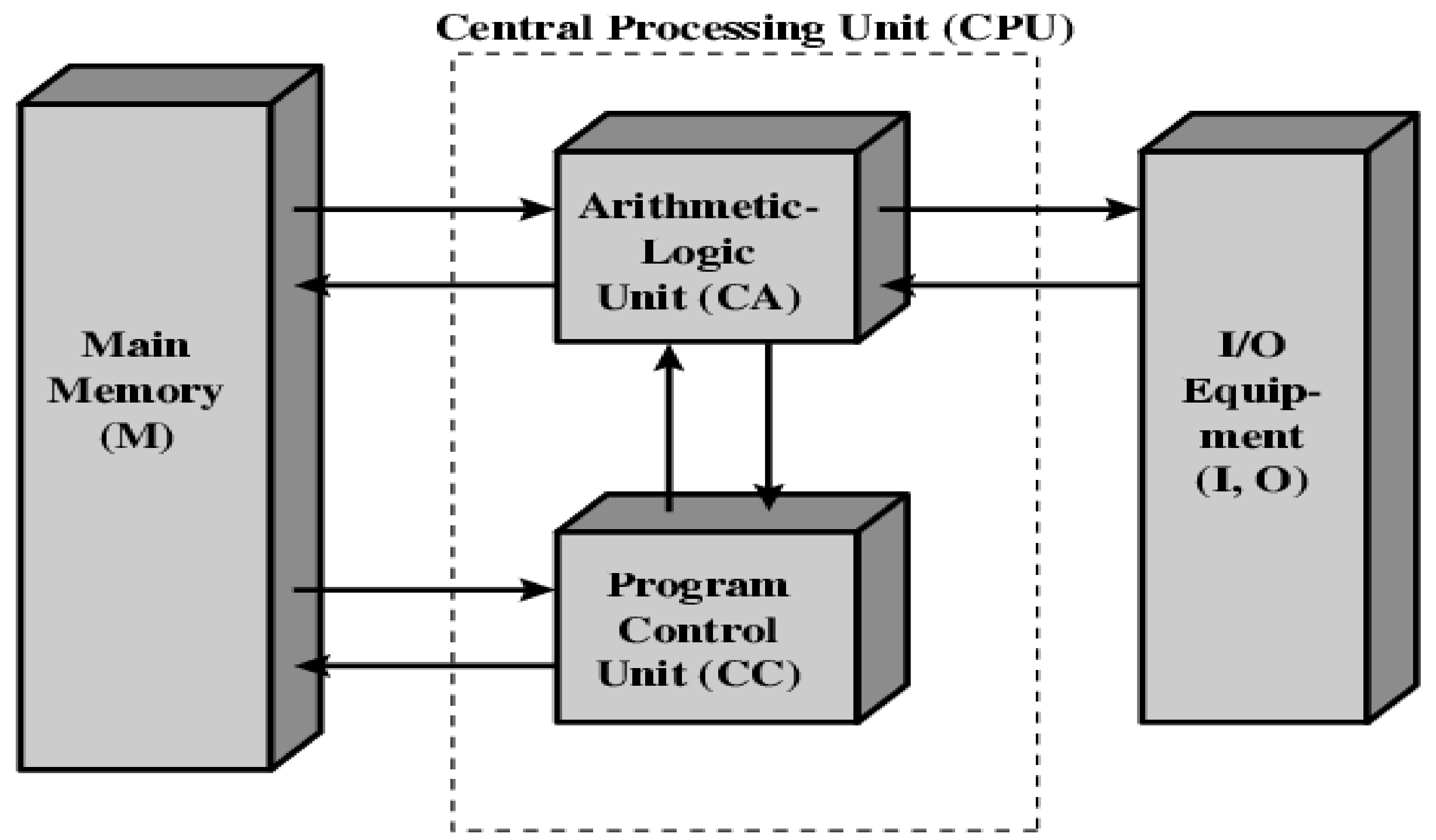
ENIAC - details

- Decimal (not binary)
- 20 accumulators of 10 digits
- Programmed manually by switches
- 18,000 vacuum tubes
- 30 tons
- 15,000 square feet
- 140 kW power consumption
- 5,000 additions per second

von Neumann/Turing

- Stored Program concept
- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- Princeton Institute for Advanced Studies
  - IAS
- Completed 1952

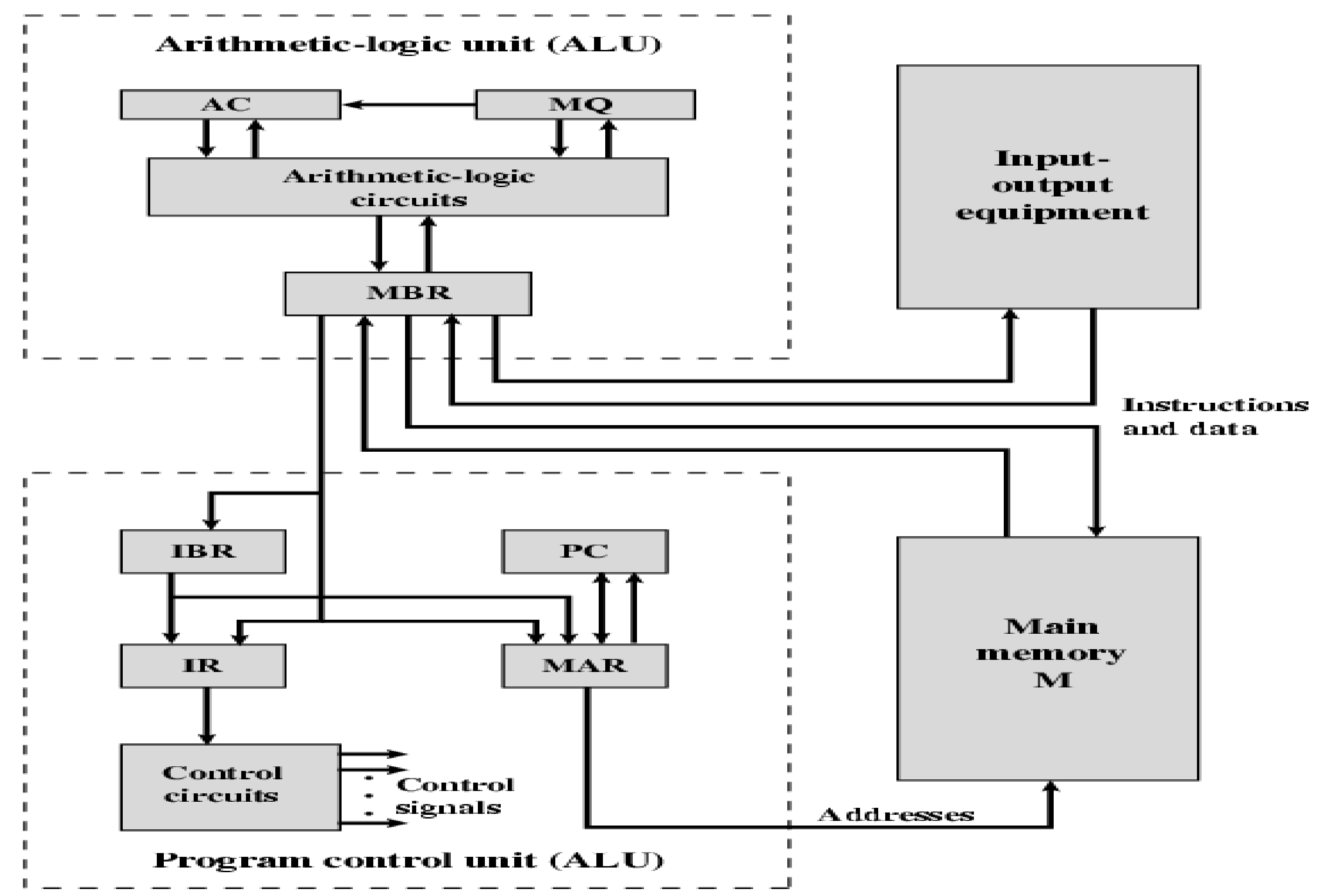
STRUCTURE OF VON NEUMANN MACHINE


Central Processing Unit (CPU)
Main Memory (M)
Arithmetic-Logic Unit (CA)
Program Control Unit (CC)
I/O Equip-ment (I, O)

IAS - details

- 1000 x 40 bit words
  - Binary number
  - 2 x 20 bit instructions
- Set of registers (storage in CPU)
  - Memory Buffer Register
  - Memory Address Register
  - Instruction Register
  - Instruction Buffer Register
  - Program Counter
  - Accumulator
  - Multiplier Quotient

Structure of IAS – detail



Commercial Computers

- 1947 - Eckert-Mauchly Computer Corporation
- UNIVAC I (Universal Automatic Computer)
- US Bureau of Census 1950 calculations
- Became part of Sperry-Rand Corporation
- Late 1950s - UNIVAC II
  - Faster
  - More memory

IBM

- Punched-card processing equipment
- 1953 - the 701
  - IBM's first stored program computer
  - Scientific calculations
- 1955 - the 702
  - Business applications
- Lead to 700/7000 series

Transistors

- Replaced vacuum tubes
- Smaller
- Cheaper
- Less heat dissipation
- Solid State device
- Made from Silicon (Sand)
- Invented 1947 at Bell Labs
- William Shockley et al.

Transistor Based Computers

- Second generation machines
- NCR & RCA produced small transistor machines
- IBM 7000
- DEC - 1957
  - Produced PDP-1

Microelectronics

- Literally - “small electronics”
- A computer is made up of gates, memory cells and interconnections
- These can be manufactured on a semiconductor
- e.g. silicon wafer

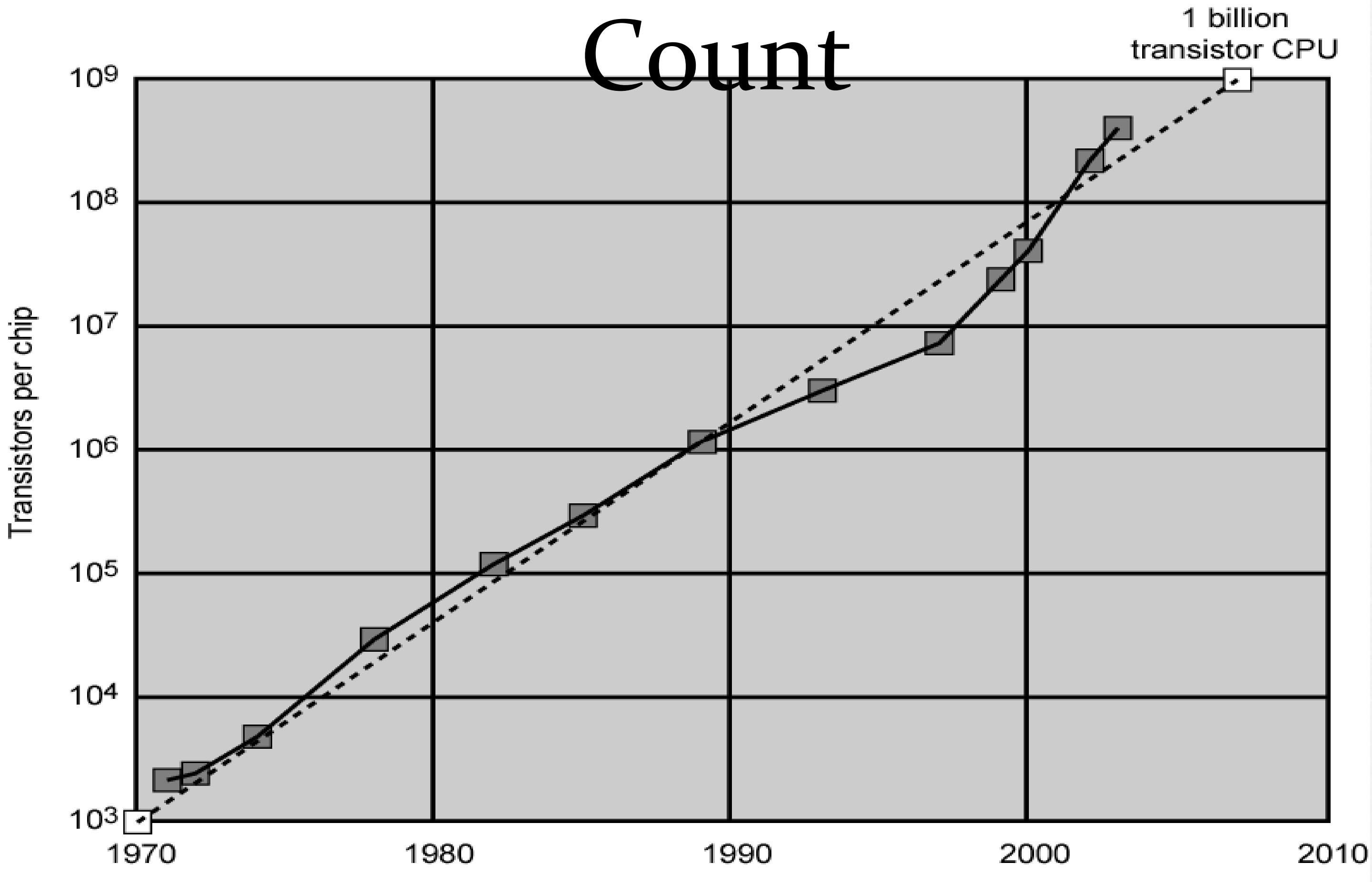
Generations of Computer

- Vacuum tube - 1946-1957
- Transistor - 1958-1964
- Small scale integration - 1965 on
  - Up to 100 devices on a chip
- Medium scale integration - to 1971
  - 100-3,000 devices on a chip
- Large scale integration - 1971-1977
  - 3,000 - 100,000 devices on a chip
- Very large scale integration - 1978 -1991
  - 100,000 - 100,000,000 devices on a chip
- Ultra large scale integration – 1991 -
  - Over 100,000,000 devices on a chip

Moore’s Law

- Increased density of components on chip
- Gordon Moore – co-founder of Intel
- Number of transistors on a chip will double every year
- Since 1970’s development has slowed a little
  - Number of transistors doubles every 18 months
- Cost of a chip has remained almost unchanged
- Higher packing density means shorter electrical paths, giving higher performance
- Smaller size gives increased flexibility
- Reduced power and cooling requirements
- Fewer interconnections increases reliability

Growth in CPU Transistor Count



IBM 360 series

- 1964
- Replaced (& not compatible with) 7000 series
- First planned "family" of computers
  - Similar or identical instruction sets
  - Similar or identical O/S
  - Increasing speed
  - Increasing number of I/O ports (i.e. more terminals)
  - Increased memory size
  - Increased cost
- Multiplexed switch structure

Speeding it up

- Pipelining
- On board cache
- On board L1 & L2 cache
- Branch prediction
- Data flow analysis
- Speculative execution

Week 3

Instruction Set

- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes

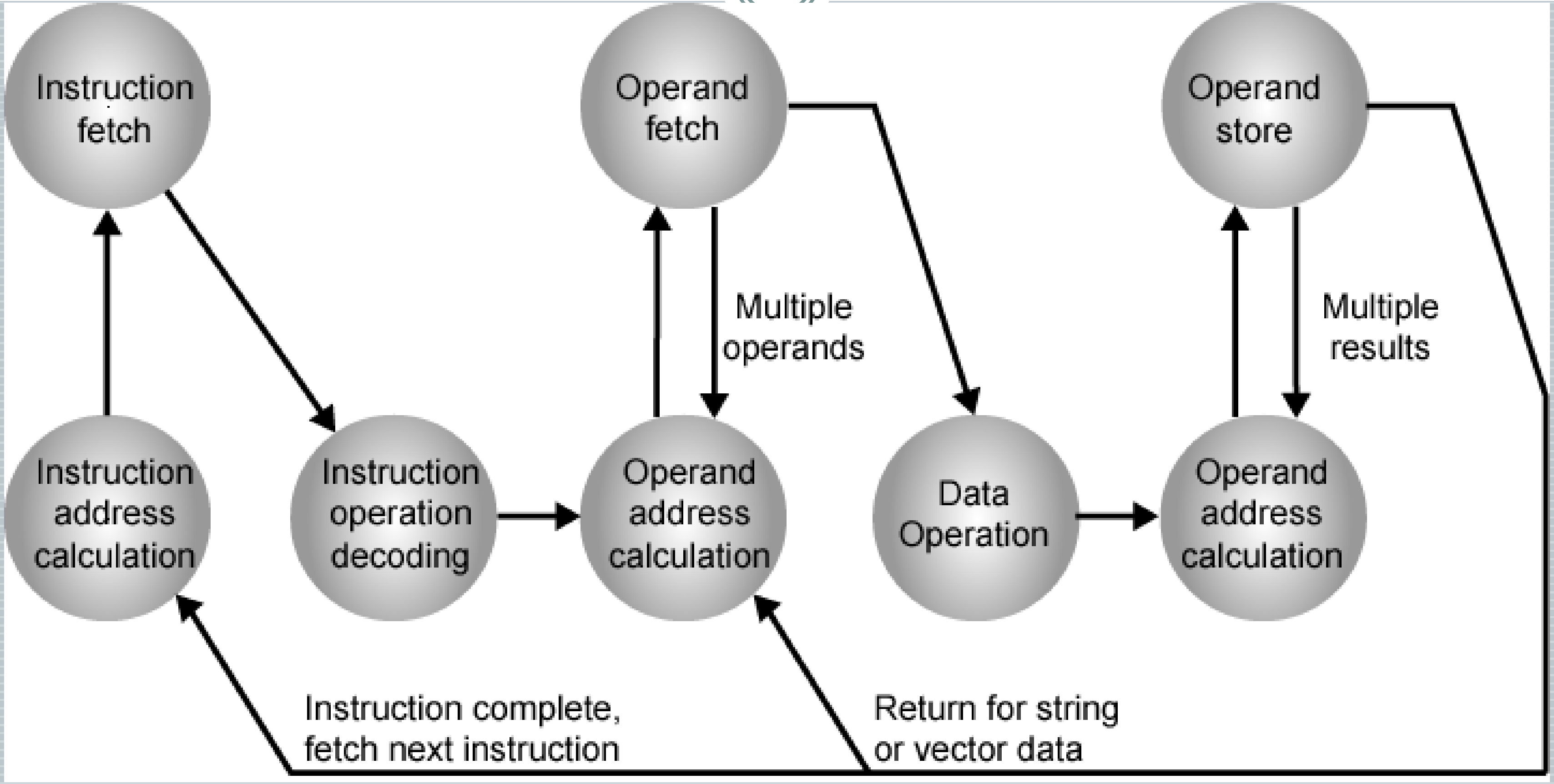
Elements of an Instruction

- Operation code (Op code)
  - Do this
- Source Operand reference
  - To this
- Result Operand reference
  - Put the answer here
- Next Instruction Reference
  - When you have done that, do this...

Where have all the Operands Gone?

- Long time passing….
- (If you don’t understand, you’re too young!)
- Main memory (or virtual memory or cache)
- CPU register
- I/O device

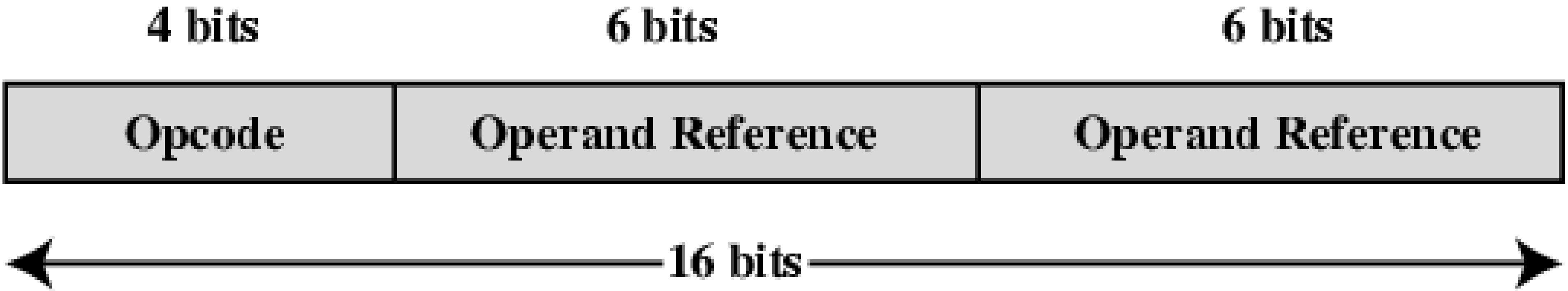
Instruction Cycle State Diagram


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

Instruction Representation

- In machine code each instruction has a unique bit pattern
- For human consumption (well, programmers anyway) a symbolic representation is used
  - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
  - ADD A,B

Simple Instruction Format



4 bits	6 bits	6 bits
Opcode	Operand Reference	Operand Reference

16 bits

Instruction Types

- Data processing
- Data storage (main memory)
- Data movement (I/O)
- Program flow control

Number of Addresses (a)

- 3 addresses
  - Operand 1, Operand 2, Result
  - a = b + c;
  - May be a forth - next instruction (usually implicit)
  - Not common
  - Needs very long words to hold everything

Number of Addresses (b)

- 2 addresses
  - One address doubles as operand and result
  - a = a + b
  - Reduces length of instruction
  - Requires some extra work
    - Temporary storage to hold some results

Number of Addresses (c)

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines

Number of Addresses (d)

- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push a
  -         push b
  -         add
  -         pop c
  - c = a + b

How Many Addresses

- More addresses
  - More complex (powerful?) instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster fetch/execution of instructions

Design Decisions (1)

- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses

Design Decisions (2)

- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes (later…)

- RISC v CISC

Types of Operand

- Addresses
- Numbers
    - Integer/floating point
- Characters
    - ASCII etc.
- Logical Data
    - Bits or flags
- (Aside: Is there any difference between numbers and characters? Ask a C programmer!)

Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

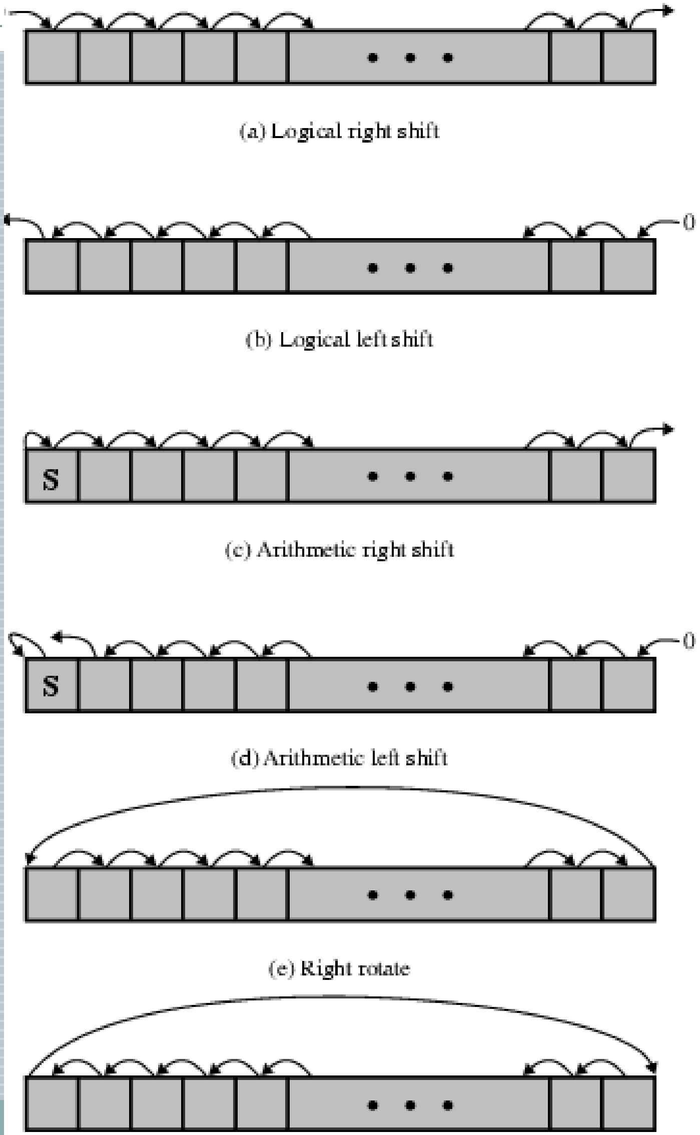
Data Transfer

- Specify
  - Source
  - Destination
  - Amount of data
- May be different instructions for different movements
  - e.g. IBM 370
- Or one instruction and different addresses
  - e.g. VAX

Arithmetic

- Add, Subtract, Multiply, Divide
- Signed Integer
- Floating point ?
- May include
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)

Shift and Rotate Operations



(a) Logical right shift

(b) Logical left shift

(c) Arithmetic right shift

(d) Arithmetic left shift

(e) Right rotate

(f) Left rotate

Logical

- Bitwise operations
- AND, OR, NOT

Conversion

- E.g. Binary to Decimal

Input/Output

- May be specific instructions
- May be done using data movement instructions (memory mapped)
- May be done by a separate controller (DMA)

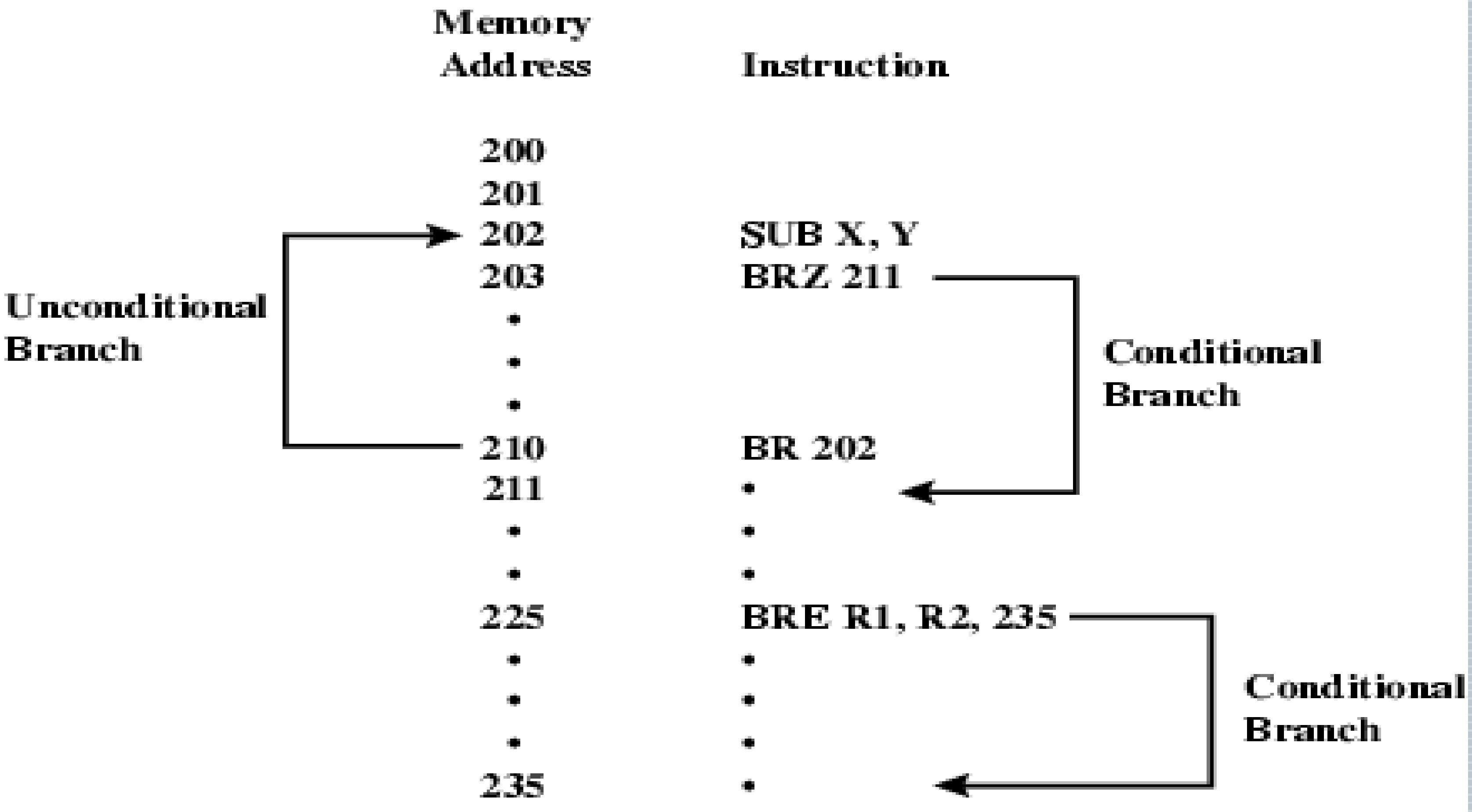
Systems Control

- Privileged instructions
- CPU needs to be in specific state
  - Ring 0 on 80386+
  - Kernel mode
- For operating systems use

Transfer of Control

- Branch
  - e.g. branch to x if result is zero
- Skip
  - e.g. increment and skip if zero
  - ISZ Register1
  - Branch xxxx
  - ADD A
- Subroutine call
  - c.f. interrupt call

Branch Instruction



Memory Address	Instruction
200	
201	
202	SUB X, Y
203	BRZ 211
•	
•	
•	
210	BR 202
211	•
•	•
•	•
225	BRE R1, R2, 235
•	•
•	•
•	•
235	•

Unconditional Branch

Conditional Branch

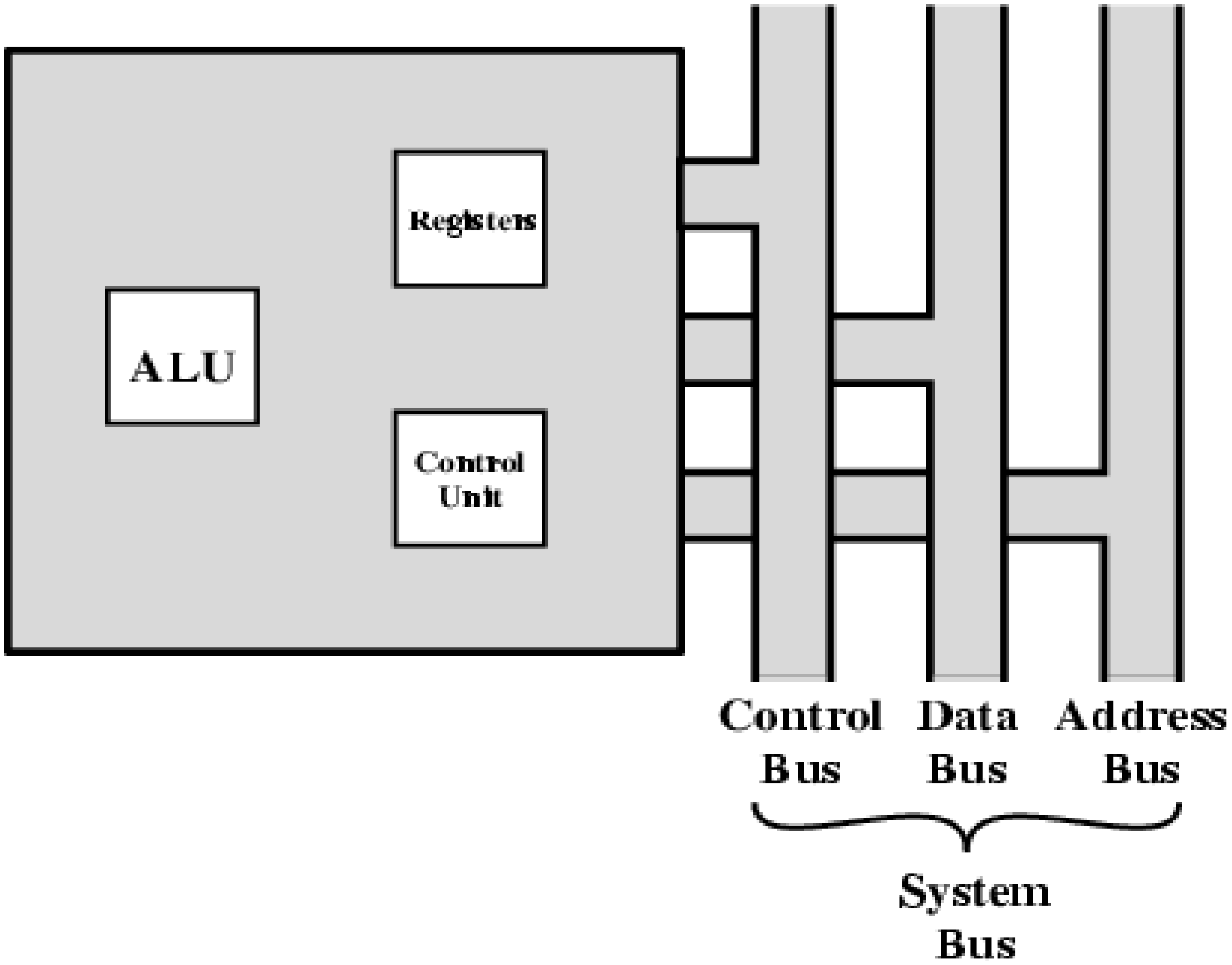
Conditional Branch

Week 4

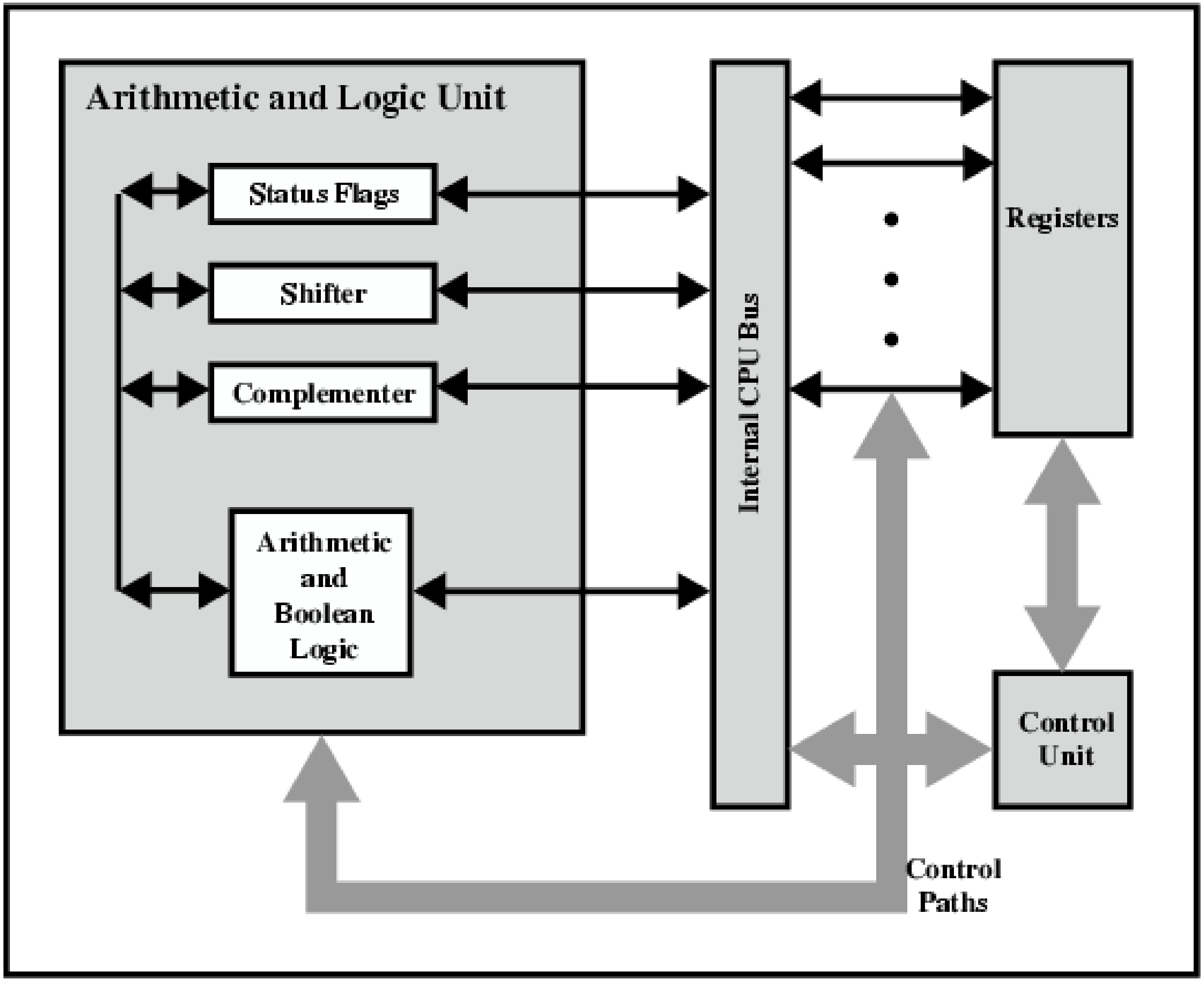
CPU Structure

- CPU must:
  - Fetch instructions
  - Interpret instructions
  - Fetch data
  - Process data
  - Write data

CPU With Systems Bus



CPU Internal Structure



Registers

- CPU must have some working space (temporary storage)
- Called registers
- Number and function vary between processor designs
- One of the major design decisions
- Top level of memory hierarchy

User Visible Registers

- General Purpose
- Data
- Address
- Condition Codes

General Purpose Registers (1)

- May be true general purpose
- May be restricted
- May be used for data or addressing
- Data
  - Accumulator
- Addressing
  - Segment

General Purpose Registers (2)

- Make them general purpose
  - Increase flexibility and programmer options
  - Increase instruction size & complexity
- Make them specialized
  - Smaller (faster) instructions
  - Less flexibility

How Many GP Registers?

- Between 8 - 32
- Fewer = more memory references
- More does not reduce memory references and takes up processor real estate
- See also RISC

How big?

- Large enough to hold full address
- Large enough to hold full word
- Often possible to combine two data registers
  - C programming
  - double int a;
  - long int a;

Condition Code Registers

- Sets of individual bits
  - e.g. result of last operation was zero
- Can be read (implicitly) by programs
  - e.g. Jump if zero
- Can not (usually) be set by programs

Control & Status Registers

- Program Counter
- Instruction Decoding Register
- Memory Address Register
- Memory Buffer Register

- Revision: what do these all do?

Program Status Word

- A set of bits
- Includes Condition Codes
- Sign of last result
- Zero
- Carry
- Equal
- Overflow
- Interrupt enable/disable
- Supervisor

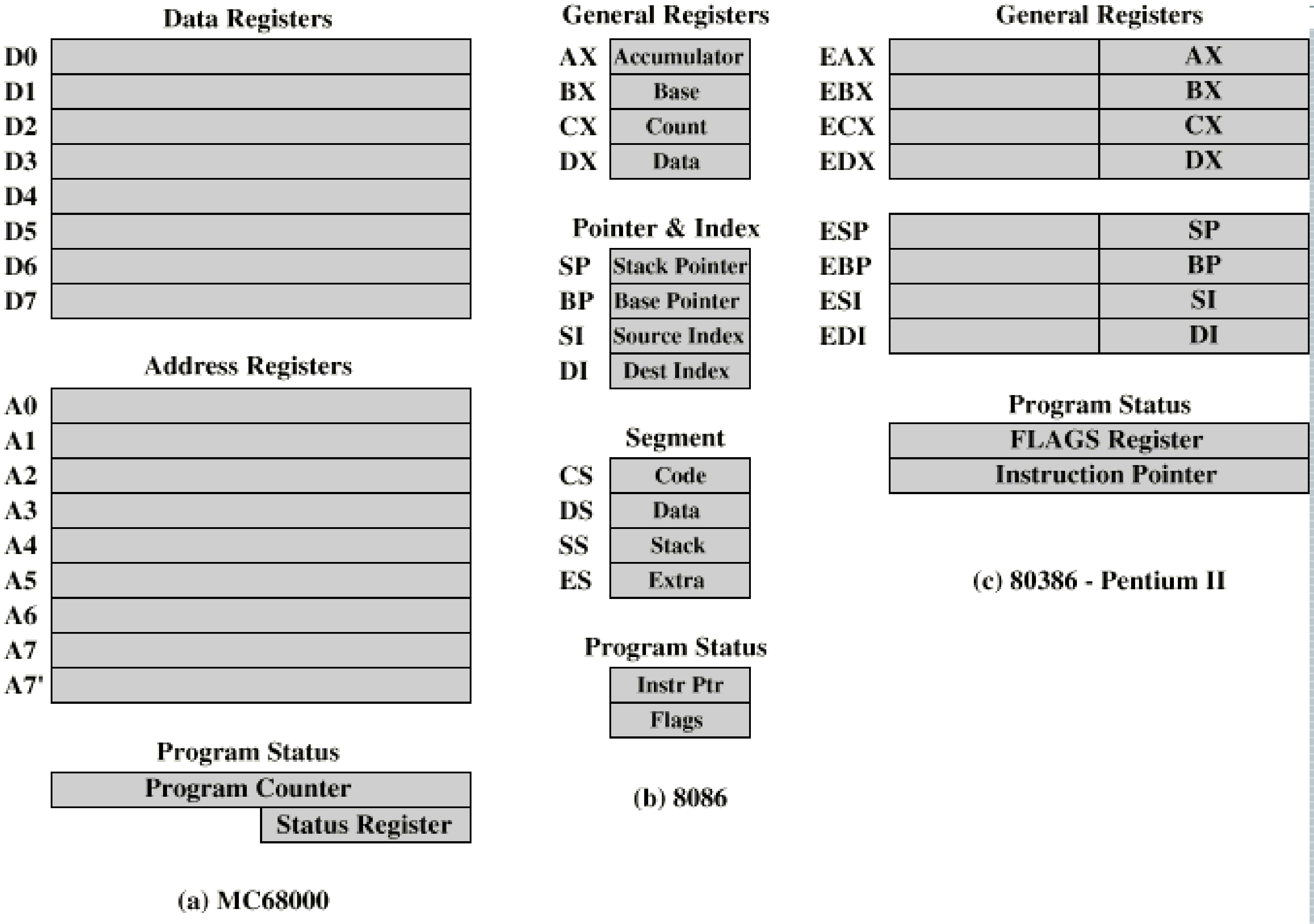
Supervisor Mode

- Intel ring zero
- Kernel mode
- Allows privileged instructions to execute
- Used by operating system
- Not available to user programs

Other Registers

- May have registers pointing to:
  - Process control blocks (see O/S)
  - Interrupt Vectors (see O/S)
- N.B. CPU design and operating system design are closely linked

Example Register Organizations



Data Registers

D0
D1
D2
D3
D4
D5
D6
D7

Address Registers

A0
A1
A2
A3
A4
A5
A6
A7
A7'

Program Status

Program Counter

Status Register

(a) MC68000

General Registers

AX	Accumulator
BX	Base
CX	Count
DX	Data

Pointer & Index

SP	Stack Pointer
BP	Base Pointer
SI	Source Index
DI	Dest Index

Segment

CS	Code
DS	Data
SS	Stack
ES	Extra

Program Status

Instr Ptr

Flags

(b) 8086

General Registers

EAX		AX
EBX		BX
ECX		CX
EDX		DX

ESP		SP
EBP		BP
ESI		SI
EDI		DI

Program Status

FLAGS Register

Instruction Pointer

(c) 80386 - Pentium II

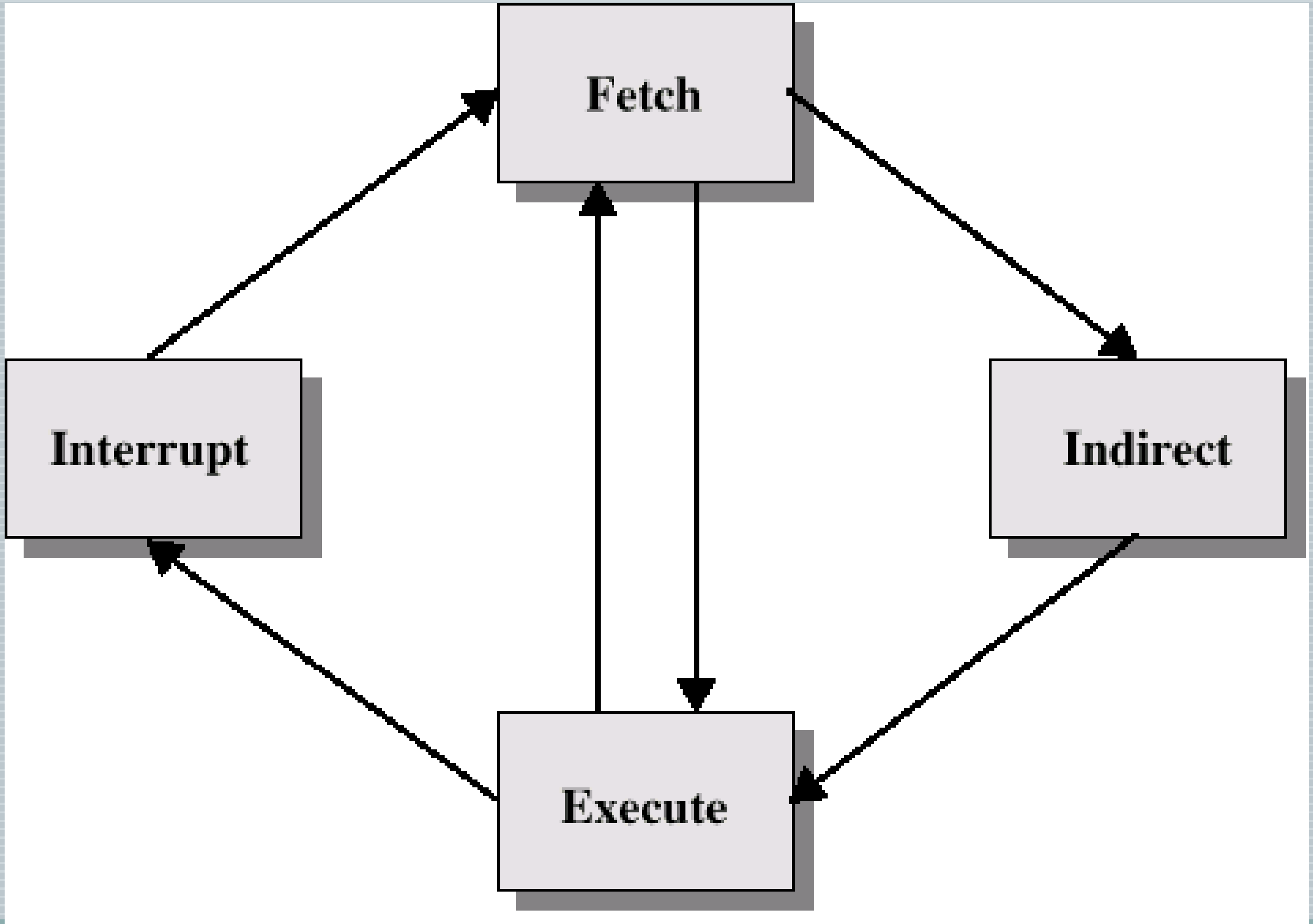
Instruction Cycle

- Revision
- Stallings Chapter 3

Indirect Cycle

- May require memory access to fetch operands
- Indirect addressing requires more memory accesses
- Can be thought of as additional instruction subcycle

Instruction Cycle with Indirect



Data Flow (Instruction Fetch)

- Depends on CPU design
- In general:

- Fetch
  - PC contains address of next instruction
  - Address moved to MAR
  - Address placed on address bus
  - Control unit requests memory read
  - Result placed on data bus, copied to MBR, then to IR
  - Meanwhile PC incremented by 1

Data Flow (Data Fetch)

- IR is examined
- If indirect addressing, indirect cycle is performed
  - Right most N bits of MBR transferred to MAR
  - Control unit requests memory read
  - Result (address of operand) moved to MBR

Data Flow (Execute)

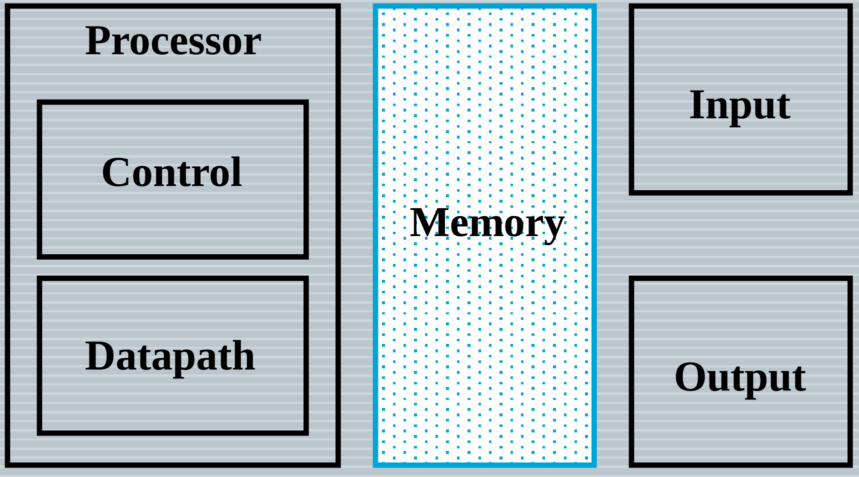
- May take many forms
- Depends on instruction being executed
- May include
    - Memory read/write
    - Input/Output
    - Register transfers
    - ALU operations

Week 5

Memory Hierarchy

- The Five Classic Components of a Computer
- Memory is usually implemented as:
  - Dynamic Random Access Memory (DRAM) - for main memory
  - Static Random Access Memory (SRAM) - for cache

The Big Picture: Where are We Now?



DRAM

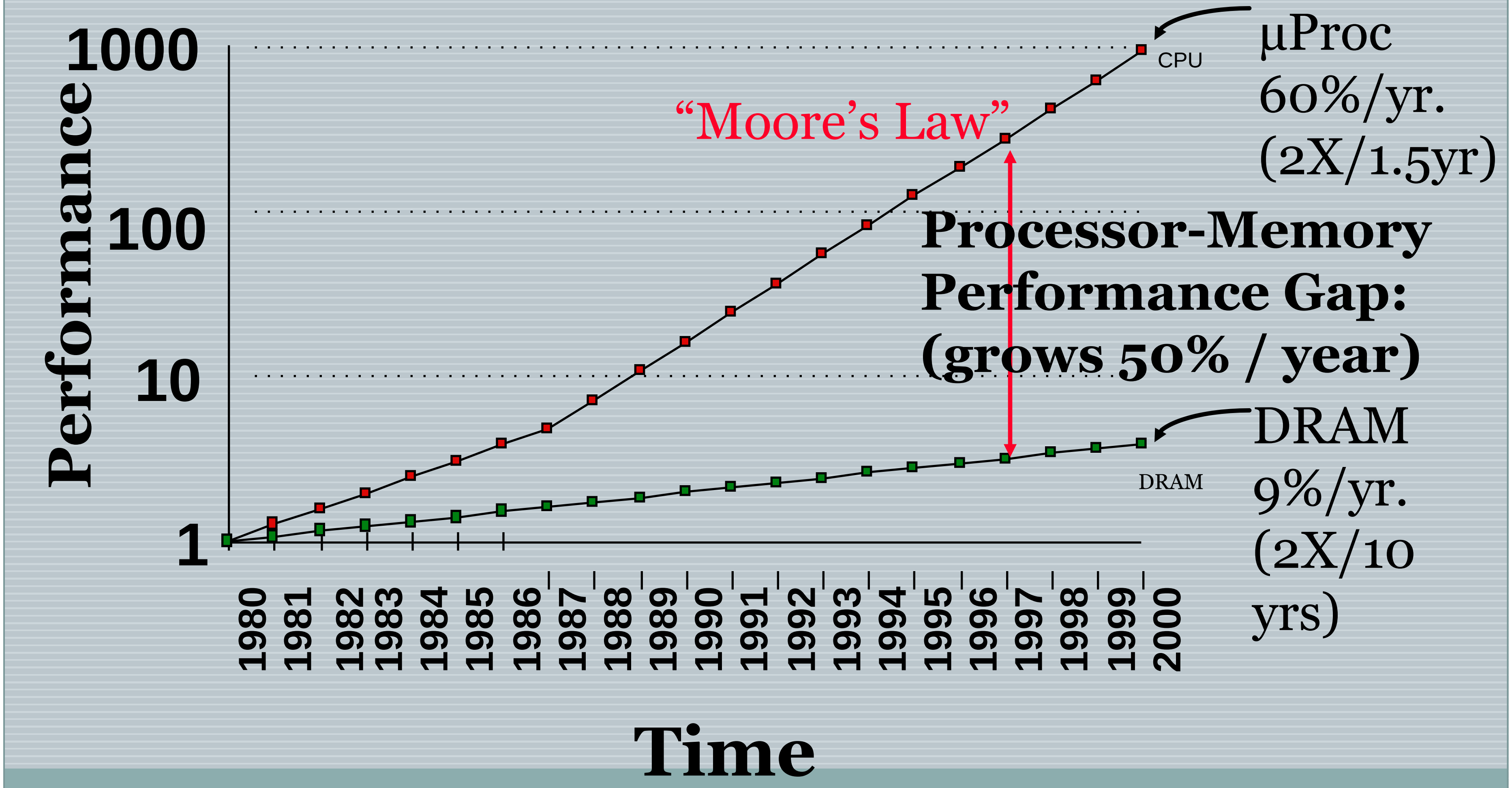
Year	Size	Cycle Time
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1995	64 Mb	120 ns
1998	256 Mb	100 ns
2001	1 Gb	80 ns

Technology Trends

	Capacity	Speed (latency)
Logic:	2x in 3 years	2x in 3 years
DRAM:	4x in 3 years	2x in 10 years
Disk:	4x in 3 years	2x in 10 years

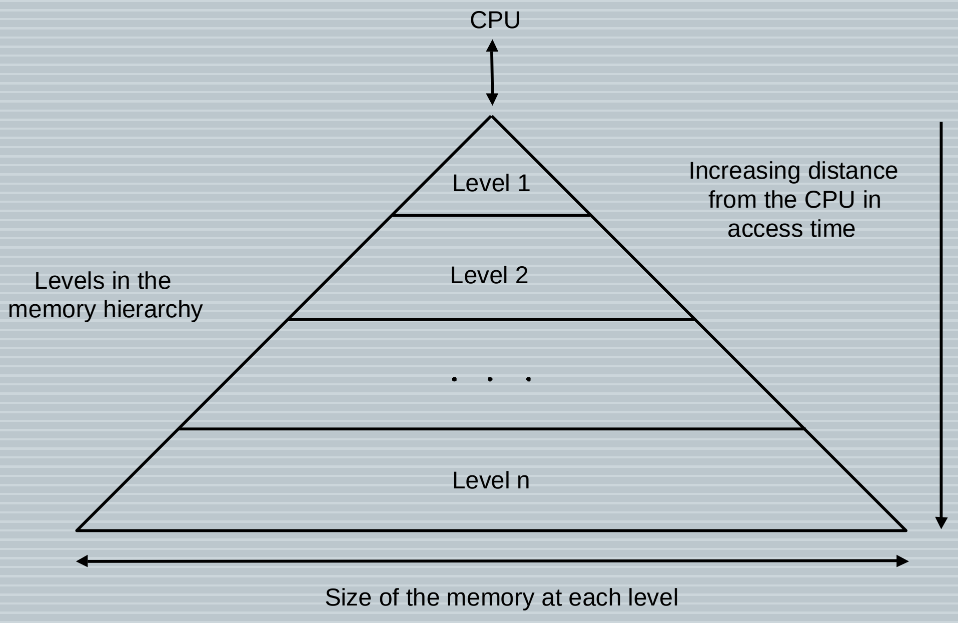
Who Cares About Memory?

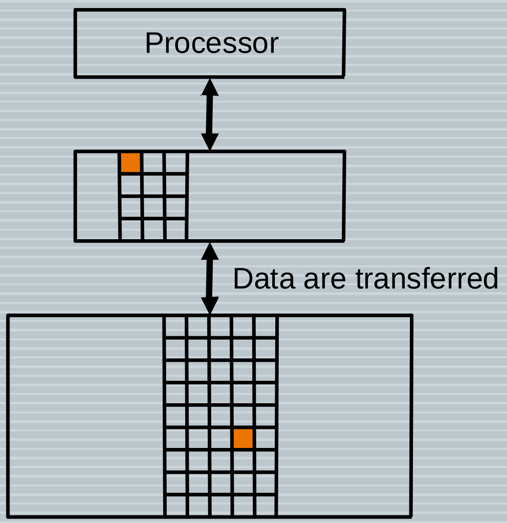
Processor-DRAM Memory Gap (latency)



Memory Hierarchy

Memory technology	Typical access time	$ per GB in 2004
SRAM	0.5–5 ns	$4000–$10,000
DRAM	50–70 ns	$100–$200
Magnetic disk	5,000,000–20,000,000 ns	$0.50–$2

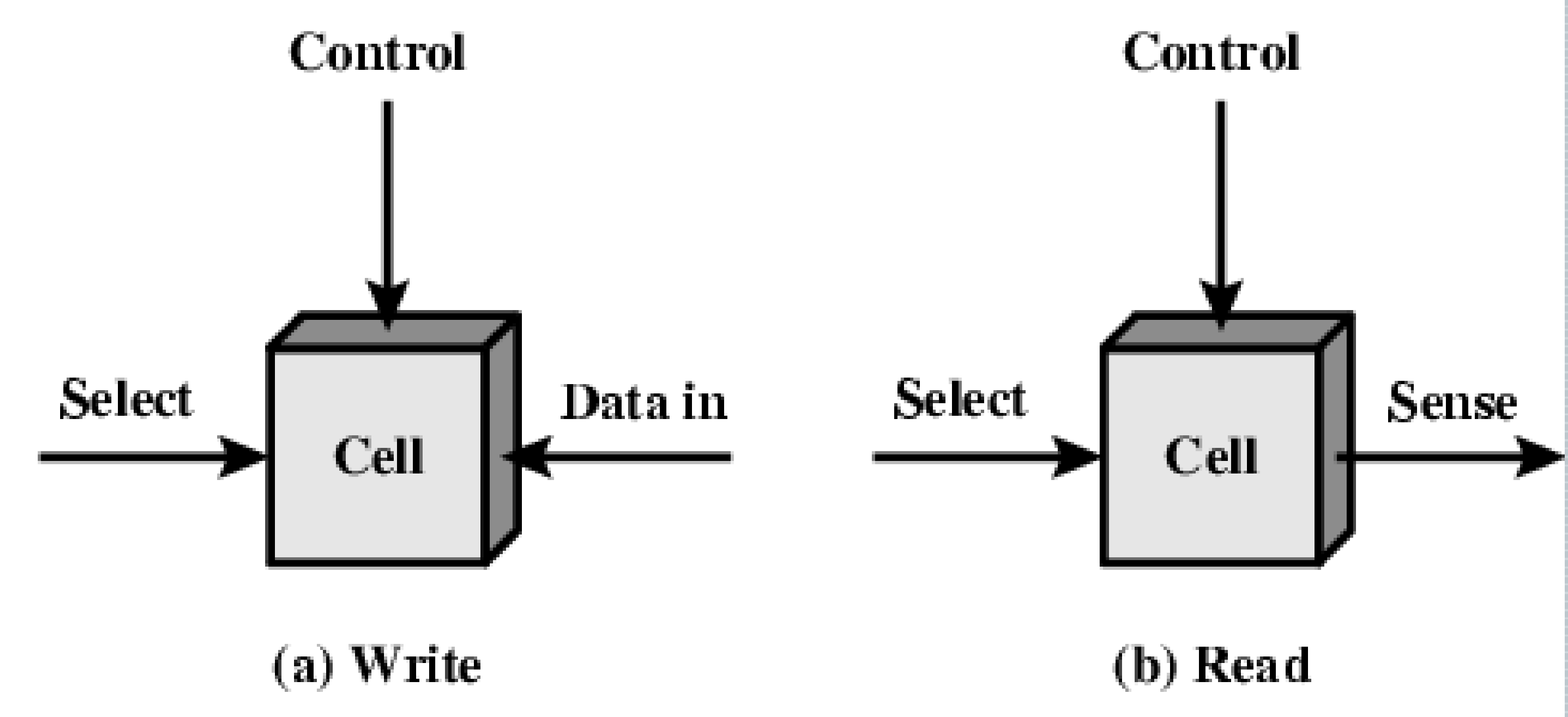

CPU
Level 1
Level 2
. . .
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level



Memory

- SRAM:
  - Value is stored on a pair of inverting gates
  - Very fast but takes up more space than DRAM (4 to 6 transistors)
- DRAM:
  - Value is stored as a charge on capacitor (must be refreshed)
  - Very small but slower than SRAM (factor of 5 to 10)

Memory Cell Operation



Dynamic RAM

- Bits stored as charge in capacitors
- Charges leak
- Need refreshing even when powered
- Simpler construction
- Smaller per bit
- Less expensive
- Need refresh circuits
- Slower
- Main memory
- Essentially analogue
  - Level of charge determines value

DRAM Operation

- Address line active when bit read or written
  - Transistor switch closed (current flows)
- Write
  - Voltage to bit line
    - High for 1 low for 0
  - Then signal address line
    - Transfers charge to capacitor
- Read
  - Address line selected
    - transistor turns on
  - Charge from capacitor fed via bit line to sense amplifier
    - Compares with reference value to determine 0 or 1
  - Capacitor charge must be restored

Static RAM

- Bits stored as on/off switches
- No charges to leak
- No refreshing needed when powered
- More complex construction
- Larger per bit
- More expensive
- Does not need refresh circuits
- Faster
- Cache
- Digital
  - Uses flip-flops

SRAM v DRAM

- Both volatile
  - Power needed to preserve data
- Dynamic cell
  - Simpler to build, smaller
  - More dense
  - Less expensive
  - Needs refresh
  - Larger memory units
- Static
  - Faster
  - Cache

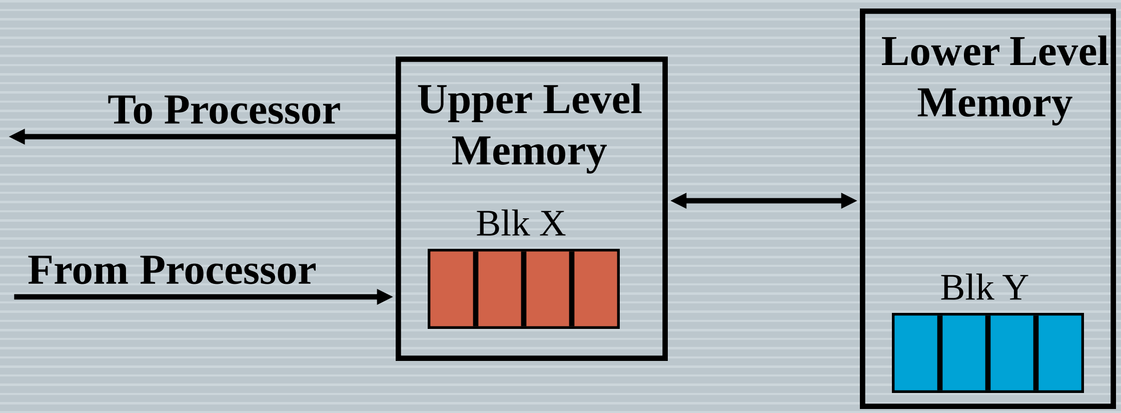
Memory Hierarchy: How Does it Work?

- Temporal Locality (Locality in Time):

  => Keep most recently accessed data items closer to the processor

- Spatial Locality (Locality in Space):

  => Move blocks consists of contiguous words to the upper levels

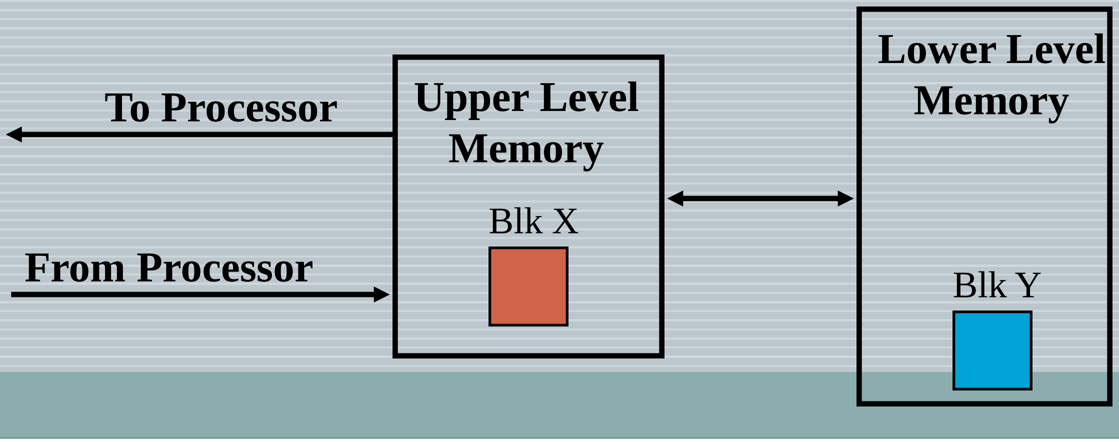


Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of

    RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level +

    Time to deliver the block the processor
- Hit Time << Miss Penalty



To Processor

From Processor

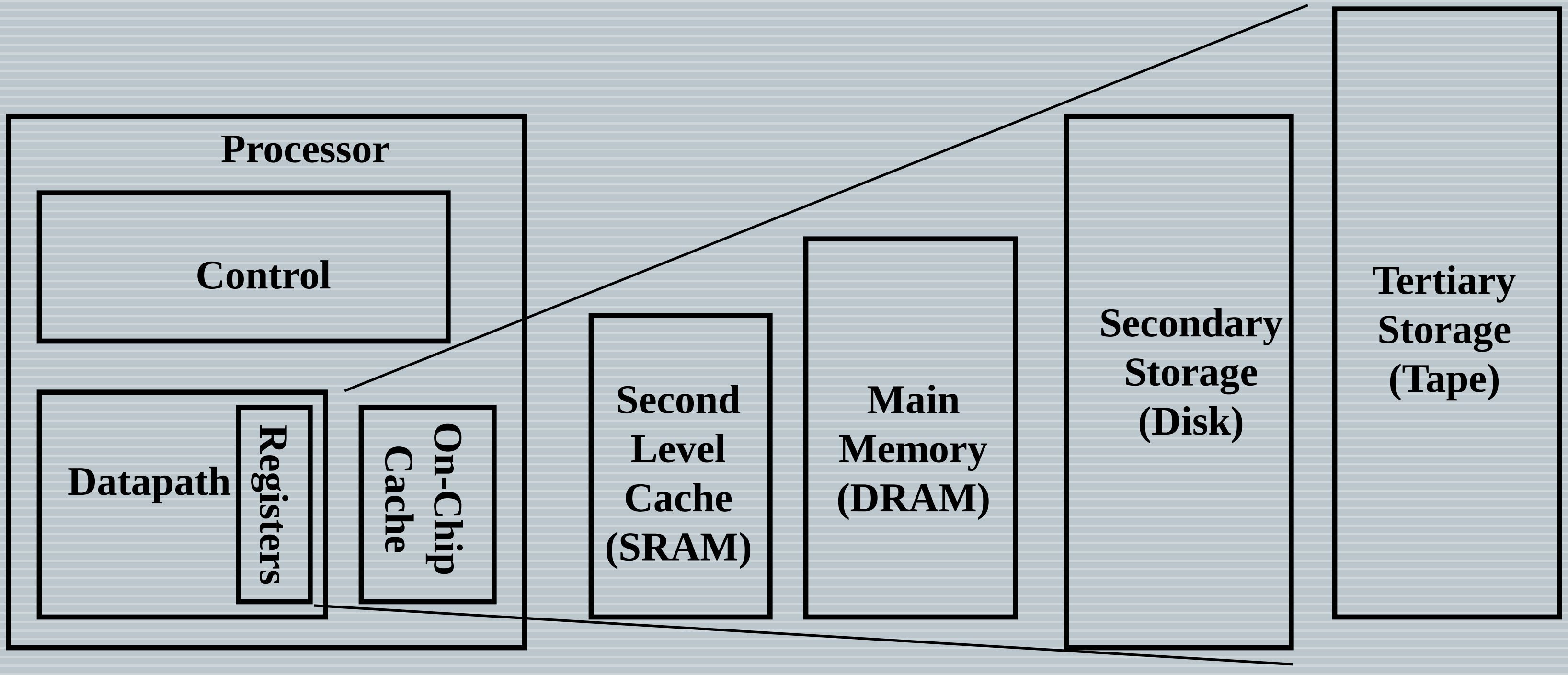
Upper Level Memory

Blk X

Lower Level Memory

Blk Y

Memory Hierarchy of a Modern Computer System



Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

Week 6

Cache Memory Design
Characteristics

- Location
- Capacity
- Unit of transfer
- Access method
- Performance
- Physical type
- Physical characteristics
- Organisation

Location

- CPU
- Internal
- External

Capacity

- Word size
  - The natural unit of organisation
- Number of words
  - or Bytes

Unit of Transfer

- Internal
  - Usually governed by data bus width
- External
  - Usually a block which is much larger than a word
- Addressable unit
  - Smallest location which can be uniquely addressed
  - Word internally
  - Cluster on M$ disks

Access Methods (1)

- Sequential
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location
  - e.g. tape
- Direct
  - Individual blocks have unique address
  - Access is by jumping to vicinity plus sequential search
  - Access time depends on location and previous location
  - e.g. disk

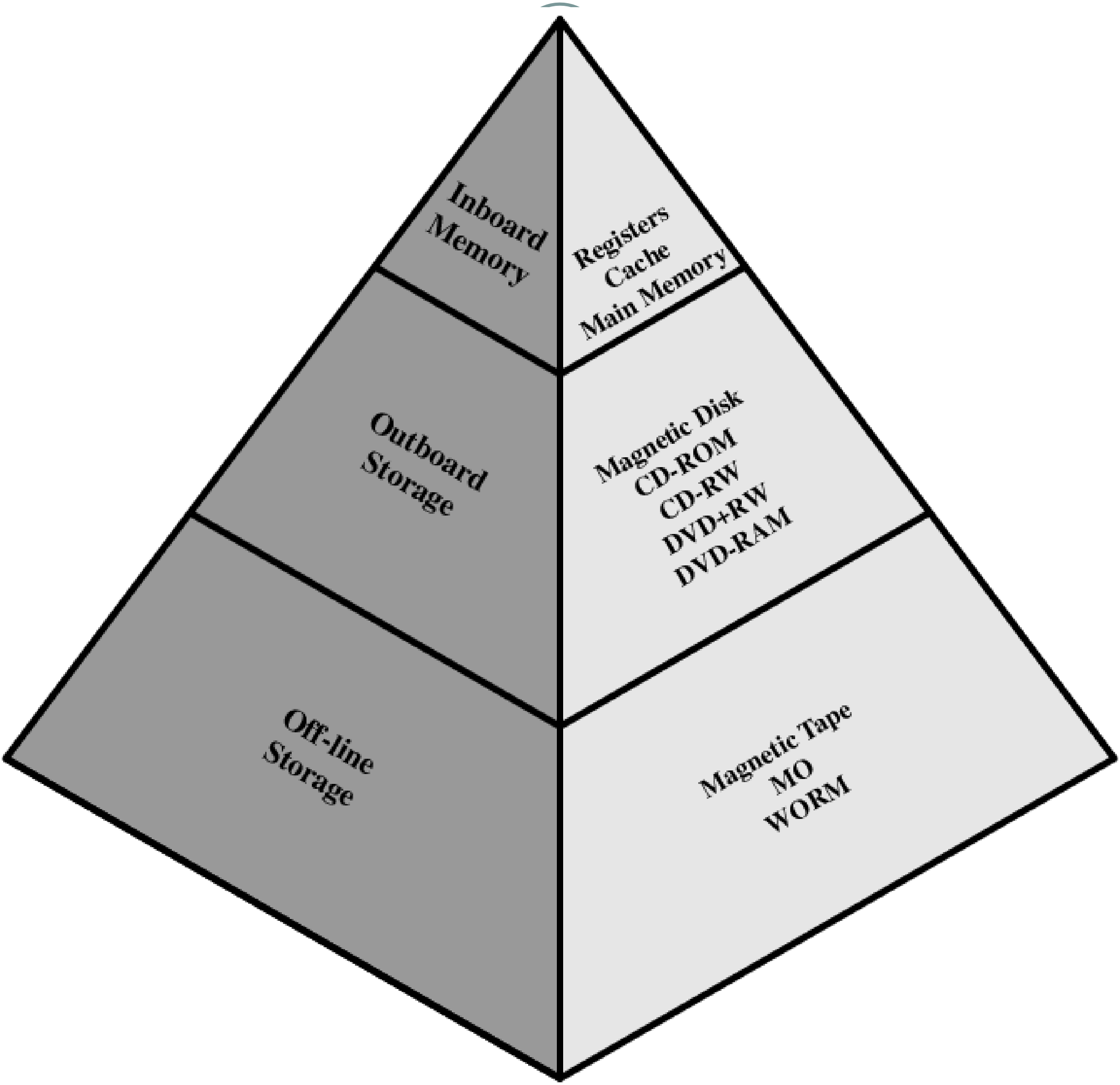
Access Methods (2)

- Random
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM
- Associative
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of location or previous access
  - e.g. cache

Memory Hierarchy

- Registers
  - In CPU
- Internal or Main memory
  - May include one or more levels of cache
  - “RAM”
- External memory
  - Backing store

Memory Hierarchy - Diagram



Performance

- Access time
  - Time between presenting the address and getting the valid data
- Memory Cycle time
  - Time may be required for the memory to “recover” before next access
  - Cycle time is access + recovery
- Transfer Rate
  - Rate at which data can be moved

Physical Types

- Semiconductor
  - RAM
- Magnetic
  - Disk & Tape
- Optical
  - CD & DVD
- Others
  - Bubble
  - Hologram

Physical Characteristics

- Decay
- Volatility
- Erasable
- Power consumption

Organisation

- Physical arrangement of bits into words
- Not always obvious
- e.g. interleaved

The Bottom Line

- How much?
  - Capacity
- How fast?
  - Time is money
- How expensive?

Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

So you want fast?

- It is possible to build a computer which uses only static RAM (see later)
- This would be very fast
- This would need no cache
  - How can you cache cache?
- This would cost a very large amount

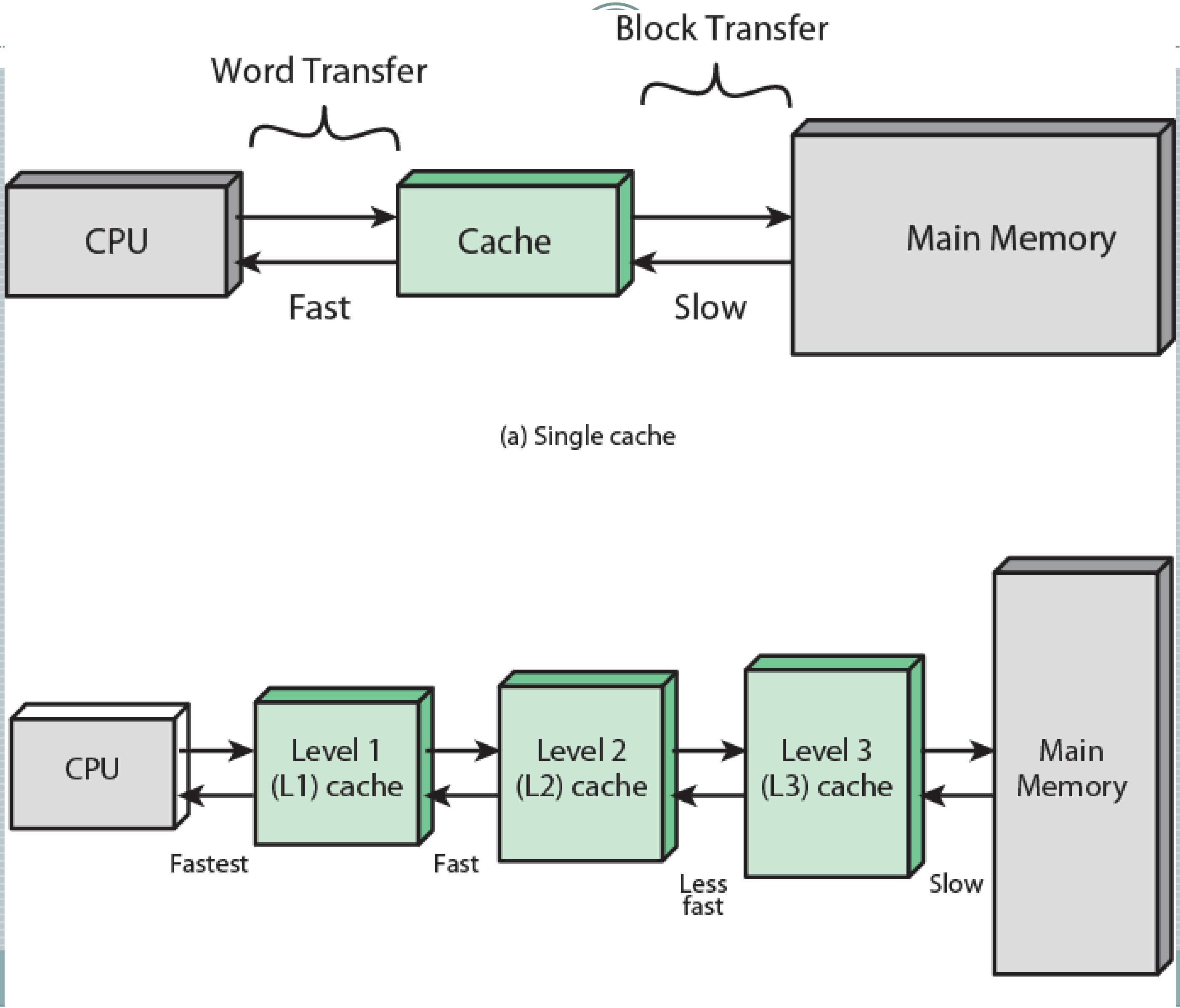
Locality of Reference

- During the course of the execution of a program, memory references tend to cluster
- e.g. loops

Cache

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module

Cache and Main Memory



(a) Single cache

(b) Three-level cache organization

Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Cache Design

- Addressing
- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Cache Addressing

- Where does cache sit?
  - Between processor and virtual memory management unit
  - Between MMU and main memory
- Logical cache (virtual cache) stores data using virtual addresses
  - Processor accesses cache directly, not thorough physical cache
  - Cache access faster, before MMU address translation
  - Virtual addresses use same address space for different applications
    - Must flush cache on each context switch
- Physical cache stores data using main memory physical addresses

Size does matter

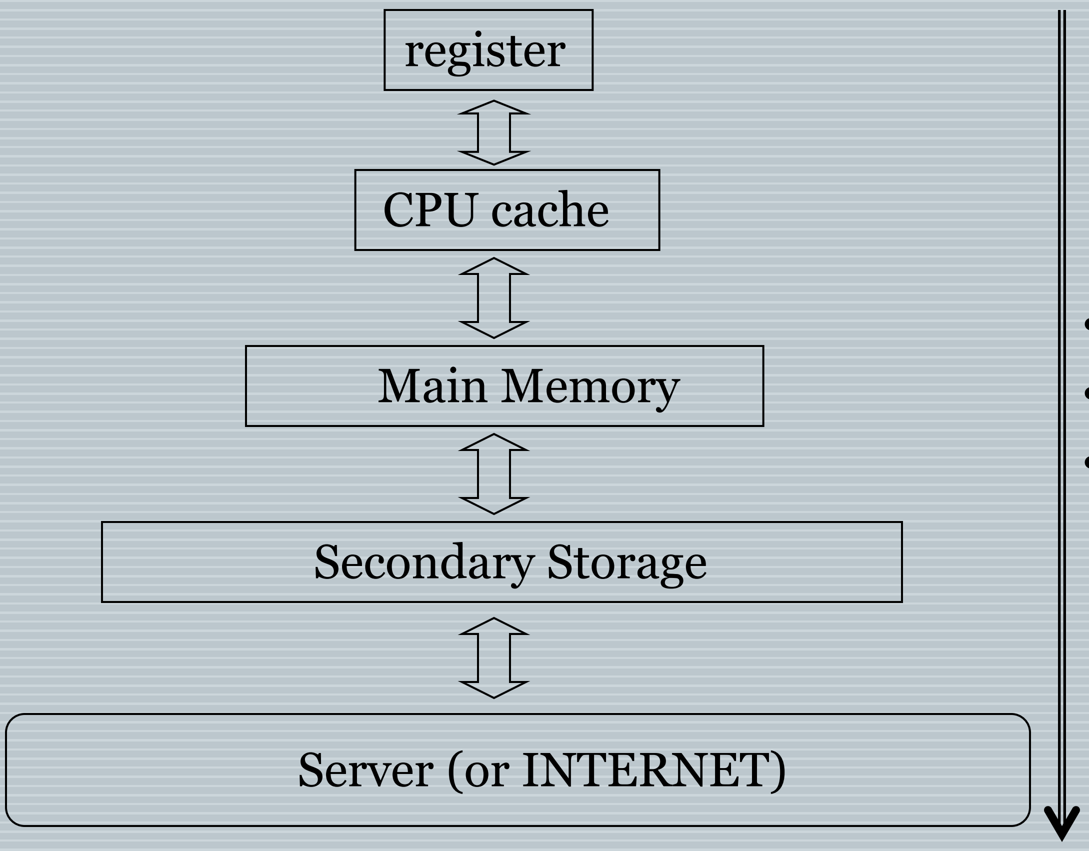
- Cost
  - More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Checking cache for data takes time

Comparison of Cache Sizes

Processor	Type	Year of Introduction	L1 cache	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/server	2002	32 KB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 KB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 KB/64 KB	1MB	—

Week 7

Virtual Memory



register

CPU cache

Main Memory

Secondary Storage

Server (or INTERNET)

- larger capacity
- lower speed
- lower cost

What is…

- Virtual memory as an alternate set of memory addresses.
- Programs use these virtual addresses rather than real addresses to store instructions and data.
- When the program is actually executed, the virtual addresses are converted into real memory addresses.

History

- virtual memory was developed in approximately 1959 – 1962, at the University of Manchester for the Atlas Computer, completed in 1962.
- In 1961, Burroughs released the B5000, the first commercial computer with virtual memory.

Why is it needed….

- Before the development of the virtual memory technique, programmers in the 1940s and 1950s had to manage directly two-level storage such as main memory or ram and secondary memory in the form of hard disks or earlier, magnetic drums.

- Enlarge the address space, the set of addresses a program can utilize.
- Virtual memory might contain twice as many addresses as main memory.

Object…

- When a computer is executing many programs at the same time, Virtual memory make the computer to share memory efficiently.
- Eliminate a restriction that a computer works in memory which is small and be limited.
- When many programs is running at the same time, by distributing each suitable memory area to each program, VM protect programs to interfere each other in each memory area.

How does it work…

- To facilitate copying virtual memory into real memory, the operating system divides virtual memory into pages, each of which contains a fixed number of addresses.
- Each page is stored on a disk until it is needed.
- When the page is needed, the operating system copies it from disk to main memory, translating the virtual addresses into real addresses.

MMU (Memory Management Unit)

- The hardware base that makes a virtual memory system possible.
- Allows software to reference physical memory by virtual addresses, quite often more than one.
- It accomplishes this through the use of page and page tables.
- Use a section of memory to translate virtual addresses into physical addresses via a series of table lookups.
- The software that handles the page fault is generally part of an operating system and the hardware that detects this situation.

Segmentation……

- Segmentation involves the relocation of variable sized segments into the physical address space.
- Generally these segments are contiguous units, and are referred to in programs by their segment number and an offset to the requested data.
- Efficient segmentation relies on programs that are very thoughtfully written for their target system.
- Since segmentation relies on memory that is located in single large blocks, it is very possible that enough free space is available to load a new module, but can not be utilized.
- Segmentation may also suffer from internal fragmentation if segments are not variable-sized, where memory above the segment is not used by the program but is still "reserved" for it.

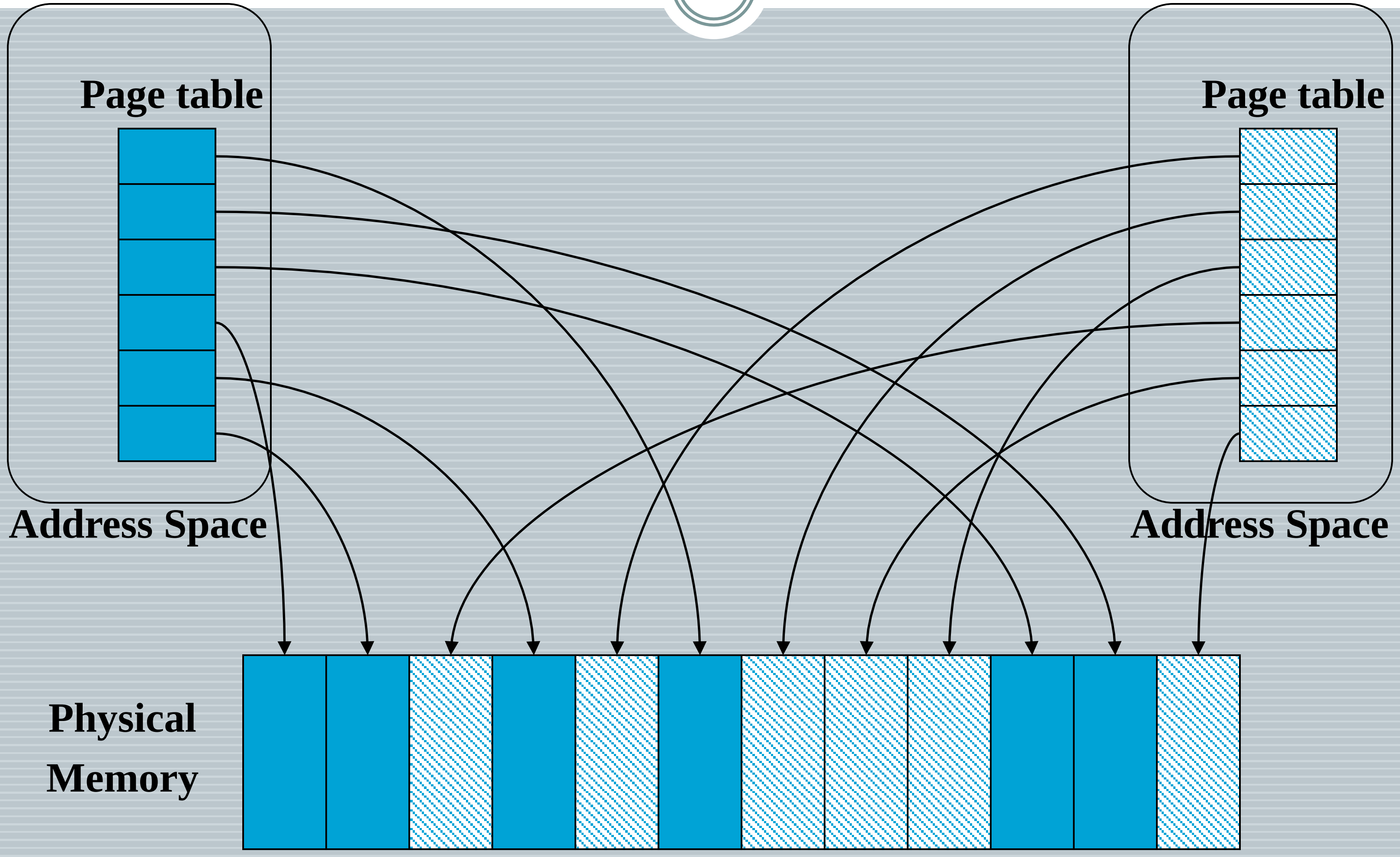
Paging……

- Paging provides a somewhat easier interface for programs, in that its operation tends to be more automatic and thus transparent.
- Each unit of transfer, referred to as a page, is of a fixed size and swapped by the virtual memory manager outside of the program's control.
- Instead of utilizing a segment/offset addressing approach, as seen in segmentation, paging uses a linear sequence of virtual addresses which are mapped to physical memory as necessary.
- Due to this addressing approach, a single program may refer to series of many non-contiguous segments.
- Although some internal fragmentation may still exist due to the fixed size of the pages, the approach virtually eliminates external fragmentation.

Paging……(cont'd)

- A technique used by virtual memory operating systems to help ensure that the data you need is available as quickly as possible.
- The operating system copies a certain number of pages from your storage device to main memory.
- When a program needs a page that is not in maim memory, the operating system copies the required page into memory and copies another page back to the disk.

Virtual Memory (Paging)



Page fault

- An interrupt to the software raised by the hardware when a program accesses a page that is not mapped in physical memory.

- when a program accesses a memory location in its memory and the page corresponding to that memory is not loaded
- when a program accesses a memory location in its memory and the program does not have privileges to access the page corresponding to that memory.

Paging replacement algorithms

- OPT(MIN) : eliminate the page that be not expected to be used.
- FIFO(first input/first output) : rather than choosing the victim page at random, the oldest page is the first to be removed.
- LRU(Least Recently used) : move out the page that is the least rarely used.
- LFU(Least Frequently used) : move out the page that is not used often in the past.

Summary…

- **Virtual memory** is a common part of most operating systems on computers.
- It has become so common because it provides a big benefit for users at a very low cost.
- benefits of executing a program that is only partially in memory.
- program is no longer constrained by the amount of physical memory.

  ⇒ user would be able to write programs for an extremely large virtual address space.
- more programs could be run at the same time

  ⇒ increase CPU utilization and throughput.
- less I/O would be needed to load or swap each user program

  ⇒ run faster

Week 8

Input/Output Systems

- Wide variety of peripherals
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- All slower than CPU and RAM
- Need I/O modules

Input/Output Module

- Interface to CPU and Memory
- Interface to one or more peripherals

External Devices

- Human readable
  - Screen, printer, keyboard
- Machine readable
  - Monitoring and control
- Communication
  - Modem
  - Network Interface Card (NIC)

I/O Module Function

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering
- Error Detection

I/O Steps

- CPU checks I/O module device status
- I/O module returns status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPU
- Variations for output, DMA, etc.

I/O Module Decisions

- Hide or reveal device properties to CPU
- Support multiple or single device
- Control device functions or leave for CPU
- Also O/S decisions
  - e.g. Unix treats everything it can as a file

Input Output Techniques

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

Programmed I/O

- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU waits for I/O module to complete operation
- Wastes CPU time

Programmed I/O - detail

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later

I/O Commands

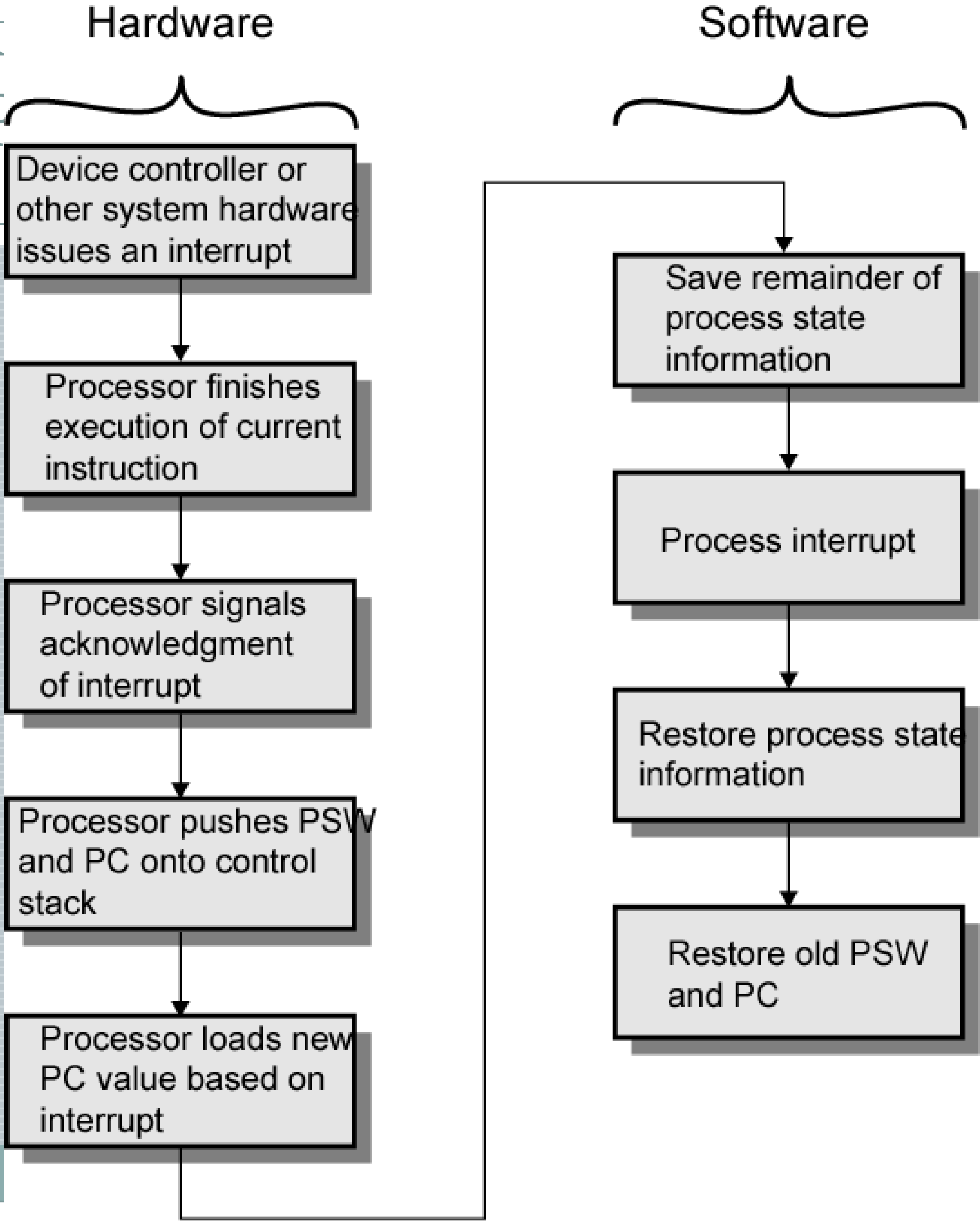
- CPU issues address
  - Identifies module (& device if >1 per module)
- CPU issues command
  - Control - telling module what to do
    - e.g. spin up disk
  - Test - check status
    - e.g. power? Error?
  - Read/Write
    - Module transfers data via buffer from/to device

Week 9

Interrupt Driven I/O and Interfacing Basic Operation

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

Sim



Hardware

Software

Device controller or other system hardware issues an interrupt

Processor finishes execution of current instruction

Processor signals acknowledgment of interrupt

Processor pushes PSW and PC onto control stack

Processor loads new PC value based on interrupt

Save remainder of process state information

Process interrupt

Restore process state information

Restore old PSW and PC

CPU Viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:-
  - Save context (registers)
  - Process interrupt
    - Fetch data & store
- See Operating Systems notes

Design Issues

- How do you identify the module issuing the interrupt?
- How do you deal with multiple interrupts?
  - i.e. an interrupt handler being interrupted

Identifying Interrupting Module (1)

- Different line for each module
  - PC
  - Limits number of devices
- Software poll
  - CPU asks each module in turn
  - Slow

Identifying Interrupting Module (2)

- Daisy Chain or Hardware poll
  - Interrupt Acknowledge sent down a chain
  - Module responsible places vector on bus
  - CPU uses vector to identify handler routine
- Bus Master
  - Module must claim the bus before it can raise interrupt
  - e.g. PCI & SCSI

Multiple Interrupts

- Each interrupt line has a priority
- Higher priority lines can interrupt lower priority lines
- If bus mastering only current master can interrupt

Example - PC Bus

- 80x86 has one interrupt line
- 8086 based systems use one 8259A interrupt controller
- 8259A has 8 interrupt lines

Sequence of Events

- 8259A accepts interrupts
- 8259A determines priority
- 8259A signals 8086 (raises INTR line)
- CPU Acknowledges
- 8259A puts correct vector on data bus
- CPU processes interrupt

ISA Bus Interrupt System

- ISA bus chains two 8259As together
- Link is via interrupt 2
- Gives 15 lines
  - 16 lines less one for link
- IRQ 9 is used to re-route anything trying to use IRQ 2
  - Backwards compatibility
- Incorporated in chip set

Direct Memory Access

- Interrupt driven and programmed I/O require active CPU intervention
  - Transfer rate is limited
  - CPU is tied up
- DMA is the answer

Week 10

Multiprocessor and Multicomputer

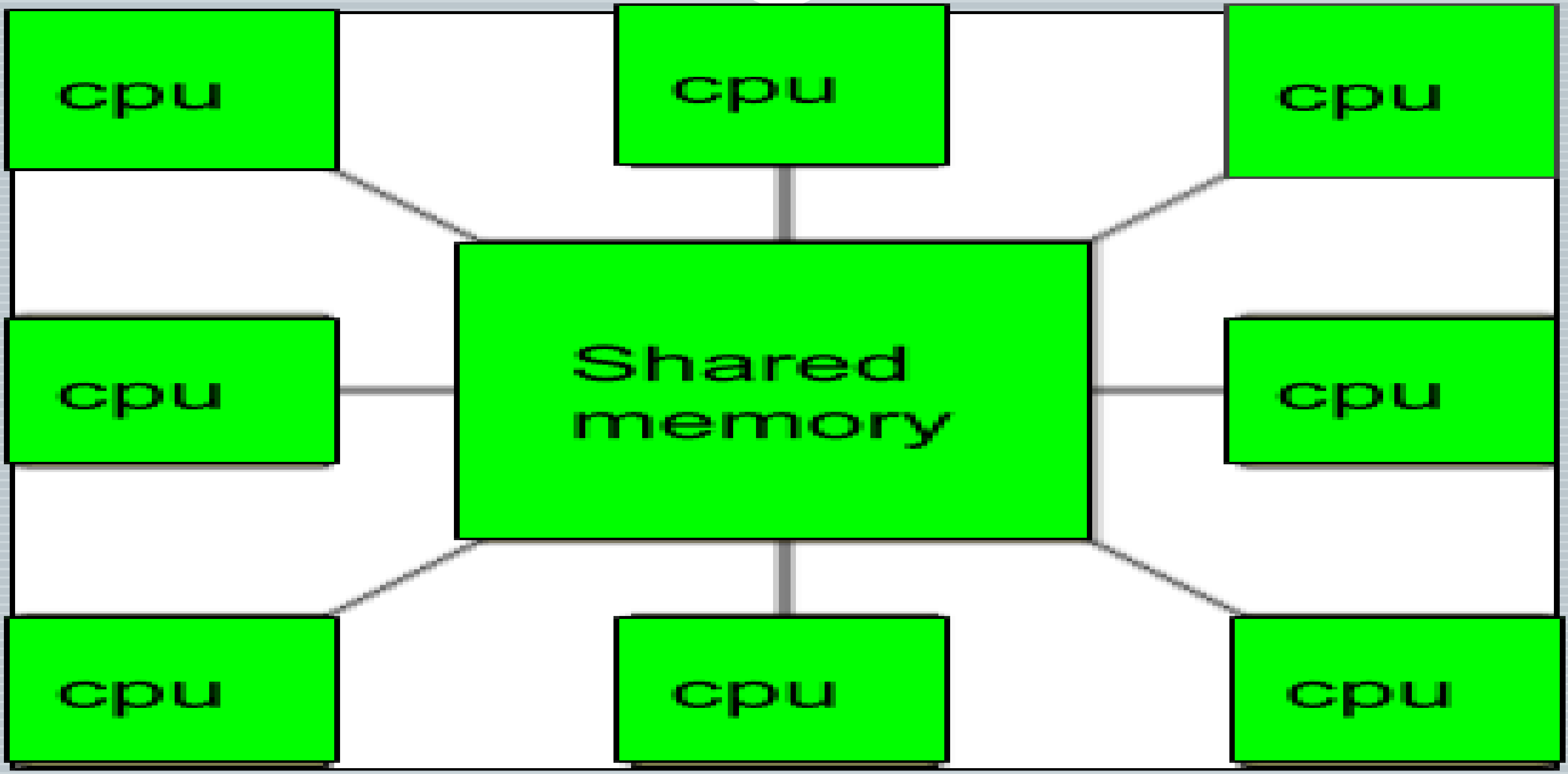
A MULTIPROCESSOR SYSTEM IS A TYPE OF COMPUTING ARCHITECTURE THAT INTEGRATES TWO OR MORE PROCESSORS WITHIN A SINGLE MACHINE TO ENHANCE PERFORMANCE

Types of Multiprocessor Systems

- **Symmetric Multiprocessing (SMP)**: In SMP systems, each processor is identical and shares a single operating system, memory, and storage resources. Processors work collaboratively and communicate directly, allowing them to balance workloads efficiently.

- **Asymmetric Multiprocessing (AMP)**: Unlike SMP, AMP systems feature processors with specialized roles. One processor typically manages the entire system, while others perform specific tasks assigned by the main processor.

Multiprocessor



Benefits of using a Multiprocessor

- Enhanced performance.
- Multiple applications.
- Multi-tasking inside an application.
- High throughput and responsiveness.
- Hardware sharing among CPUs.

Multiprocessor Connection

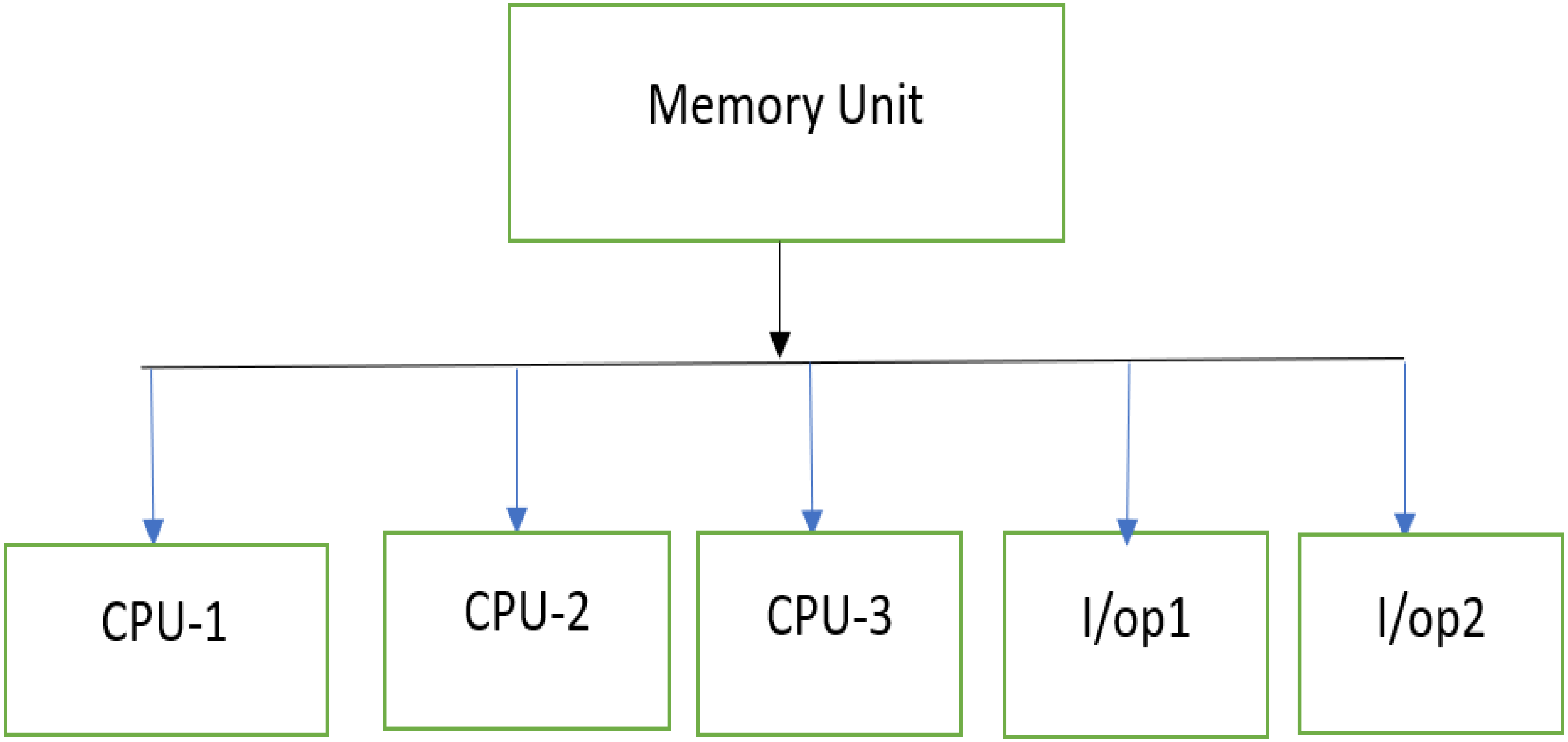
The interconnection among two or more processor and shared memory is done with three methods

1)Time shared common bus

2)Multiport memories

3)Crossbar switch network

Time shared common bus



- A single shared bus through which all processor & memory unit can be communicated.

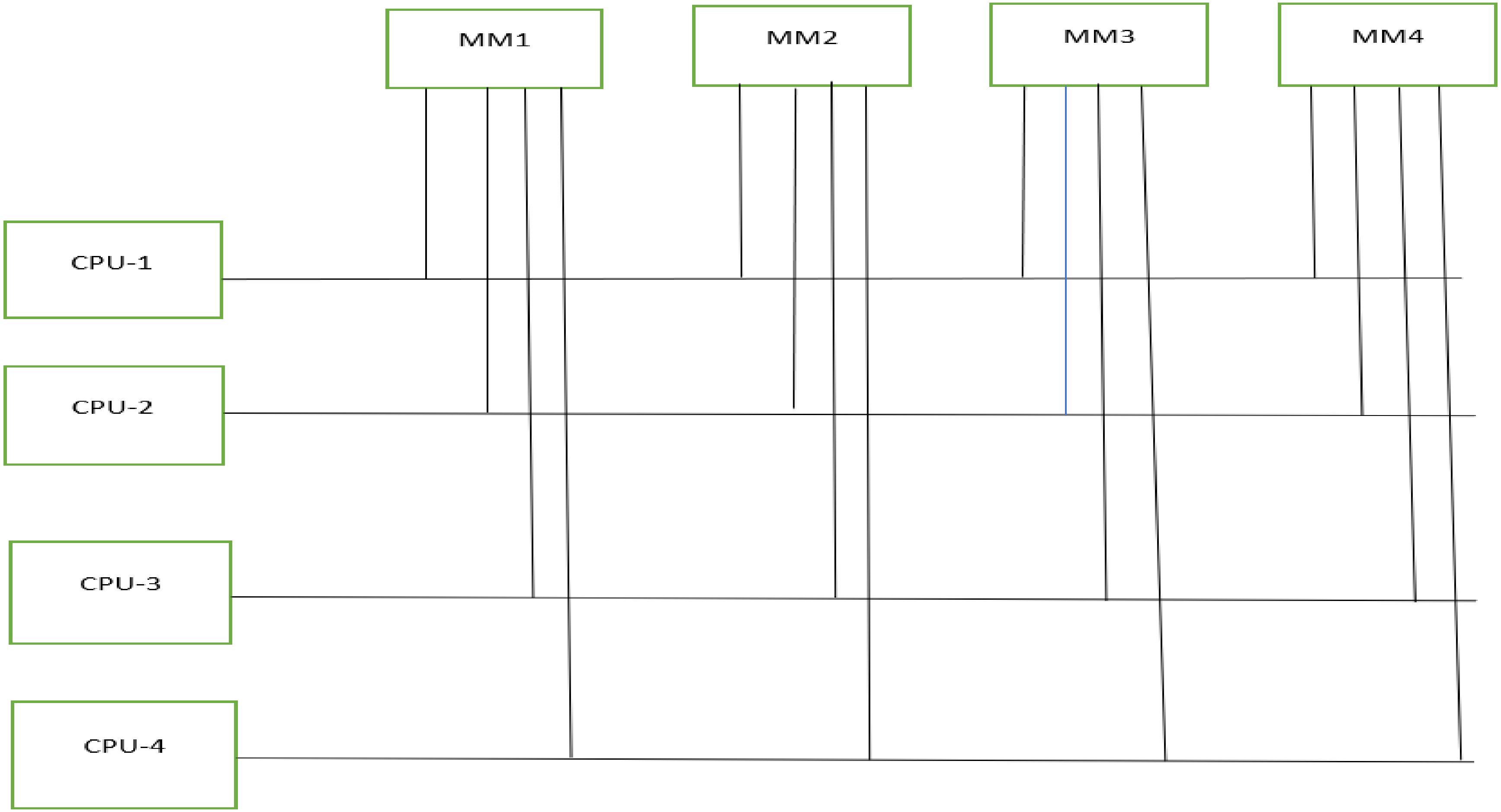
Advantage:

- Simple to implement.
- Due to single common bus cost to implement is very less.

Disadvantage:

- **Data transfer rate is slow.**

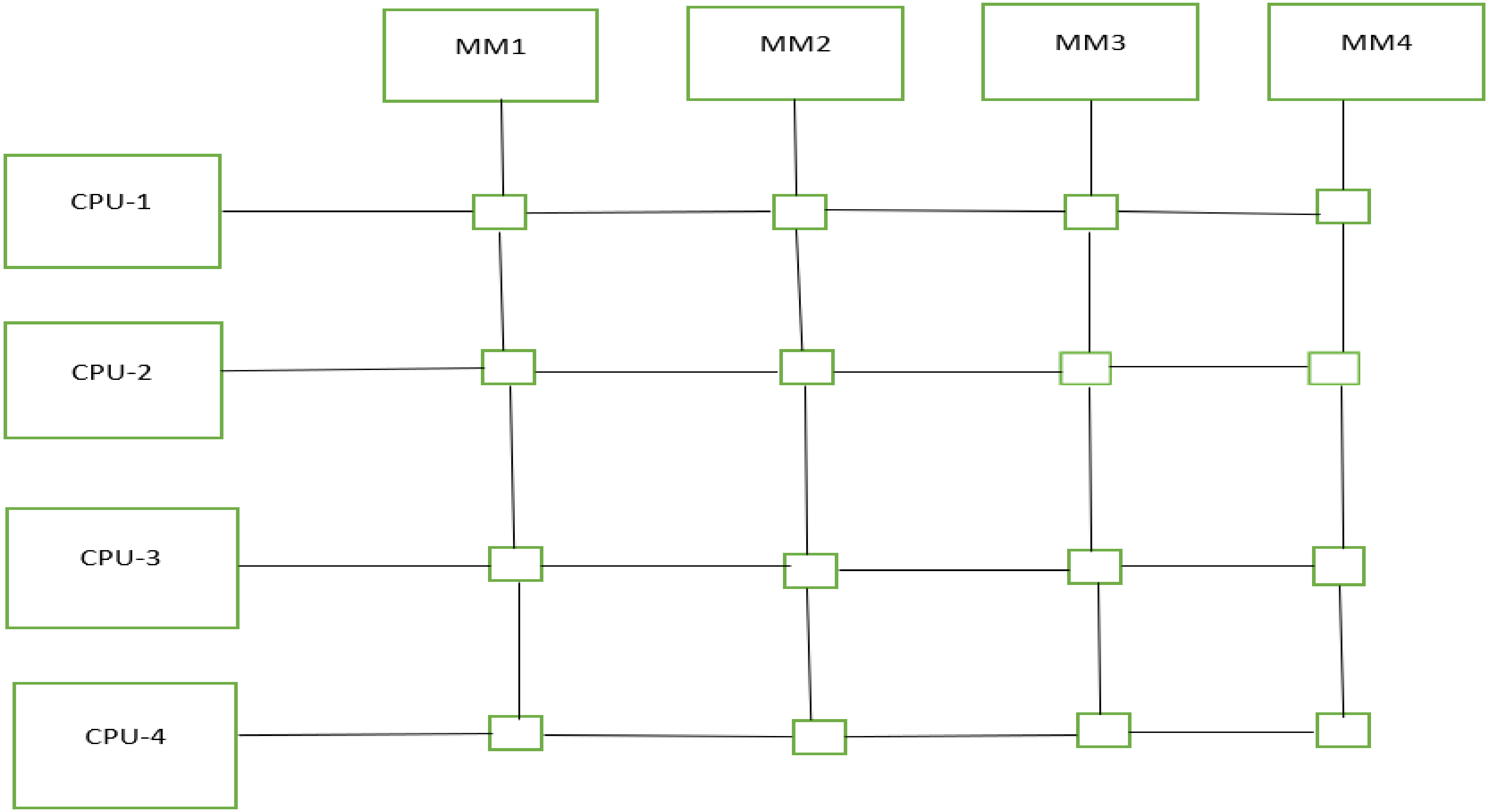
Multiport memories



Multiport memories

- separate bus for each processor to communicate with the memory module.
- all the process can be communicated parallel.

Crossbar switch network



Crossbar switch network

- a switch will be installed between memory unit and CPU
- Switch is responsible for whether to pass the request to a particular memory module or not based on the request made for.

Advantage:

- High data through rate.

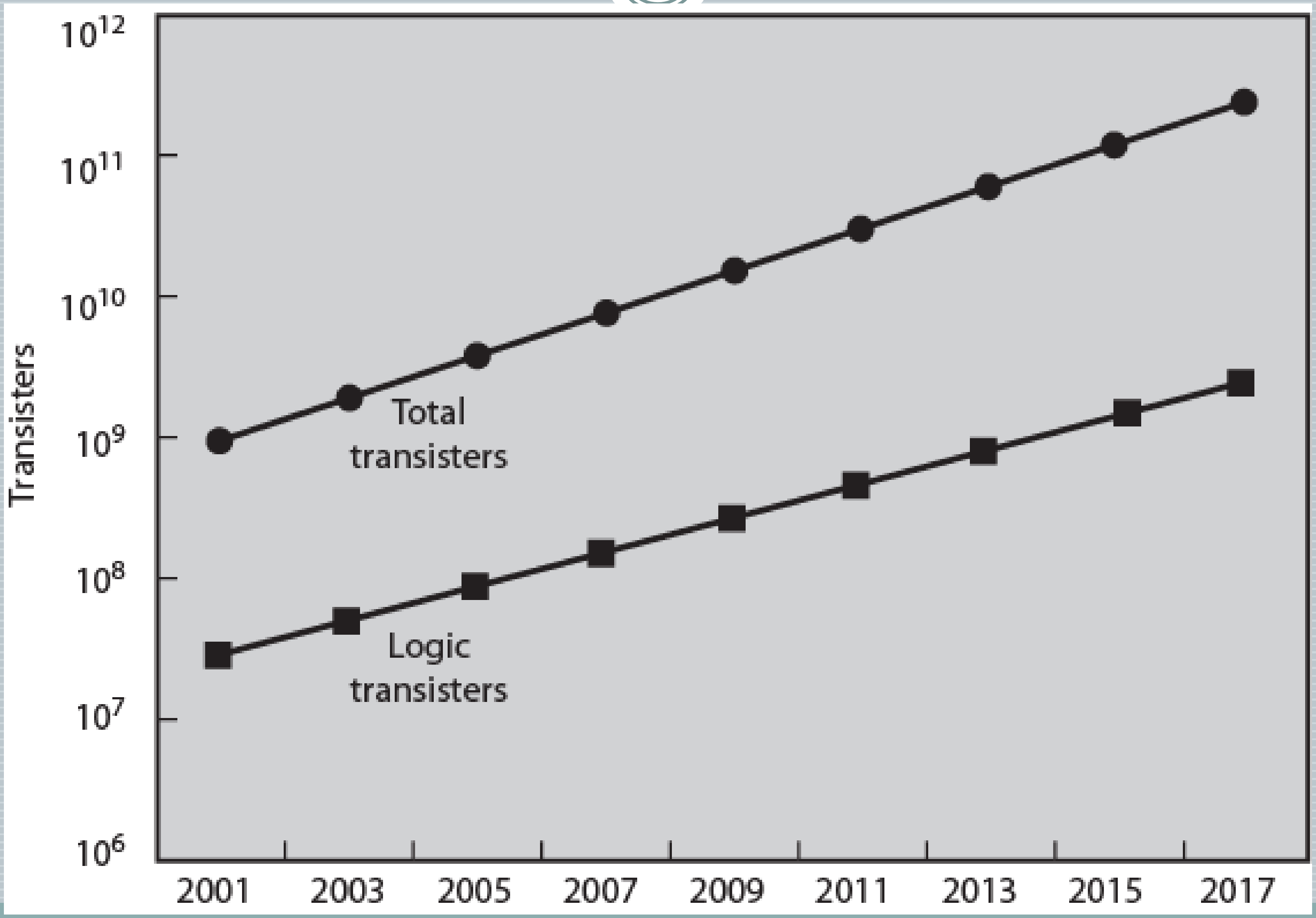
Disadvantage:

- **Complex to implement as more switches involved.**
- Costlier to implement.

Multi Core Processor
Hardware Performance Issues

- Microprocessors have seen an exponential increase in performance
  - Improved organization
  - Increased clock frequency
- Increase in Parallelism
  - Pipelining
  - Superscalar
  - Simultaneous multithreading (SMT)
- Diminishing returns
  - More complexity requires more logic
  - Increasing chip area for coordinating and signal transfer logic
    - Harder to design, make and debug

Chip Utilization of Transistors



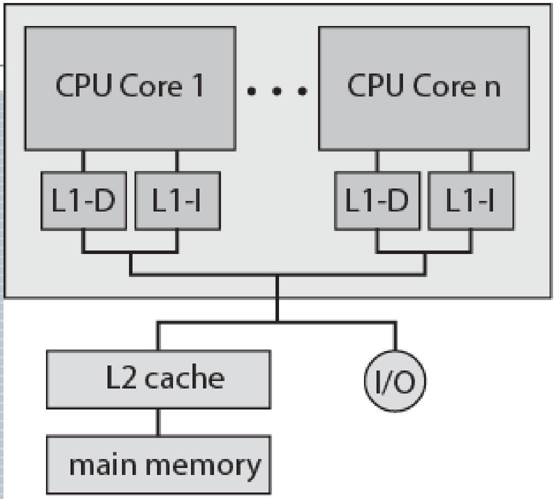
Software Performance Issues

- Performance benefits dependent on effective exploitation of parallel resources
- Even small amounts of serial code impact performance
  - 10% inherently serial on 8 processor system gives only 4.7 times performance
- Communication, distribution of work and cache coherence overheads
- Some applications effectively exploit multicore processors

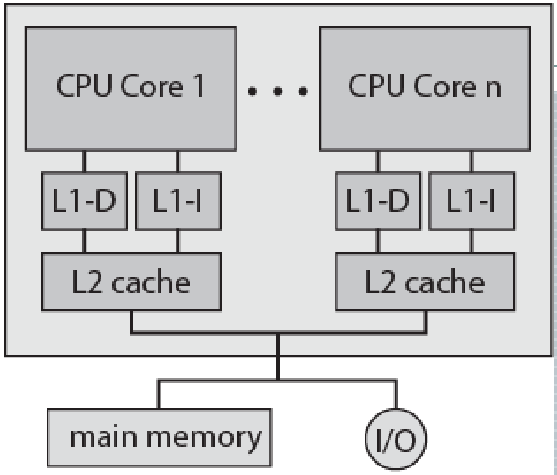
Multicore Organization

- Number of core processors on chip
- Number of levels of cache on chip
- Amount of shared cache
- Next slide examples of each organization:
- (a) ARM11 MPCore
- (b) AMD Opteron
- (c) Intel Core Duo
- (d) Intel Core i7

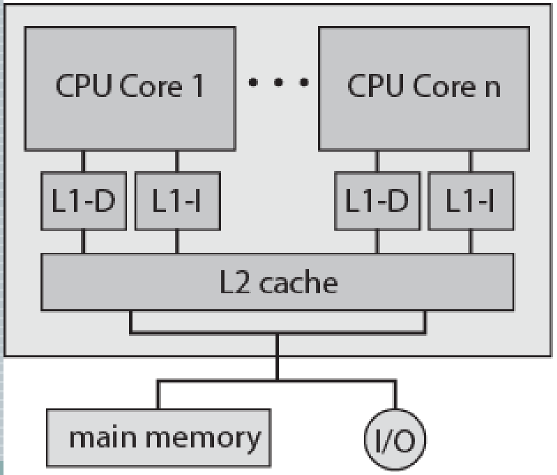
Multicore Organization Alternatives



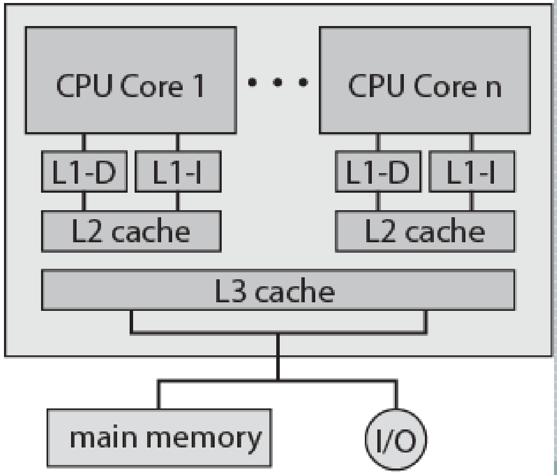
(a) Dedicated L1 cache



(b) Dedicated L2 cache



(c) Shared L2 cache



(d) Shared L3 cache

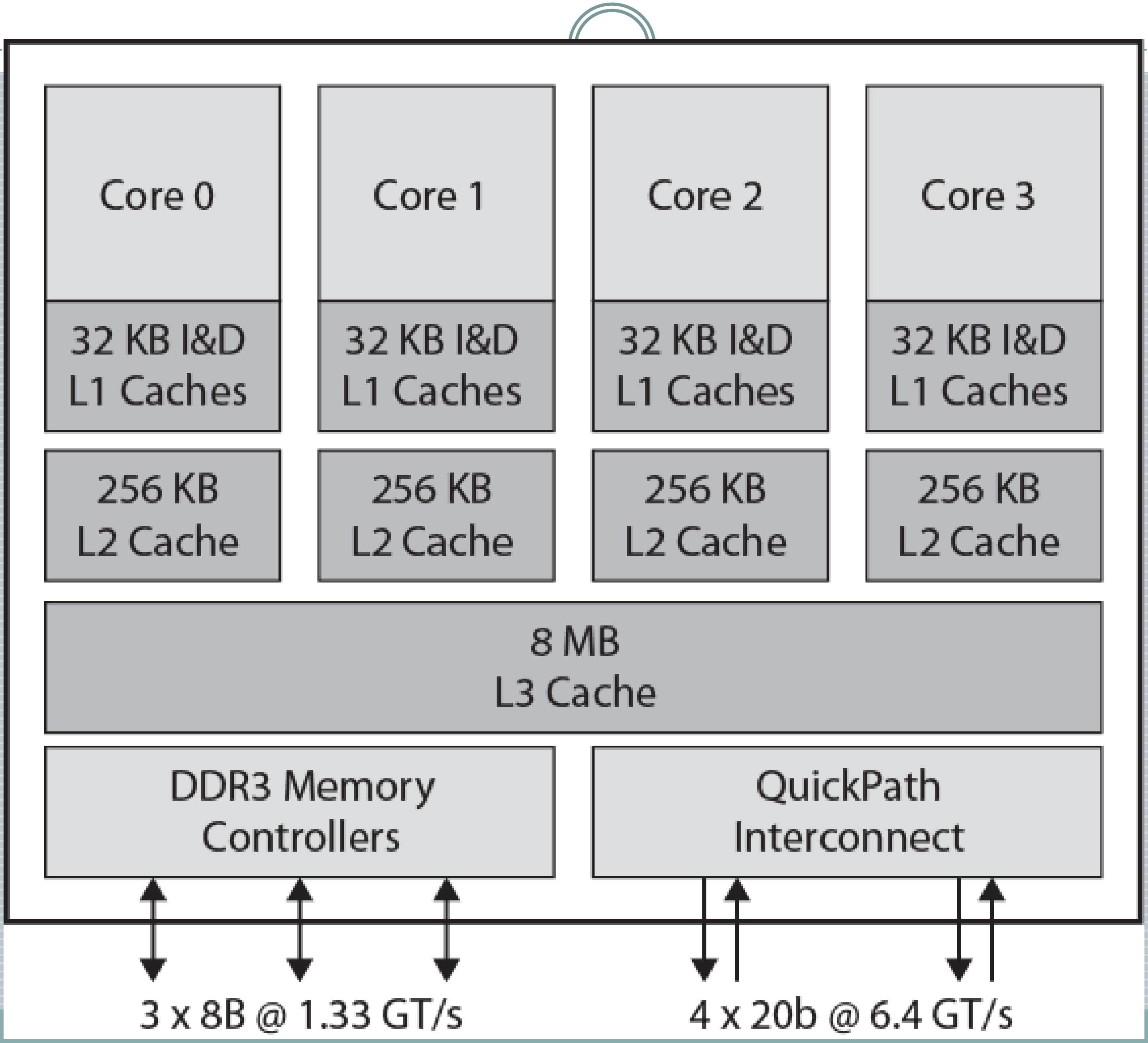
Advantages of shared L2 Cache

- Constructive interference reduces overall miss rate
- Data shared by multiple cores not replicated at cache level
- With proper frame replacement algorithms mean amount of shared cache dedicated to each core is dynamic
  - Threads with less locality can have more cache
- Easy inter-process communication through shared memory
- Cache coherency confined to L1
- Dedicated L2 cache gives each core more rapid access
  - Good for threads with strong locality
- Shared L3 cache may also improve performance

Individual Core Architecture

- Intel Core Duo uses superscalar cores
- Intel Core i7 uses simultaneous multi-threading (SMT)
  - Scales up number of threads supported
    - 4 SMT cores, each supporting 4 threads appears as 16 core

Intel Core i& Block Diagram



Week 11

Parallel Processing

Multiple Processor Organization

- Single instruction, single data stream - SISD
- Single instruction, multiple data stream - SIMD
- Multiple instruction, single data stream - MISD
- Multiple instruction, multiple data stream- MIMD

Single Instruction, Single Data Stream - SISD

- Single processor
- Single instruction stream
- Data stored in single memory
- Uni-processor

Single Instruction, Multiple Data Stream - SIMD

- Single machine instruction
- Controls simultaneous execution
- Number of processing elements
- Lockstep basis
- Each processing element has associated data memory
- Each instruction executed on different set of data by different processors
- Vector and array processors

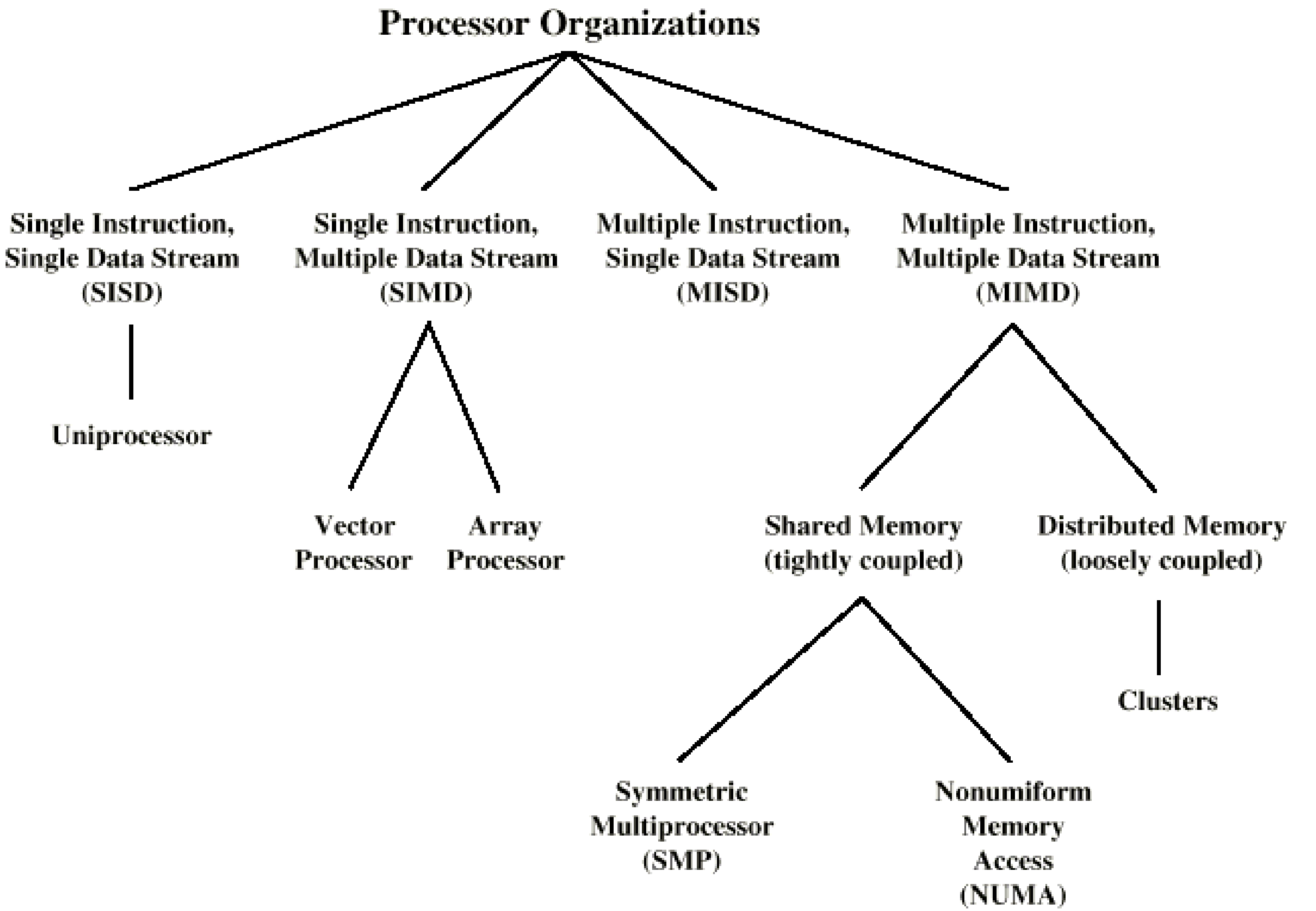
Multiple Instruction, Single Data Stream - MISD

- Sequence of data
- Transmitted to set of processors
- Each processor executes different instruction sequence
- Never been implemented

Multiple Instruction, Multiple Data Stream- MIMD

- Set of processors
- Simultaneously execute different instruction sequences
- Different sets of data
- SMPs, clusters and NUMA systems

Taxonomy of Parallel Processor Architectures


Processor Organizations
Single Instruction, Single Data Stream (SISD)
Single Instruction, Multiple Data Stream (SIMD)
Multiple Instruction, Single Data Stream (MISD)
Multiple Instruction, Multiple Data Stream (MIMD)
Uniprocessor
Vector Processor
Array Processor
Shared Memory (tightly coupled)
Distributed Memory (loosely coupled)
Clusters
Symmetric Multiprocessor (SMP)
Nonumiform Memory Access (NUMA)

MIMD - Overview

- General purpose processors
- Each can process all instructions necessary
- Further classified by method of processor communication

Tightly Coupled - SMP

- Processors share memory
- Communicate via that shared memory
- Symmetric Multiprocessor (SMP)
  - Share single memory or pool
  - Shared bus to access memory
  - Memory access time to given area of memory is approximately the same for each processor

Tightly Coupled - NUMA

- Nonuniform memory access
- Access times to different regions of memroy may differ

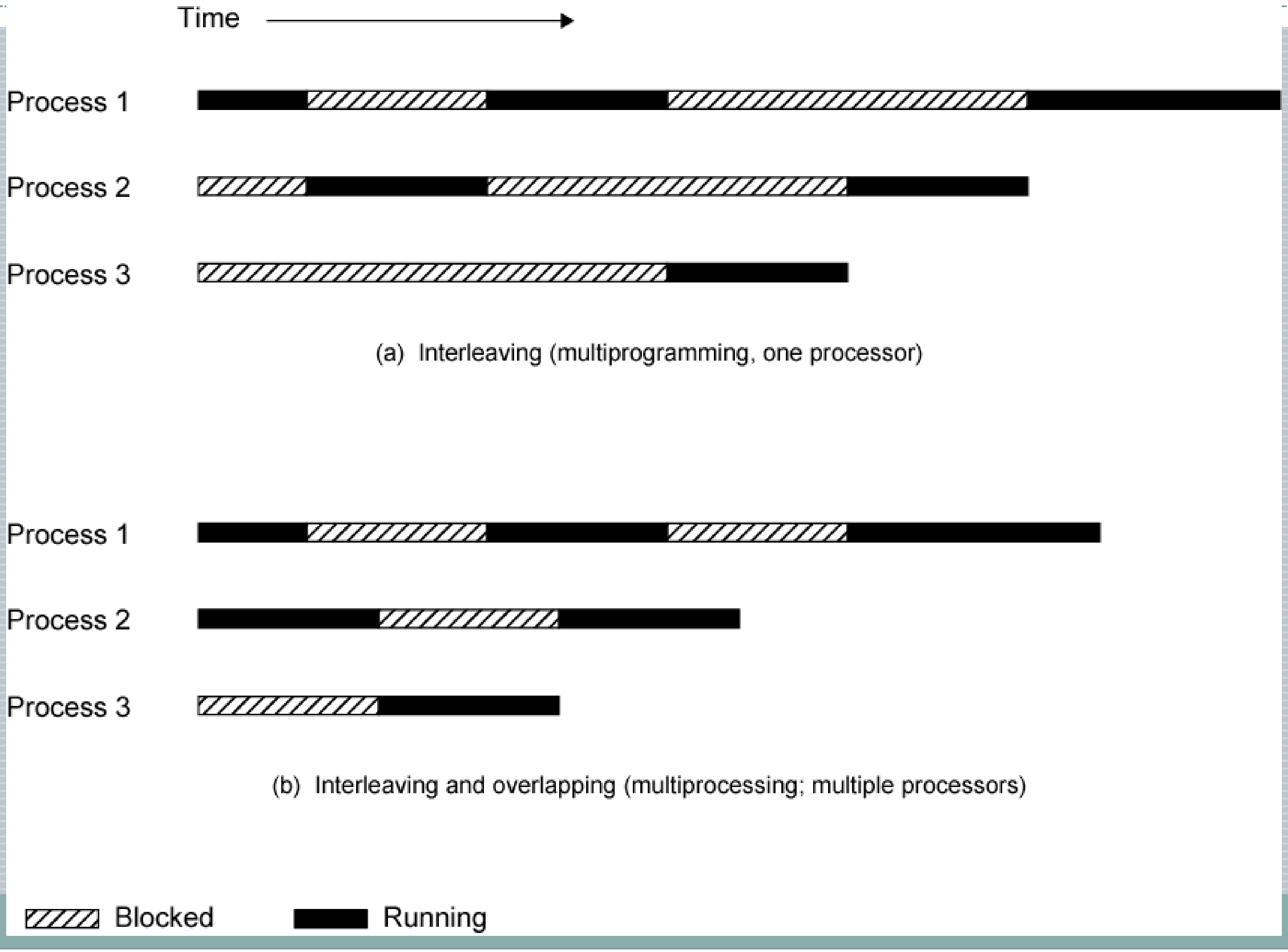
Loosely Coupled - Clusters

- Collection of independent uniprocessors or SMPs
- Interconnected to form a cluster
- Communication via fixed path or network connections

Symmetric Multiprocessors

- A stand alone computer with the following characteristics
  - Two or more similar processors of comparable capacity
  - Processors share same memory and I/O
  - Processors are connected by a bus or other internal connection
  - Memory access time is approximately the same for each processor
  - All processors share access to I/O
    - Either through same channels or different channels giving paths to same devices
  - All processors can perform the same functions (hence symmetric)
  - System controlled by integrated operating system
    - providing interaction between processors
    - Interaction at job, task, file and data element levels

Multiprogramming and Multiprocessing



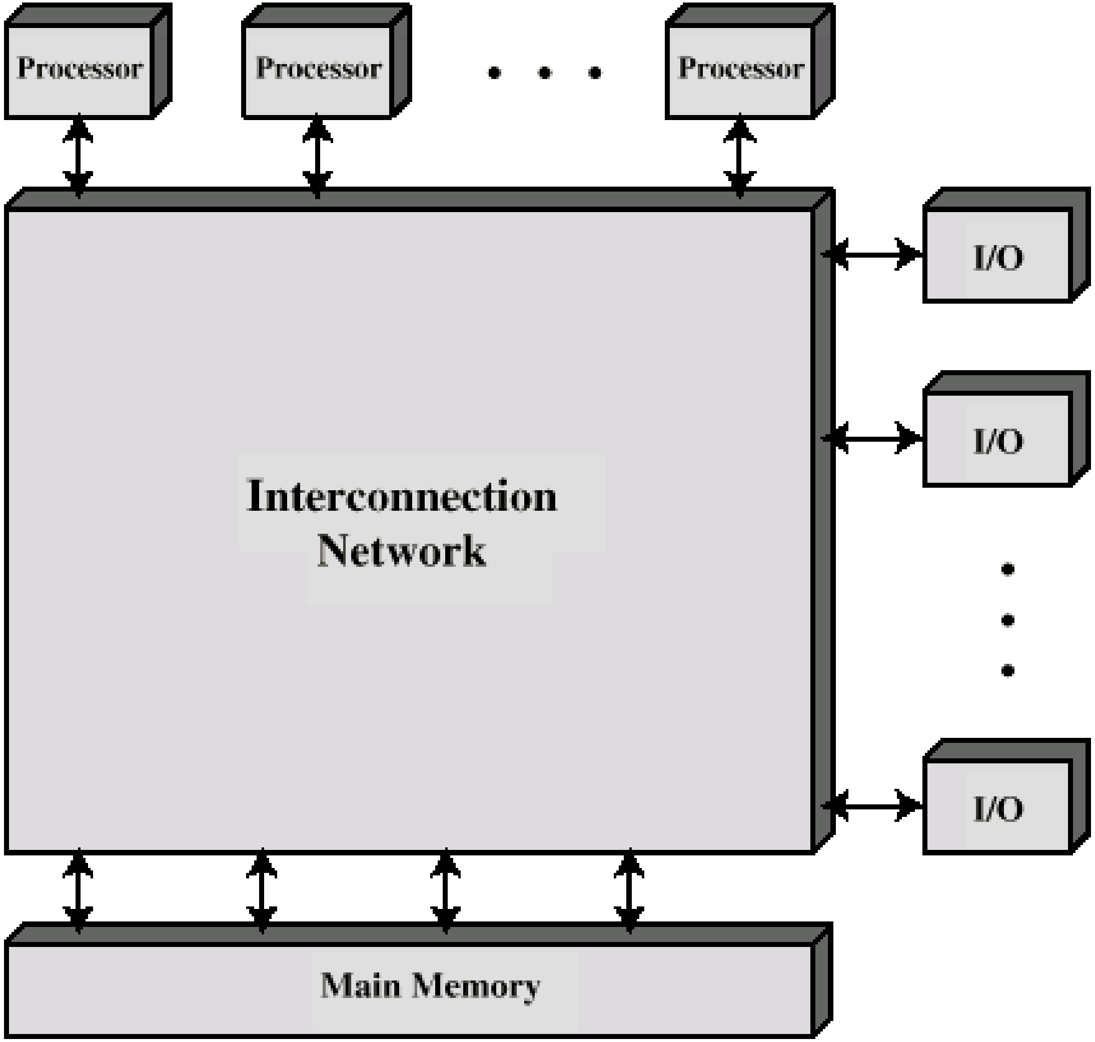
(a) Interleaving (multiprogramming, one processor)

(b) Interleaving and overlapping (multiprocessing; multiple processors)

SMP Advantages

- Performance
  - If some work can be done in parallel
- Availability
  - Since all processors can perform the same functions, failure of a single processor does not halt the system
- Incremental growth
  - User can enhance performance by adding additional processors
- Scaling
  - Vendors can offer range of products based on number of processors

Block Diagram of Tightly Coupled Multiprocessor



Week 12

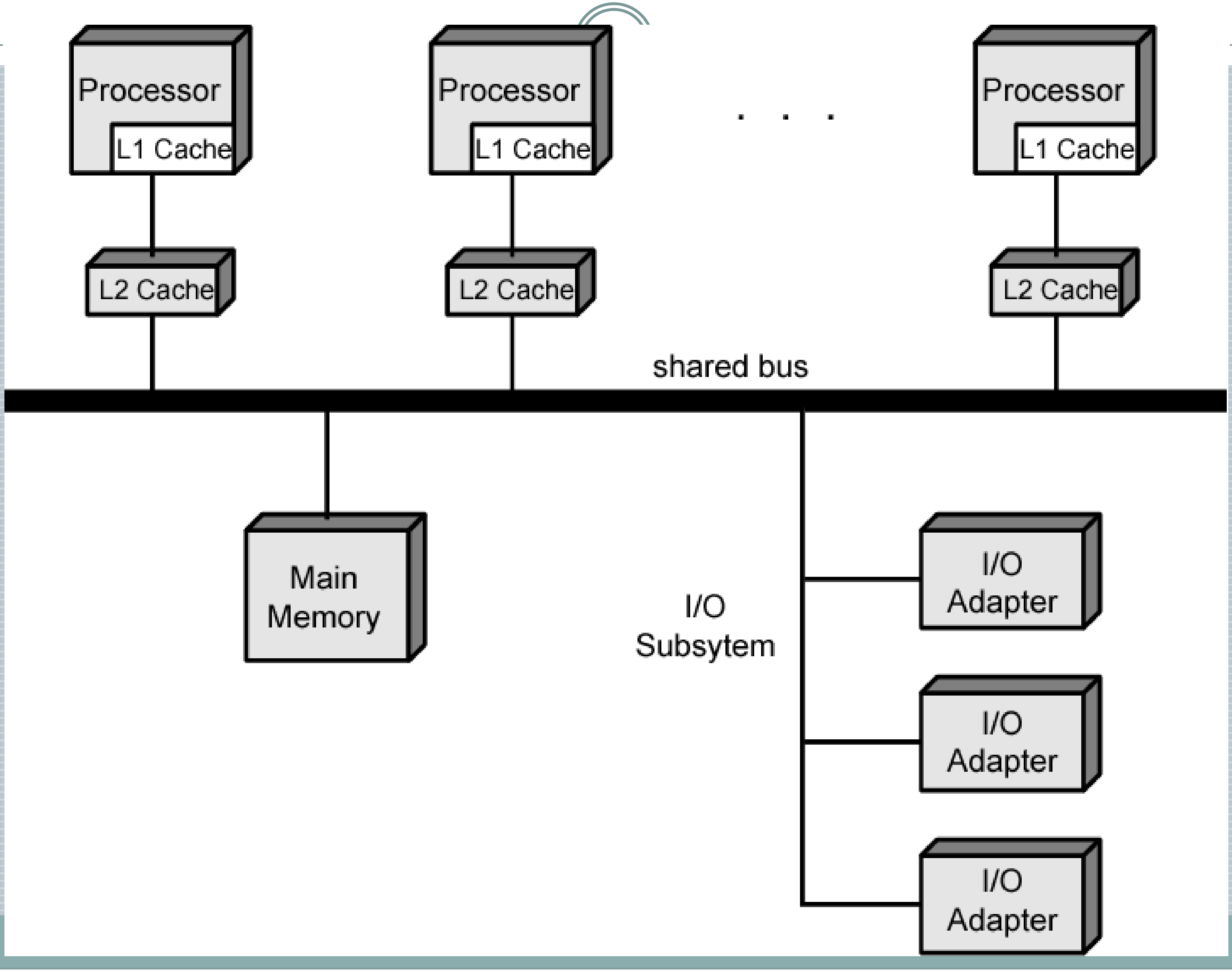
Processor Organization Classification

- Time shared or common bus
- Multiport memory
- Central control unit

Time Shared Bus

- Simplest form
- Structure and interface similar to single processor system
- Following features provided
  - Addressing - distinguish modules on bus
  - Arbitration - any module can be temporary master
  - Time sharing - if one module has the bus, others must wait and may have to suspend
- Now have multiple processors as well as multiple I/O modules

Symmetric Multiprocessor Organization



Time Share Bus - Advantages

- Simplicity
- Flexibility
- Reliability

Time Share Bus - Disadvantage

- Performance limited by bus cycle time
- Each processor should have local cache
  - Reduce number of bus accesses
- Leads to problems with cache coherence
  - Solved in hardware - see later

Operating System Issues

- Simultaneous concurrent processes
- Scheduling
- Synchronization
- Memory management
- Reliability and fault tolerance

Cache Coherence and MESI Protocol

- Problem - multiple copies of same data in different caches
- Can result in an inconsistent view of memory
- Write back policy can lead to inconsistency
- Write through can also give problems unless caches monitor memory traffic

Software Solutions

- Compiler and operating system deal with problem
- Overhead transferred to compile time
- Design complexity transferred from hardware to software
- However, software tends to make conservative decisions
  - Inefficient cache utilization
- Analyze code to determine safe periods for caching shared variables

Hardware Solution

- Cache coherence protocols
- Dynamic recognition of potential problems
- Run time
- More efficient use of cache
- Transparent to programmer
- Directory protocols
- Snoopy protocols

Directory Protocols

- Collect and maintain information about copies of data in cache
- Directory stored in main memory
- Requests are checked against directory
- Appropriate transfers are performed
- Creates central bottleneck
- Effective in large scale systems with complex interconnection schemes

Snoopy Protocols

- Distribute cache coherence responsibility among cache controllers
- Cache recognizes that a line is shared
- Updates announced to other caches
- Suited to bus based multiprocessor
- Increases bus traffic

Write Invalidate

- Multiple readers, one writer
- When a write is required, all other caches of the line are invalidated
- Writing processor then has exclusive (cheap) access until line required by another processor
- Used in Pentium II and PowerPC systems
- State of every line is marked as modified, exclusive, shared or invalid
- MESI

Write Update

- Multiple readers and writers
- Updated word is distributed to all other processors

- Some systems use an adaptive mixture of both solutions

Week 13

Performance Evaluation

OBJECTIVE: TO MEASURE, COMPARE, AND PREDICT THE PERFORMANCE OF COMPUTER SYSTEMS.

IMPORTANCE:

- GUIDES DESIGN DECISIONS.
- HELPS OPTIMIZE EXISTING SYSTEMS.
- BENCHMARKS PERFORMANCE FOR SPECIFIC WORKLOADS.

Key Metrics for Performance Evaluation

1. THROUGHPUT:

DEFINITION: NUMBER OF TASKS A SYSTEM COMPLETES IN A GIVEN TIME.

EXAMPLE: INSTRUCTIONS PER SECOND, TRANSACTIONS PER MINUTE.

PERFORMANCE = (1 / EXECUTION TIME)

(PERFORMANCE OF A / PERFORMANCE OF B)

(EXECUTION TIME OF B / EXECUTION

Key Metrics for Performance Evaluation

EXAMPLE – MACHINE A RUNS A PROGRAM IN 100 SECONDS, MACHINE B RUNS THE SAME PROGRAM IN 125 SECONDS

(PERFORMANCE OF A / PERFORMANCE OF B)

= (EXECUTION TIME OF B / EXECUTION TIME OF A)

= 125 / 100 = 1.25

Key Metrics for Performance Evaluation

THE TIME TO EXECUTE A GIVEN PROGRAM CAN BE COMPUTED AS:

EXECUTION TIME = CPU CLOCK CYCLES X CLOCK CYCLE TIME

EXECUTION TIME = CPU CLOCK CYCLES / CLOCK RATE

Key Metrics for Performance Evaluation

CPU CLOCK CYCLES

= (NO. OF INSTRUCTIONS / PROGRAM) X (CLOCK CYCLES / INSTRUCTION)

= INSTRUCTION COUNT X CPI

SO,

EXECUTION TIME

= INSTRUCTION COUNT X CPI X CLOCK CYCLE TIME

= INSTRUCTION COUNT X CPI / CLOCK RATE

Key Metrics for Performance Evaluation

2. LATENCY/RESPONSE TIME:

DEFINITION: TIME TAKEN TO COMPLETE A SINGLE TASK.

EXAMPLE: TIME FROM USER INPUT TO SYSTEM OUTPUT.

Key Metrics for Performance Evaluation

3. CPU PERFORMANCE:

MEASURED BY:

CLOCK SPEED: HIGHER CLOCK SPEEDS MEAN FASTER EXECUTION.

CPI (CYCLES PER INSTRUCTION): AVERAGE NUMBER OF CLOCK CYCLES PER INSTRUCTION.

4. MEMORY PERFORMANCE:

MEASURED BY:

ACCESS TIME: TIME TAKEN TO ACCESS MEMORY.

Key Metrics for Performance Evaluation

5. POWER EFFICIENCY:

- Performance per watt is critical for portable and energy-efficient systems.

6. UTILIZATION:

- Percentage of system resources actively used during operation.

Methods for Performance Evaluation

1. ANALYTICAL MODELING:

- Uses mathematical formulas to predict performance.
- Suitable for early design stages.

2. SIMULATION:

- Simulates hardware or software operations.
- Can be cycle-accurate but time-consuming.

Methods for Performance Evaluation

3. BENCHMARKING:

- Uses standardized tests to compare systems.
- Types:
  - **Synthetic Benchmarks**: Artificial workloads to stress specific components.
  - **Application Benchmarks**: Real-world applications to measure overall performance.

4. MEASUREMENT:

- Directly measures performance using tools and counters during execution.

Benchmarks and Tools

SPEC (STANDARD PERFORMANCE EVALUATION CORPORATION):
BENCHMARKS FOR CPU, MEMORY, AND I/O.

TPCC:
FOR DATABASE TRANSACTION SYSTEMS.

LINPACK:
MEASURES FLOATING-POINT PERFORMANCE (USED IN HPC).

TOOLS:
PERF (LINUX), INTEL VTUNE, GPU PROFILER.

Factors Affecting Performance

HARDWARE:

PROCESSOR SPEED, NUMBER OF CORES, MEMORY HIERARCHY, AND I/O SUBSYSTEMS.

SOFTWARE:

ALGORITHM EFFICIENCY, COMPILER OPTIMIZATIONS, AND OPERATING SYSTEM OVERHEAD.

Factors Affecting Performance

WORKLOAD:

PERFORMANCE VARIES WITH THE NATURE OF THE TASK (COMPUTE-INTENSIVE VS I/O-INTENSIVE).

CONCURRENCY:

ABILITY TO HANDLE MULTIPLE TASKS SIMULTANEOUSLY.

Case Study: Evaluating Performance

TASK: EVALUATE A MULTI-CORE PROCESSOR’S PERFORMANCE FOR PARALLEL WORKLOADS

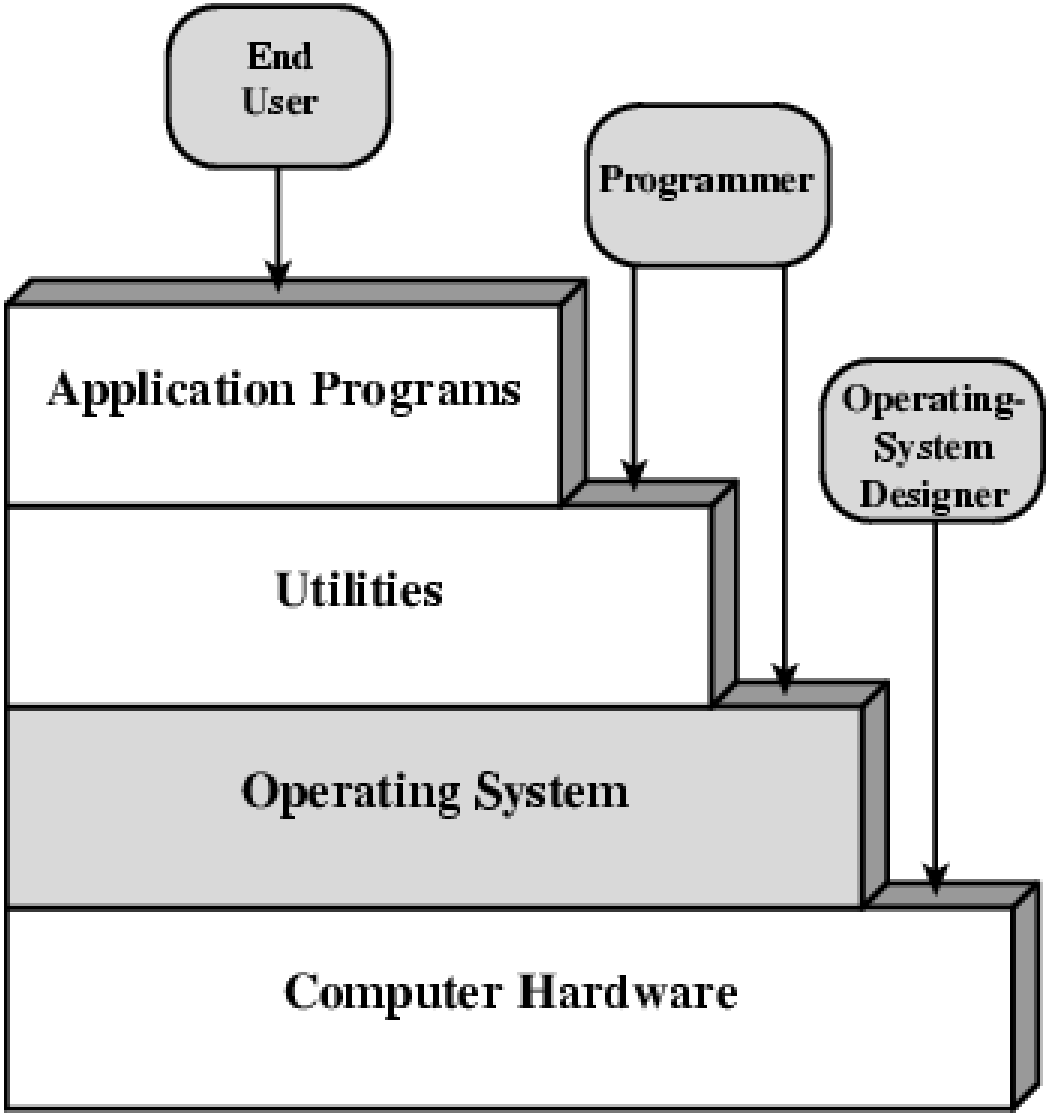
- METRICS: SPEEDUP, THROUGHPUT, AND POWER EFFICIENCY.
- METHODS:
  - Run parallel benchmarks.
  - Measure time for different thread counts.

Week 14

Operating System Support Objectives and Functions

- Convenience
  - Making the computer easier to use
- Efficiency
  - Allowing better use of computer resources

Layers and Views of a Computer System



End User

Programmer

Operating-System Designer

Application Programs

Utilities

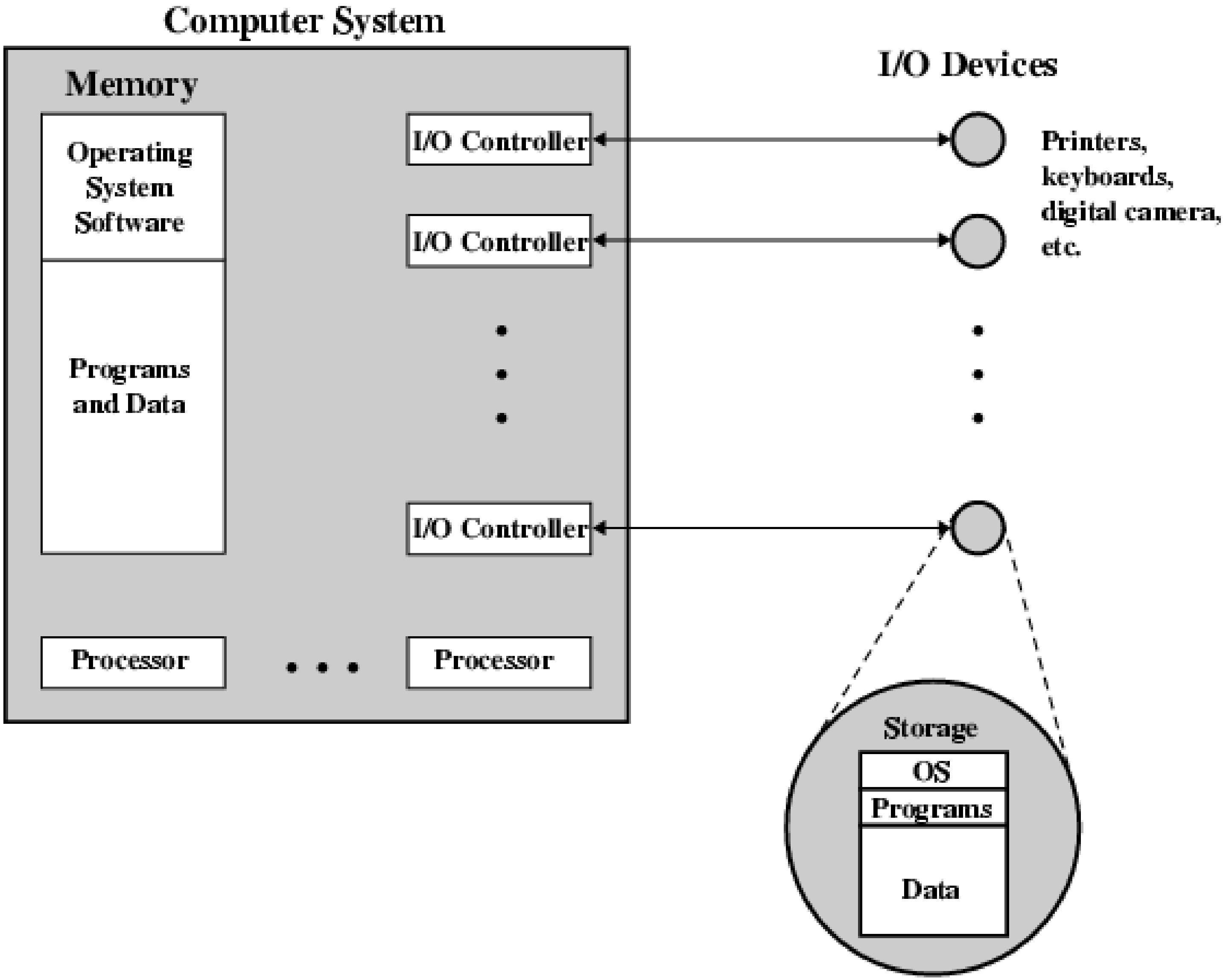
Operating System

Computer Hardware

Operating System Services

- Program creation
- Program execution
- Access to I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting

O/S as a Resource Manager



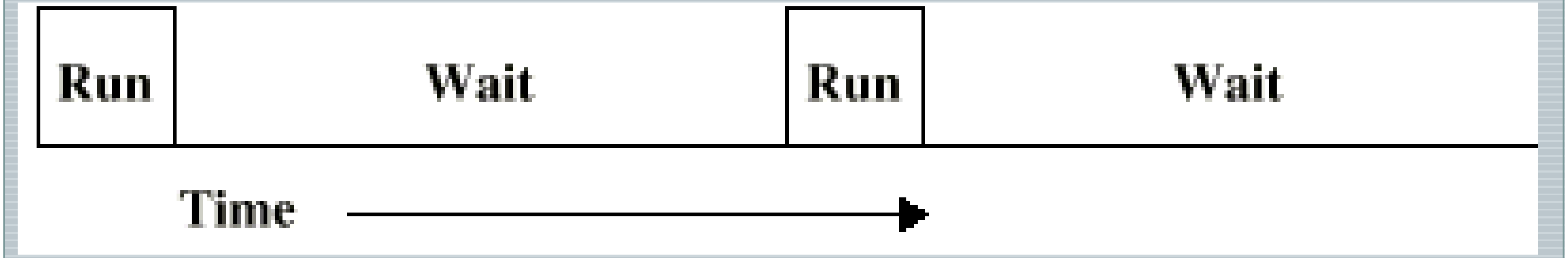
Types of Operating System

- Interactive
- Batch
- Single program (Uni-programming)
- Multi-programming (Multi-tasking)

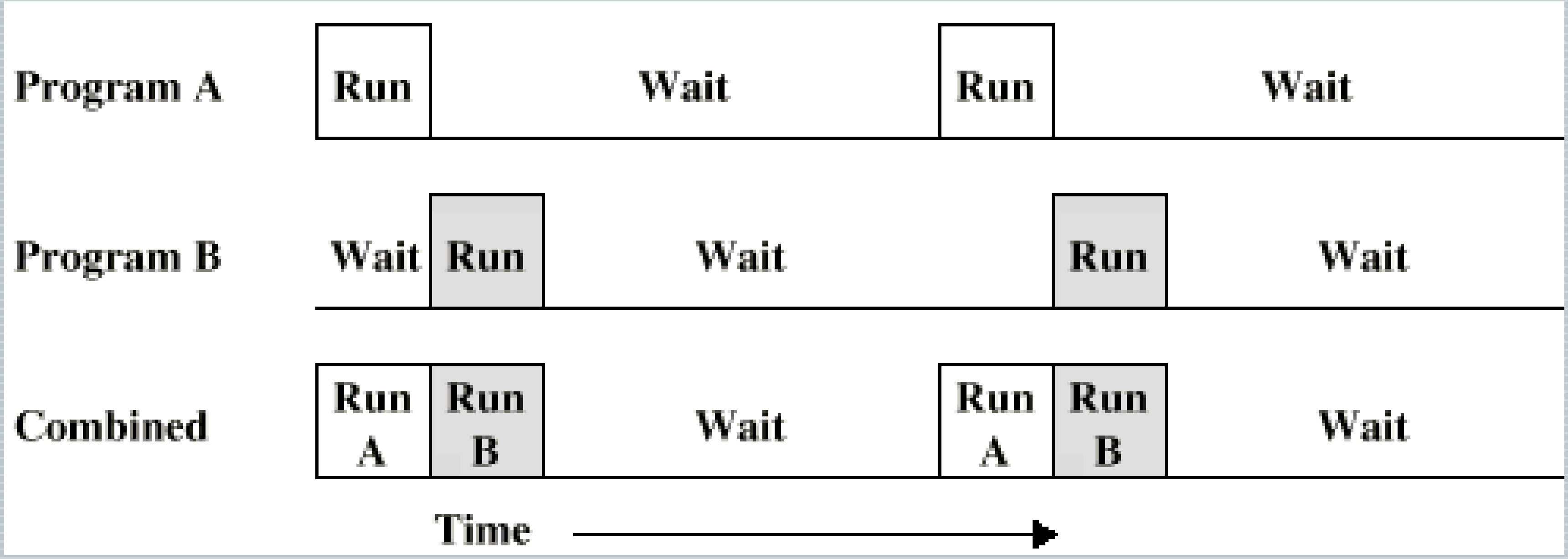
Early Systems

- Late 1940s to mid 1950s
- No Operating System
- Programs interact directly with hardware
- Two main problems:
  - Scheduling
  - Setup time

Single Program



Multi-Programming with Two Programs



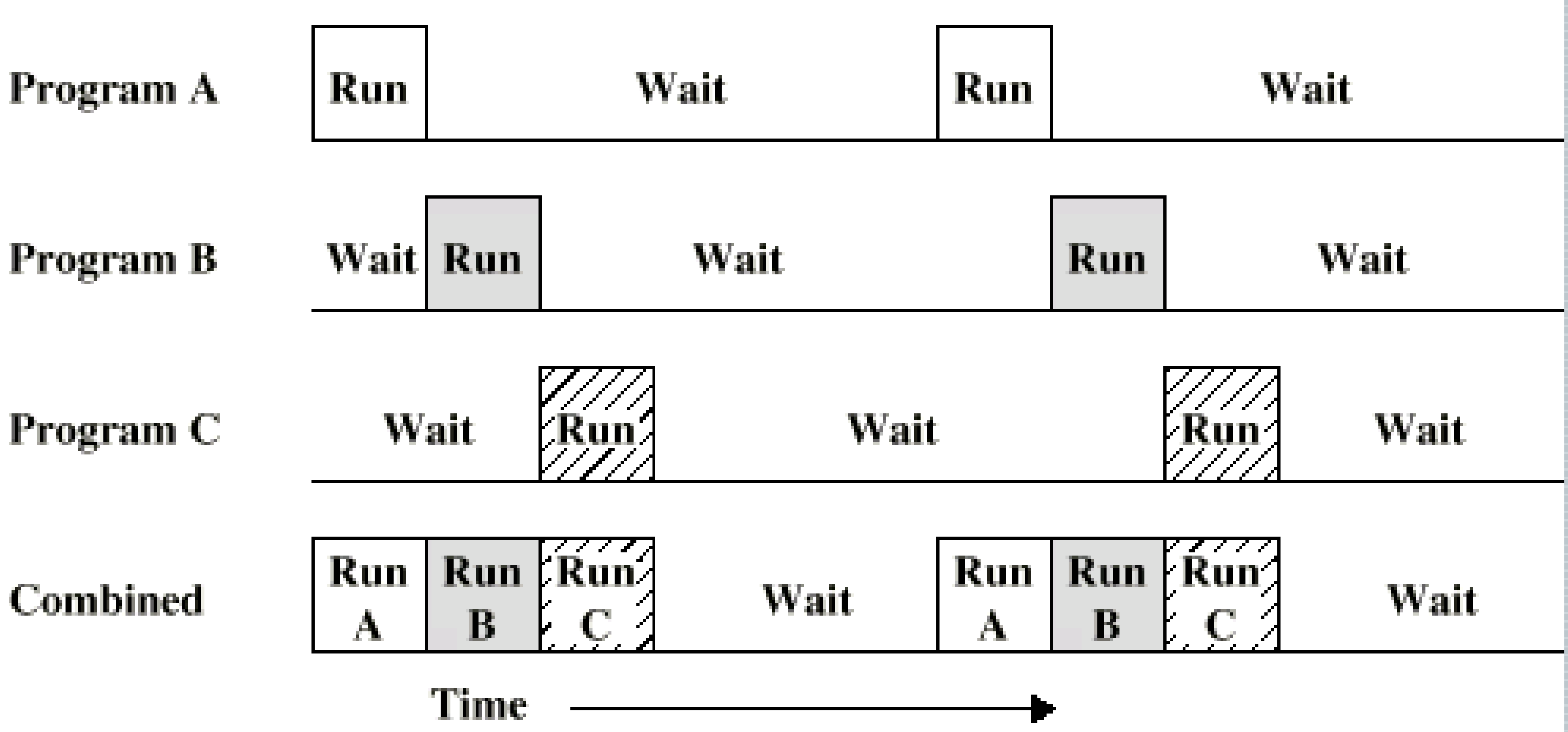
Program A Run Wait Run Wait

Program B Wait Run Wait Run Wait

Combined Run A Run B Wait Run A Run B Wait

Time →

Multi-Programming with Three Programs



Program A: Run, Wait, Run, Wait

Program B: Wait, Run, Wait, Run, Wait

Program C: Wait, Run, Wait, Run, Wait

Combined: Run A, Run B, Run C, Wait, Run A, Run B, Run C, Wait

Time

Time Sharing Systems

- Allow users to interact directly with the computer
  - i.e. Interactive
- Multi-programming allows a number of users to interact with the computer

Scheduling

- Key to multi-programming
- Long term
- Medium term
- Short term
- I/O

Long Term Scheduling

- Determines which programs are submitted for processing
- i.e. controls the degree of multi-programming
- Once submitted, a job becomes a process for the short term scheduler
- (or it becomes a swapped out job for the medium term scheduler)

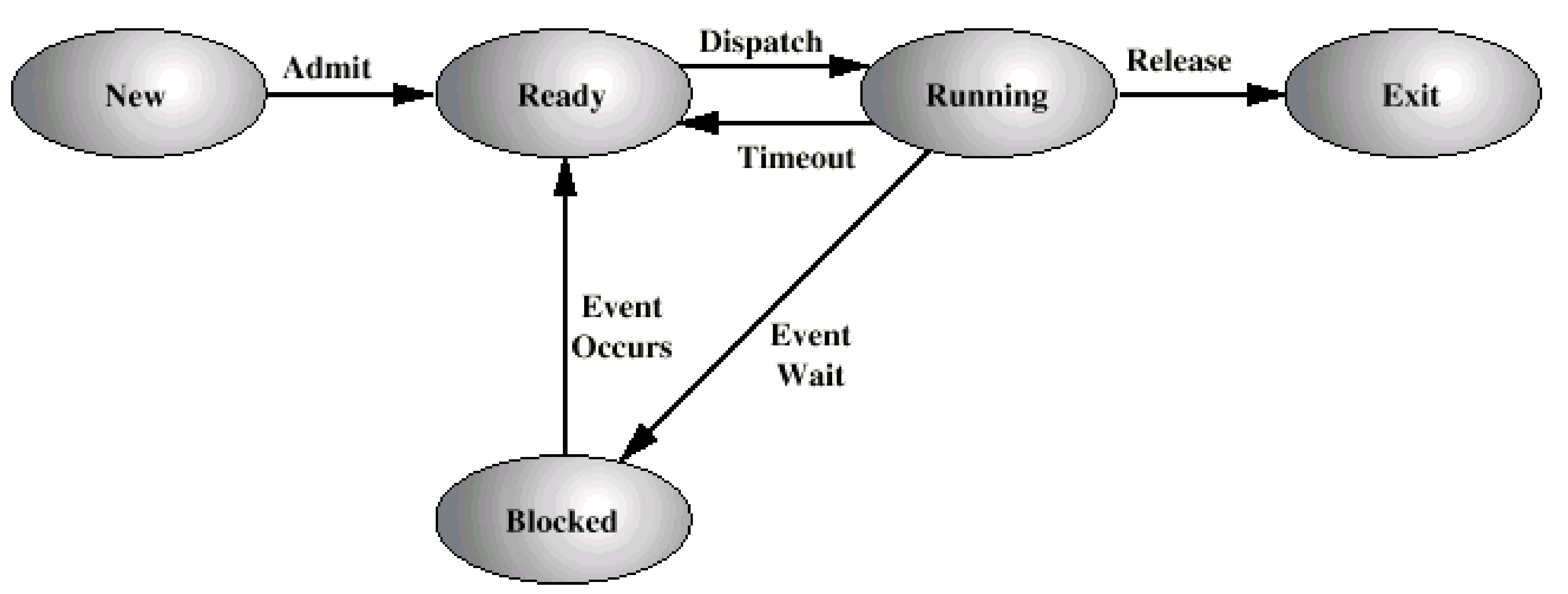
Medium Term Scheduling

- Part of the swapping function (later…)
- Usually based on the need to manage multi-programming
- If no virtual memory, memory management is also an issue

Short Term Scheduler

- Dispatcher
- Fine grained decisions of which job to execute next
- i.e. which job actually gets to use the processor in the next time slot

Five State Process Model

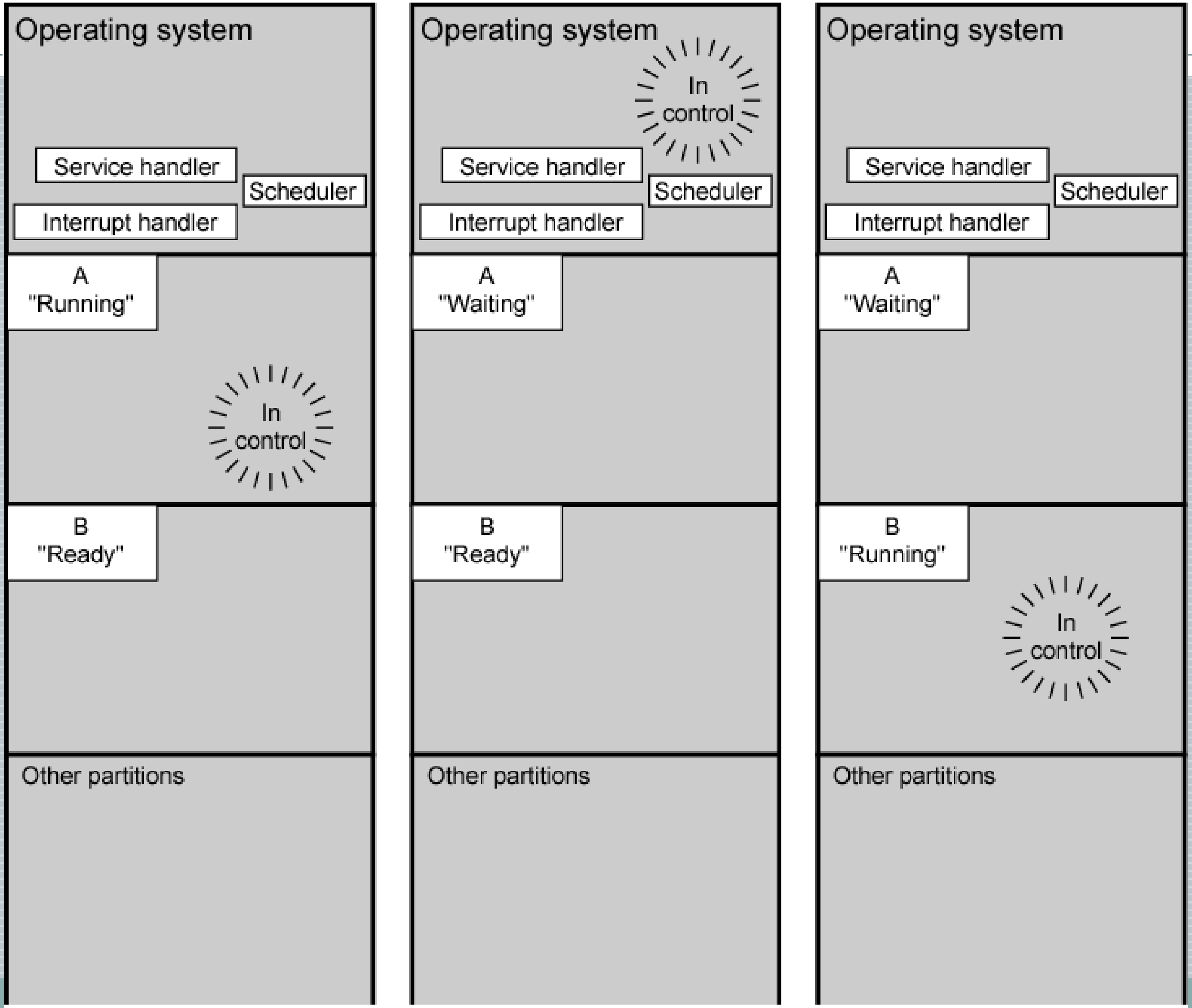


Process Control Block

- Identifier
- State
- Priority
- Program counter
- Memory pointers
- Context data
- I/O status
- Accounting information

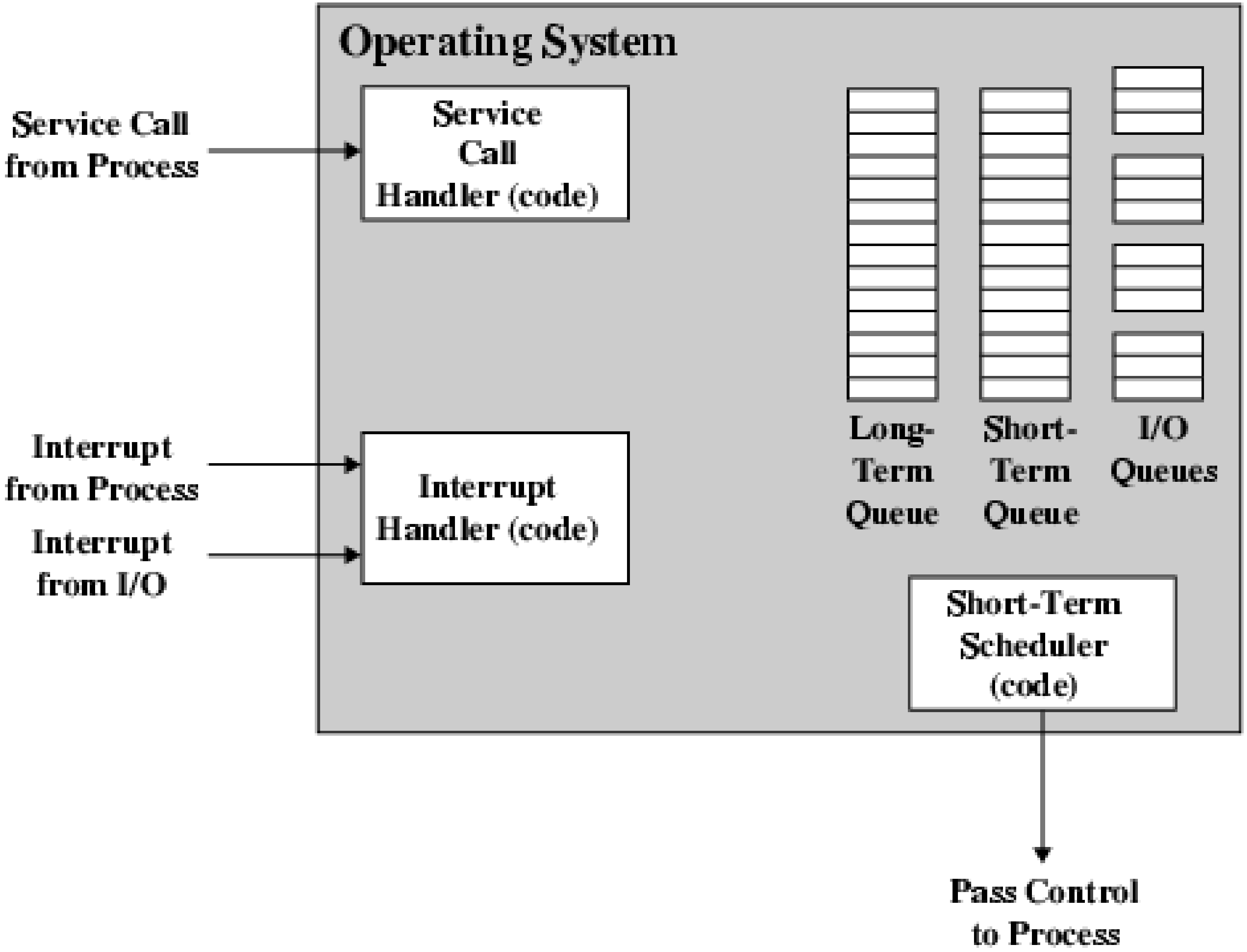
Week 15

Scheduling Example



(a) (b) (c)

Key Elements of O/S



Memory Management

- Uni-program
  - Memory split into two
  - One for Operating System (monitor)
  - One for currently executing program
- Multi-program
  - "User" part is sub-divided and shared among active processes

Swapping

- Problem: I/O is so slow compared with CPU that even in multi-programming system, CPU can be idle most of the time
- Solutions:
  - Increase main memory
    - Expensive
    - Leads to larger programs
  - Swapping

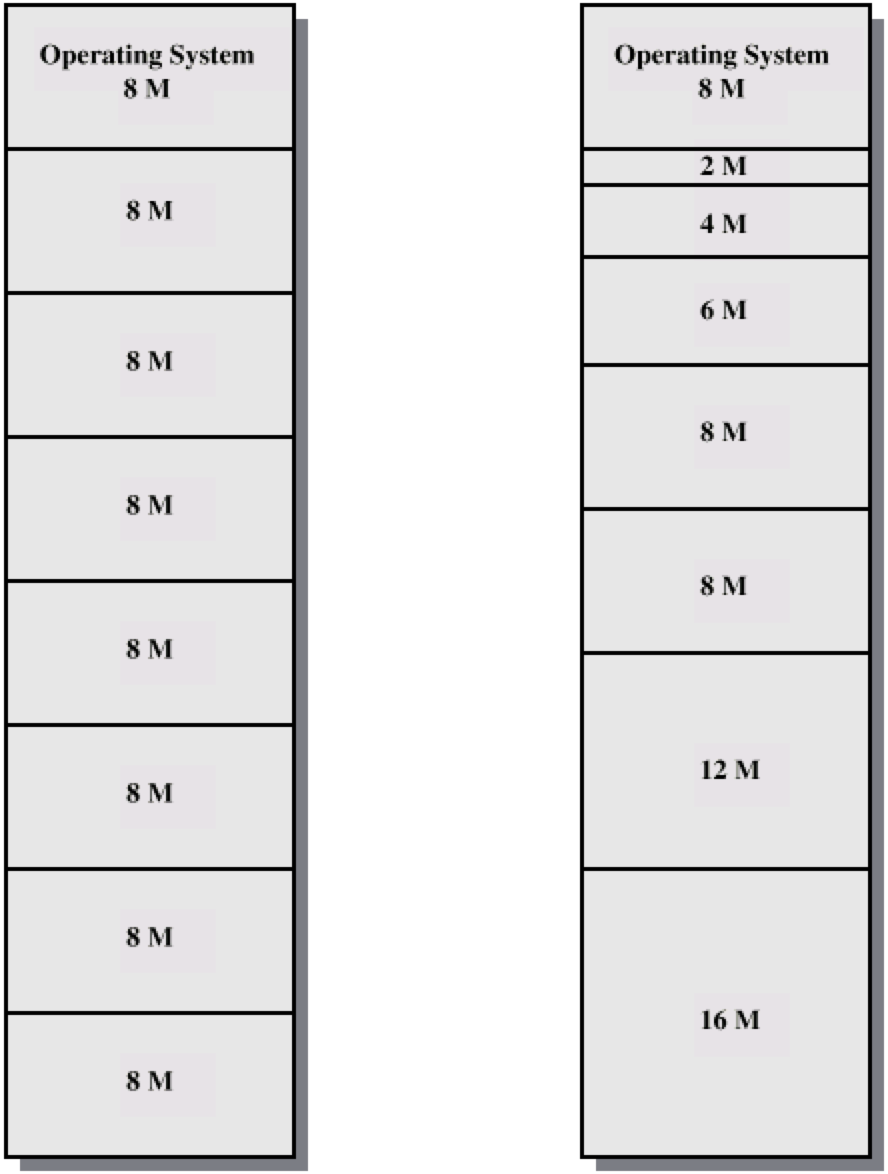
What is Swapping?

- Long term queue of processes stored on disk
- Processes “swapped” in as space becomes available
- As a process completes it is moved out of main memory
- If none of the processes in memory are ready (i.e. all I/O blocked)
    - Swap out a blocked process to intermediate queue
    - Swap in a ready process or a new process
    - But swapping is an I/O process...

Partitioning

- Splitting memory into sections to allocate to processes (including Operating System)
- Fixed-sized partitions
  - May not be equal size
  - Process is fitted into smallest hole that will take it (best fit)
  - Some wasted memory
  - Leads to variable sized partitions

Fixed



(a) Equal-size partitions

(b) Unequal-size partitions

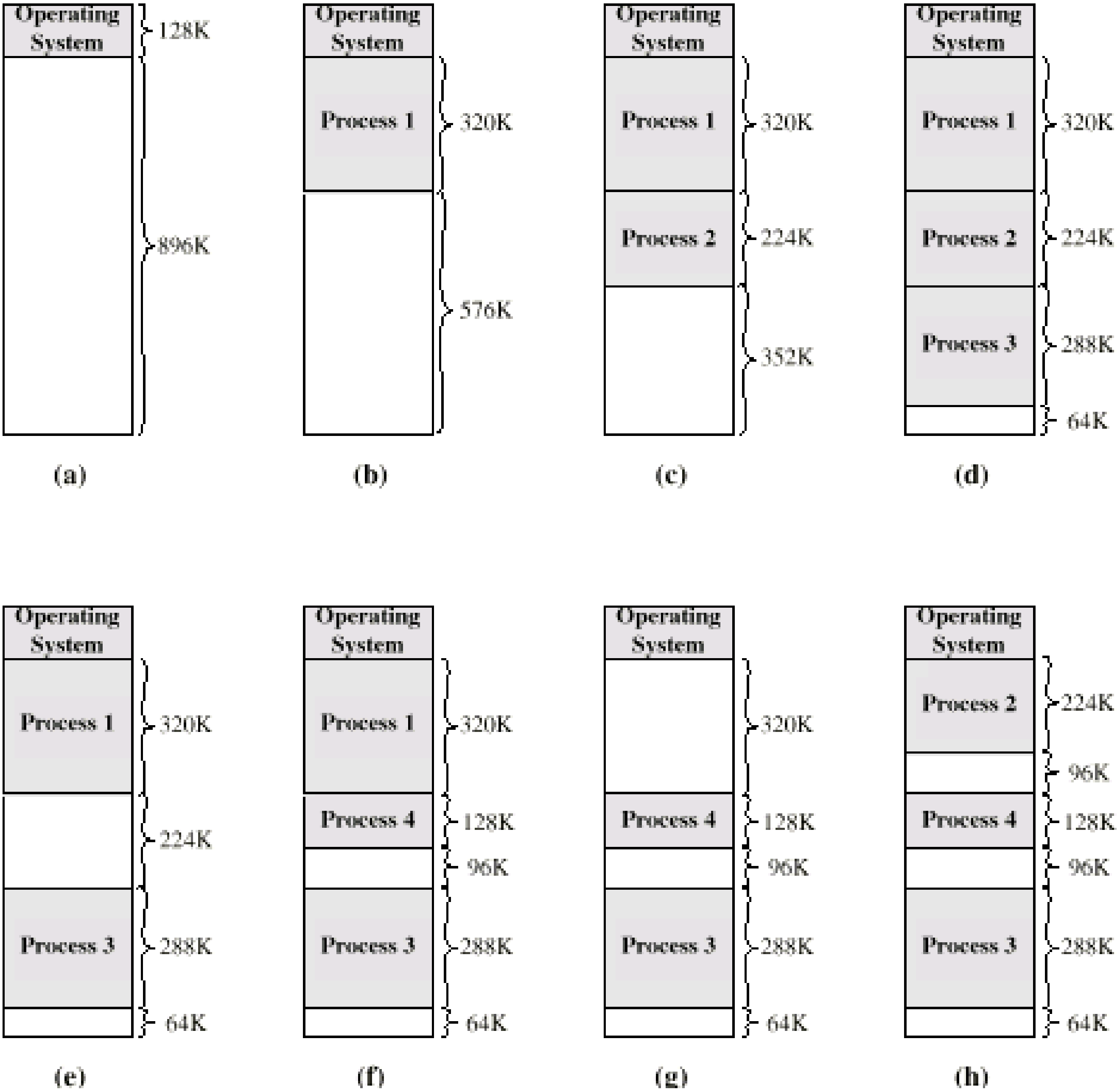
Variable Sized Partitions (1)

- Allocate exactly the required memory to a process
- This leads to a hole at the end of memory, too small to use
  - Only one small hole - less waste
- When all processes are blocked, swap out a process and bring in another
- New process may be smaller than swapped out process
- Another hole

Variable Sized Partitions (2)

- Eventually have lots of holes (fragmentation)
- Solutions:
  - Coalesce - Join adjacent holes into one large hole
  - Compaction - From time to time go through memory and move all hole into one free block (c.f. disk de-fragmentation)

Effect of Dynamic Partitioning



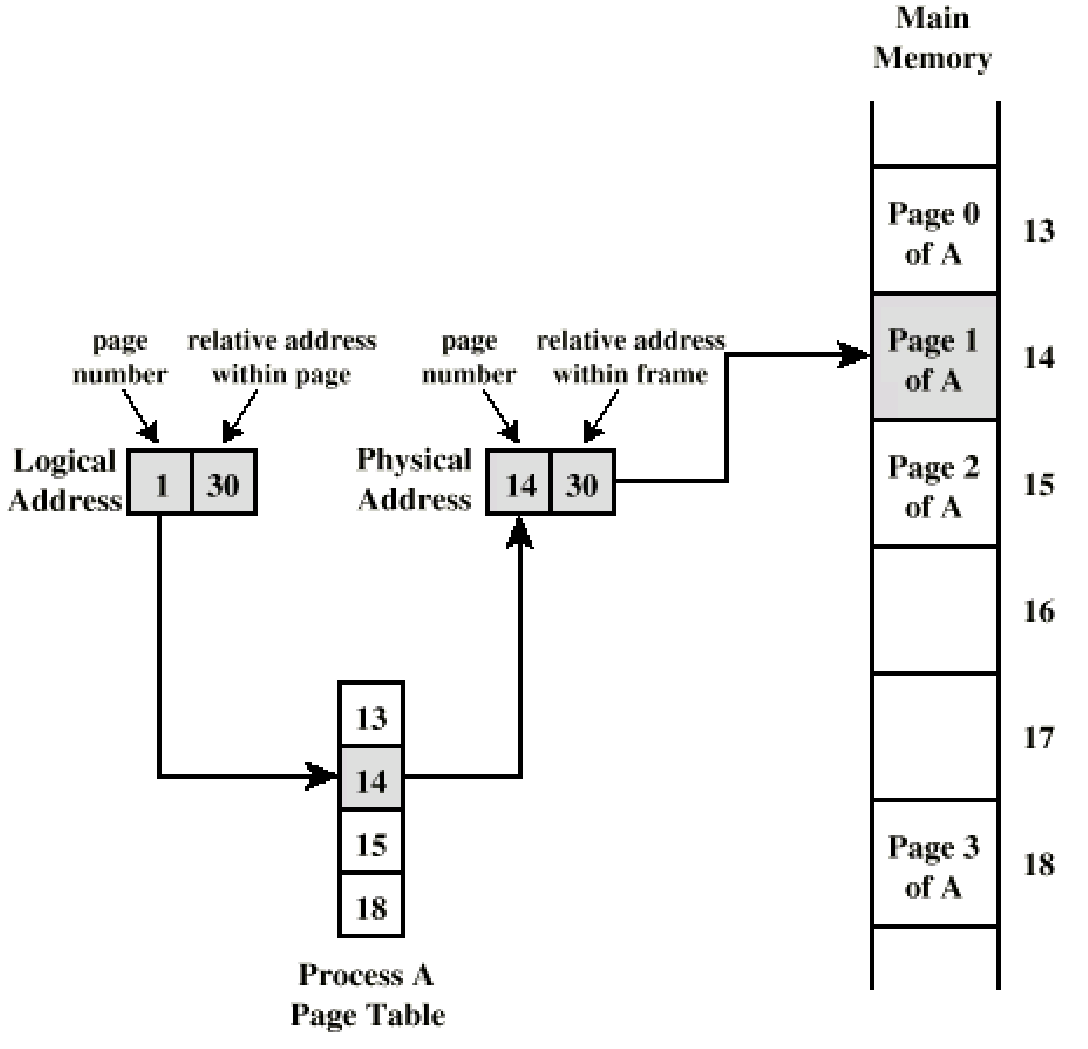
Relocation

- No guarantee that process will load into the same place in memory
- Instructions contain addresses
  - Locations of data
  - Addresses for instructions (branching)
- Logical address - relative to beginning of program
- Physical address - actual location in memory (this time)
- Automatic conversion using base address

Paging

- Split memory into equal sized, small chunks -page frames
- Split programs (processes) into equal sized small chunks - pages
- Allocate the required number page frames to a process
- Operating System maintains list of free frames
- A process does not require contiguous page frames
- Use page table to keep track

Logical and Physical Addresses - Paging



Virtual Memory

- Demand paging
  - Do not require all pages of a process in memory
  - Bring in pages as required
- Page fault
  - Required page is not in memory
  - Operating System must swap in required page
  - May need to swap out a page to make space
  - Select page to throw out based on recent history

Thrashing

- Too many processes in too little memory
- Operating System spends all its time swapping
- Little or no real work is done
- Disk light is on all the time

- Solutions
  - Good page replacement algorithms
  - Reduce number of processes running
  - Fit more memory

Bonus

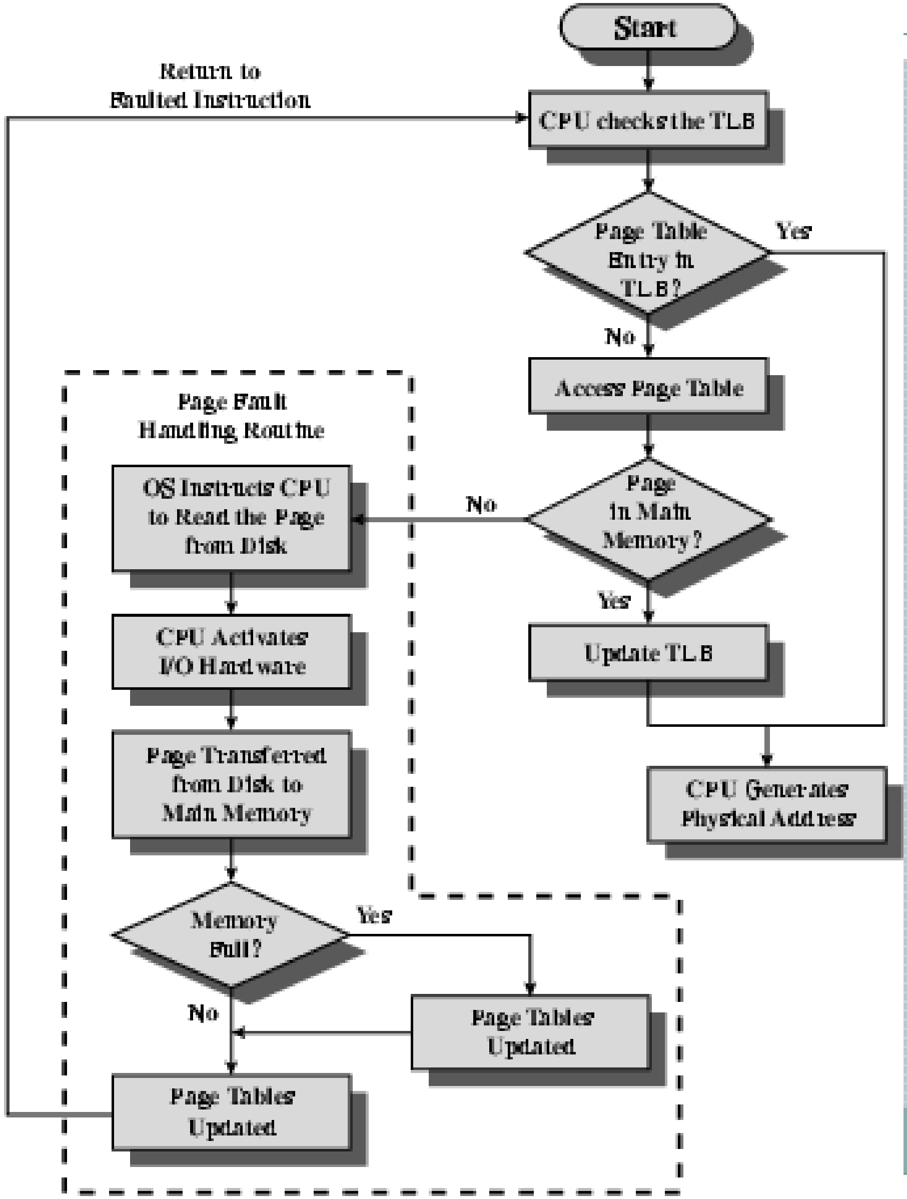
- We do not need all of a process in memory for it to run
- We can swap in pages as required
- So - we can now run processes that are bigger than total memory available!

- Main memory is called real memory
- User/programmer sees much bigger memory - virtual memory

Translation Lookaside Buffer

- Every virtual memory reference causes two physical memory access
  - Fetch page table entry
  - Fetch data
- Use special cache for page table
  - TLB

TLB Operation



Segmentation

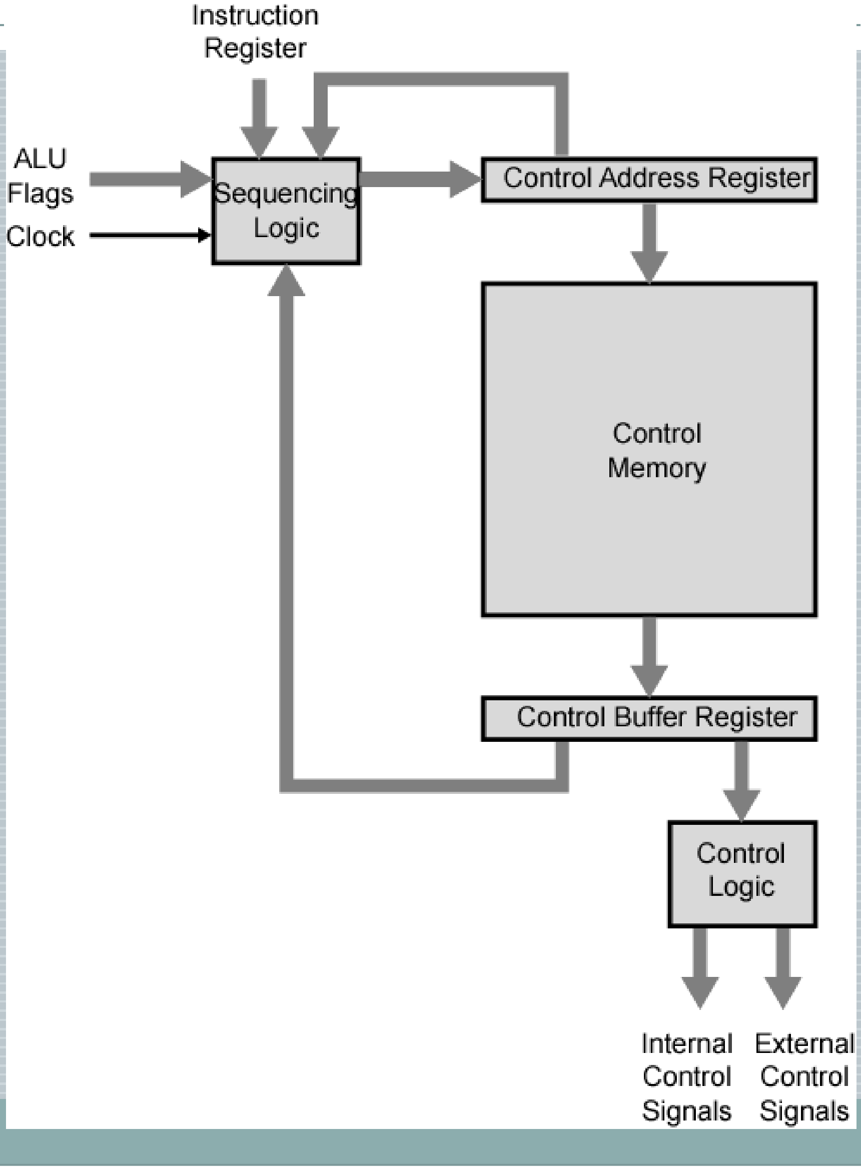
- Paging is not (usually) visible to the programmer
- Segmentation is visible to the programmer
- Usually different segments allocated to program and data
- May be a number of program and data segments

Advantages of Segmentation

- Simplifies handling of growing data structures
- Allows programs to be altered and recompiled independently, without re-linking and re-loading
- Lends itself to sharing among processes
- Lends itself to protection
- Some systems combine segmentation with paging

Week 16

Micro Programmed Controlled Control Unit Organization



Micro-programmed Control

- Use sequences of instructions (see earlier notes) to control complex operations
- Called micro-programming or firmware

Implementation (1)

- All the control unit does is generate a set of control signals
- Each control signal is on or off
- Represent each control signal by a bit
- Have a control word for each micro-operation
- Have a sequence of control words for each machine code instruction
- Add an address to specify the next micro-instruction, depending on conditions

Implementation (2)

- Today’s large microprocessor
  - Many instructions and associated register-level hardware
  - Many control points to be manipulated
- This results in control memory that
  - Contains a large number of words
    - co-responding to the number of instructions to be executed
  - Has a wide word width
    - Due to the large number of control points to be manipulated

Micro-program Word Length

- Based on 3 factors
  - Maximum number of simultaneous micro-operations supported
  - The way control information is represented or encoded
  - The way in which the next micro-instruction address is specified

Micro-instruction Types

- Each micro-instruction specifies single (or few) micro-operations to be performed
  - (*vertical* micro-programming)
- Each micro-instruction specifies many different micro-operations to be performed in parallel
  - (*horizontal* micro-programming)

Vertical Micro-programming

- Width is narrow
- n control signals encoded into $\log_2$ n bits
- Limited ability to express parallelism
- Considerable encoding of control information requires external memory word decoder to identify the exact control line being manipulated

Horizontal Micro-programming

- Wide memory word
- High degree of parallel operations possible
- Little encoding of control information

Compromise

- Divide control signals into disjoint groups
- Implement each group as separate field in memory word
- Supports reasonable levels of parallelism without too much complexity

Control Unit Function

- Sequence login unit issues read command
- Word specified in control address register is read into control buffer register
- Control buffer register contents generates control signals and next address information
- Sequence login loads new address into control buffer register based on next address information from control buffer register and ALU flags

Next Address Decision

- Depending on ALU flags and control buffer register
  - Get next instruction
    - Add 1 to control address register
  - Jump to new routine based on jump microinstruction
    - Load address field of control buffer register into control address register
  - Jump to machine instruction routine
    - Load control address register based on opcode in IR

Advantages and Disadvantages of Microprogramming

- Simplifies design of control unit
  - Cheaper
  - Less error-prone
- Slower

Tasks Done By Microprogrammed Control Unit

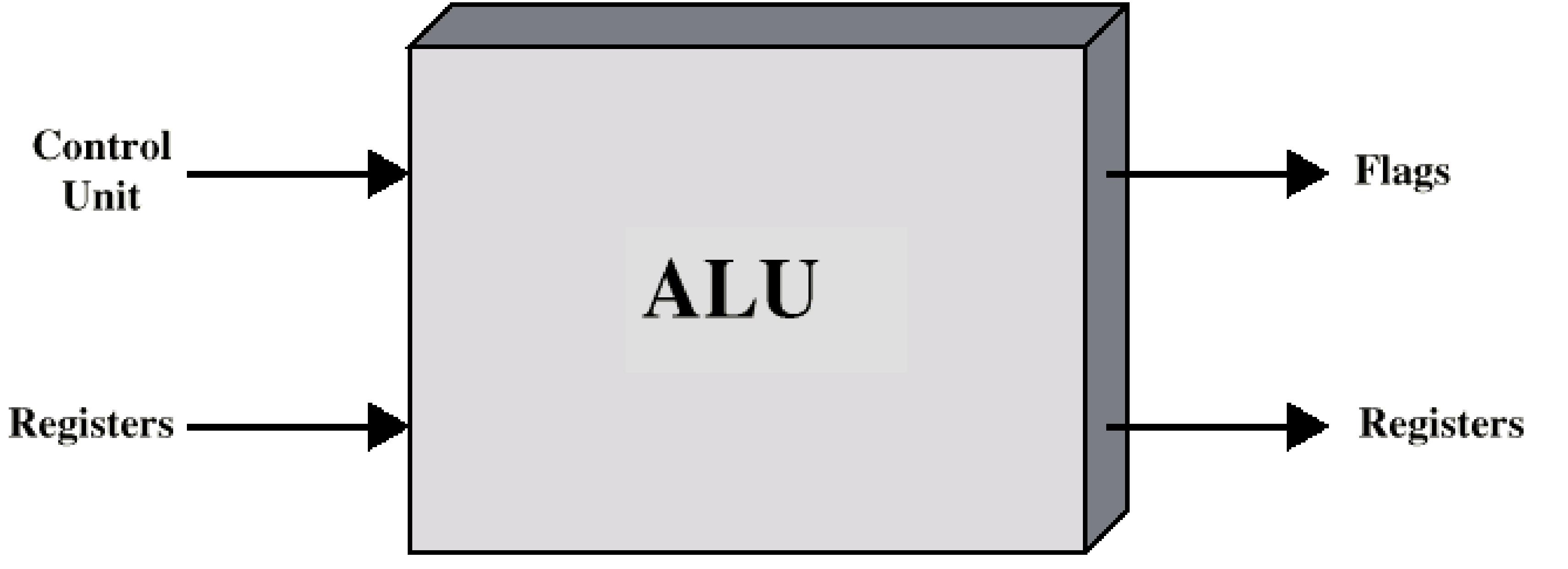
- Microinstruction sequencing
- Microinstruction execution
- Must consider both together

Week 17

Arithmetic & Logic Unit

- Does the calculations
- Everything else in the computer is there to service this unit
- Handles integers
- May handle floating point (real) numbers
- May be separate FPU (maths co-processor)
- May be on chip separate FPU (486DX +)

ALU Inputs and Outputs



Integer Representation

- Only have 0 & 1 to represent everything
- Positive numbers stored in binary
  - e.g. 41=00101001
- No minus sign
- No period
- Sign-Magnitude
- Two’s compliment

Sign-Magnitude

- Left most bit is sign bit
- 0 means positive
- 1 means negative
- +18 = 00010010
- -18 = 10010010
- Problems
  - Need to consider both sign and magnitude in arithmetic
  - Two representations of zero (+0 and -0)

Two’s Compliment

- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000
- -1 = 11111111
- -2 = 11111110
- -3 = 11111101

Benefits

- One representation of zero
- Arithmetic works easily (see later)
- Negating is fairly easy
  - 3 = 00000011
  - Boolean complement gives 11111100
  - Add 1 to LSB 11111101

Negation Special Case 1

- 0 = 00000000
- Bitwise not 11111111
- Add 1 to LSB +1
- Result 1 00000000
- Overflow is ignored, so:
- - 0 = 0 √

Negation Special Case 2

- -128 = 10000000
- bitwise not 01111111
- Add 1 to LSB +1
- Result 10000000
- So:
- -(-128) = -128 X
- Monitor MSB (sign bit)
- It should change during negation

Range of Numbers

- 8 bit 2s compliment
  - +127 = 01111111 = 2^7 -1
  - -128 = 10000000 = -2^7
- 16 bit 2s compliment
  - +32767 = 011111111 11111111 = 2^{15} - 1
  - -32768 = 100000000 00000000 = -2^{15}

Conversion Between Lengths

- Positive number pack with leading zeros
- +18 = 00010010
- +18 = 00000000 00010010
- Negative numbers pack with leading ones
- -18 = 10010010
- -18 = 11111111 10010010
- i.e. pack with MSB (sign bit)

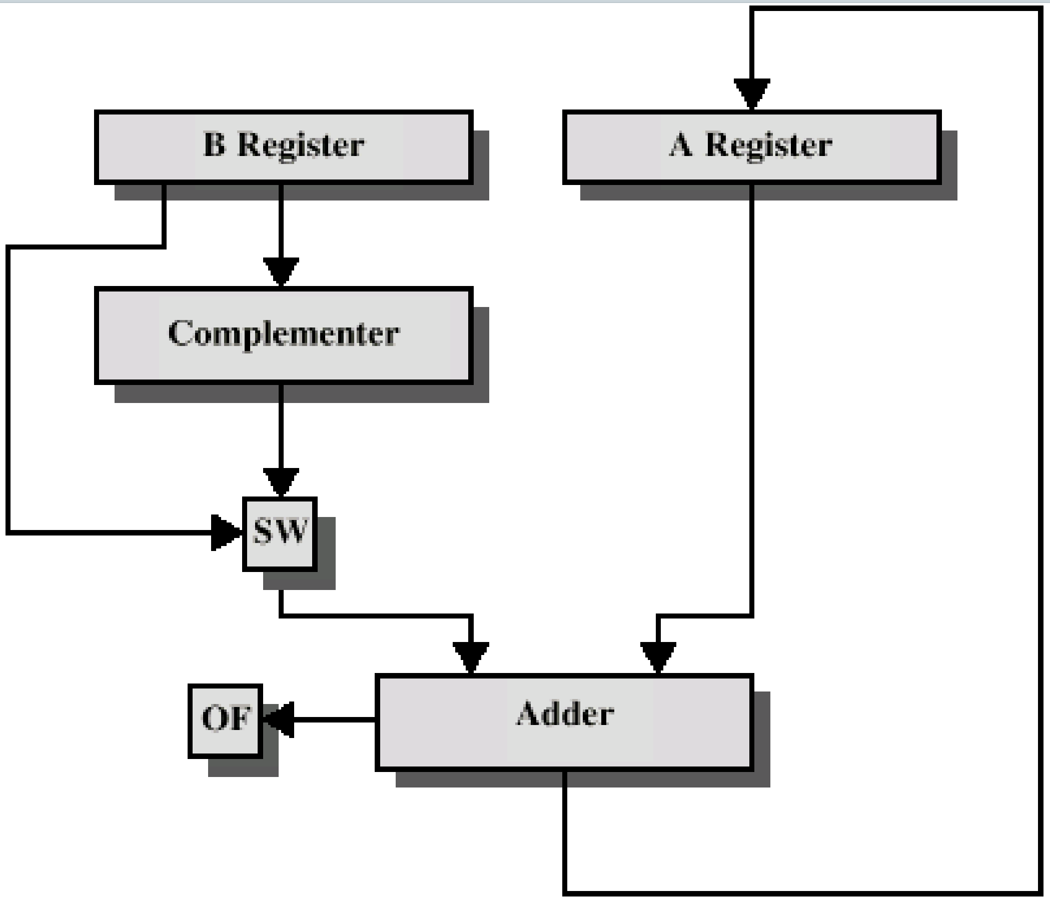
Addition and Subtraction

- Normal binary addition
- Monitor sign bit for overflow

- Take twos compliment of substahend and add to minuend
  - i.e. a - b = a + (-b)

- So we only need addition and complement circuits

Hardware for Addition and Subtraction



OF = overflow bit
SW = Switch (select addition or subtraction)

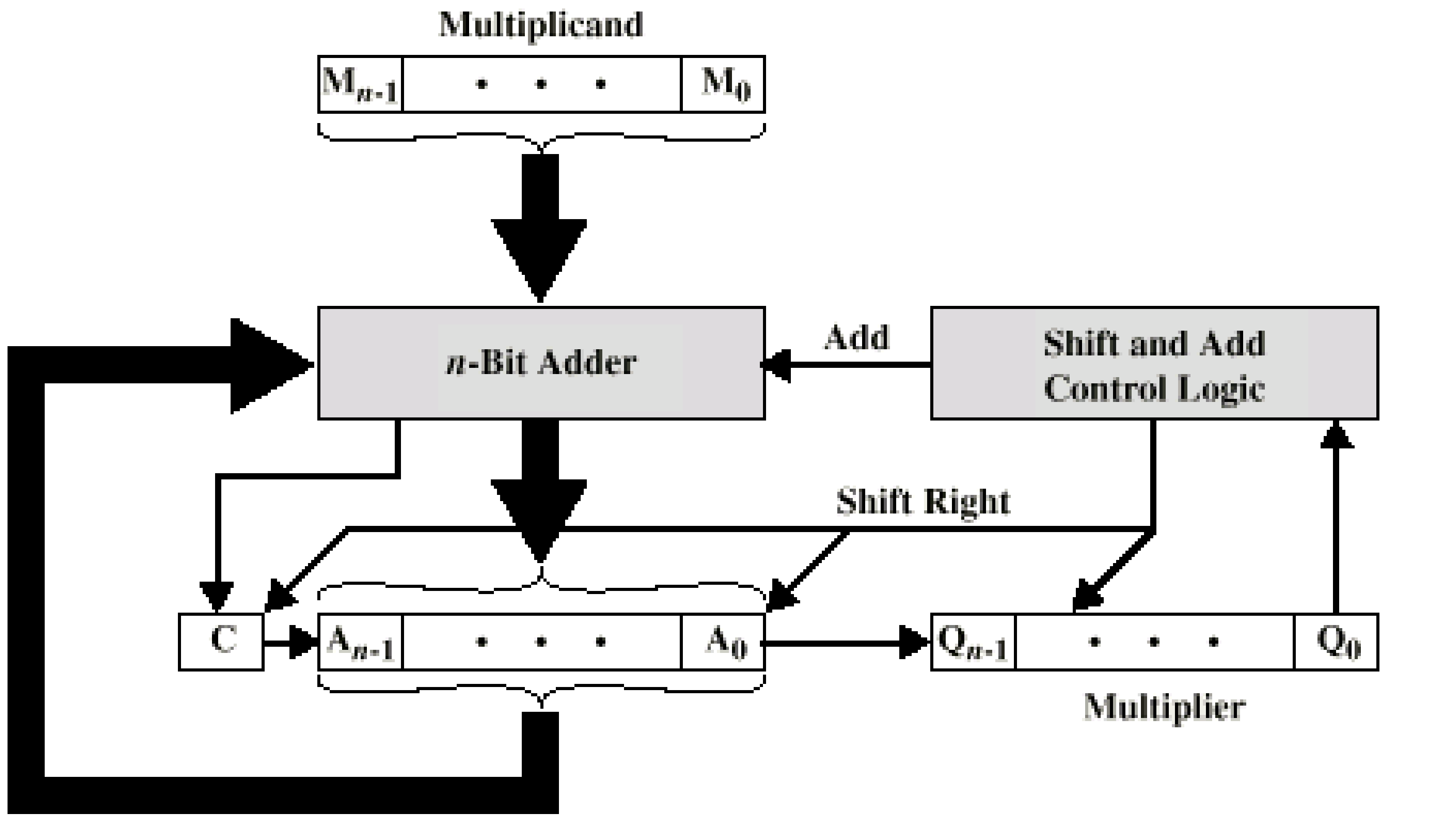
Multiplication

- Complex
- Work out partial product for each digit
- Take care with place value (column)
- Add partial products

Multiplication Example

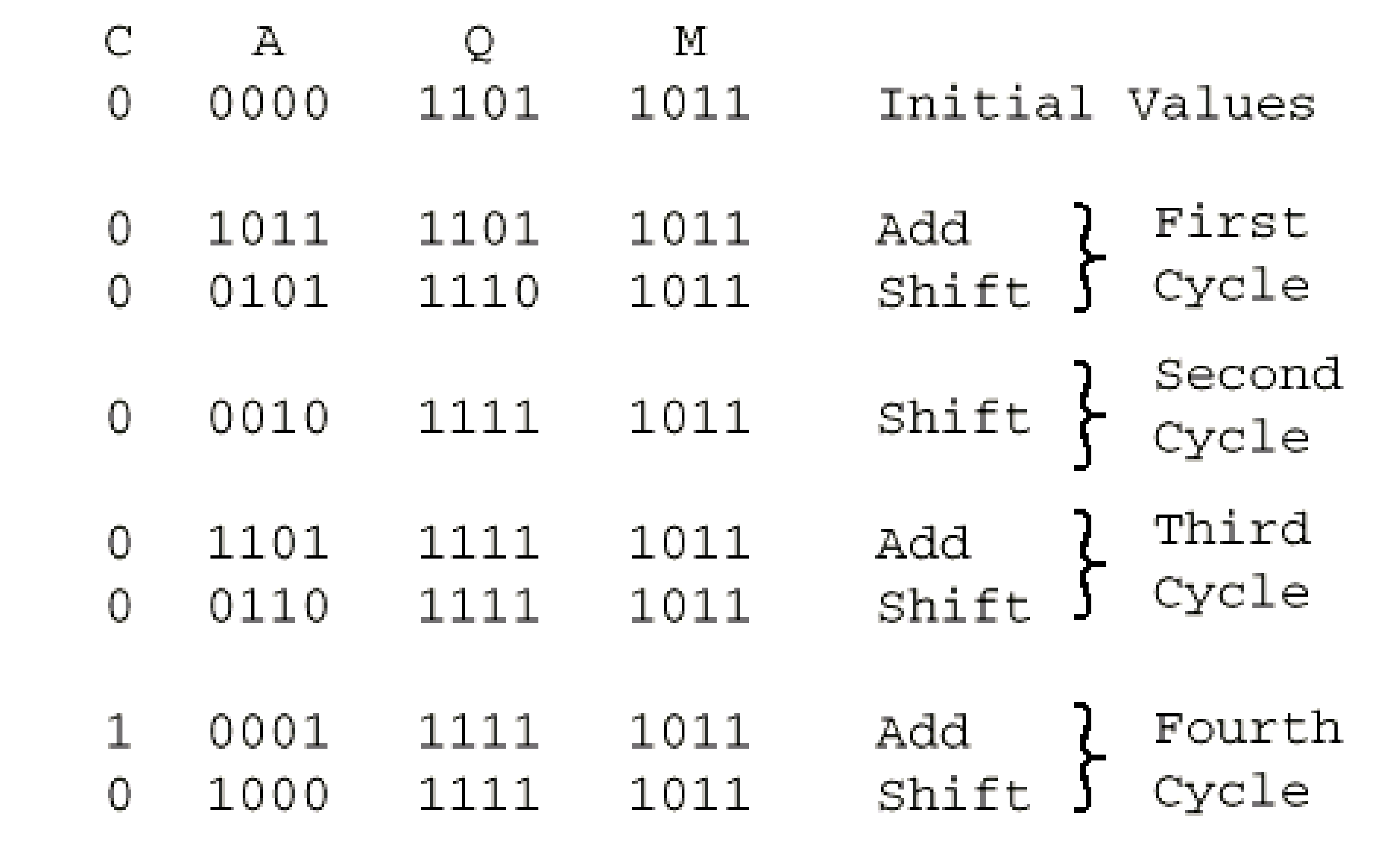
- 1011 Multiplicand (11 dec)
- x 1101 Multiplier (13 dec)
- 1011 Partial products
- 0000 Note: if multiplier bit is 1 copy
- 1011 multiplicand (place value)
- 1011 otherwise zero
- 10001111 Product (143 dec)
- Note: need double length result

Unsigned Binary Multiplication



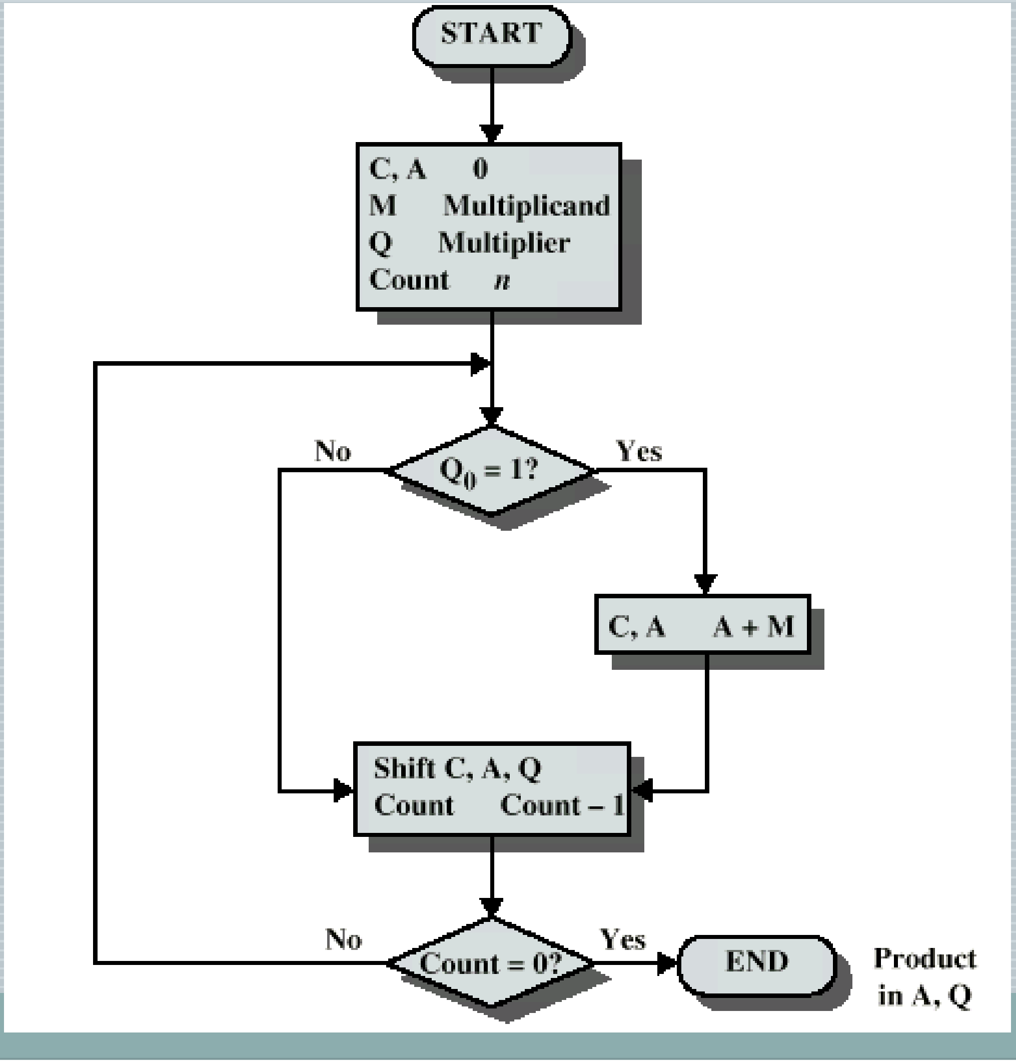
(a) Block Diagram

Execution of Example



C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

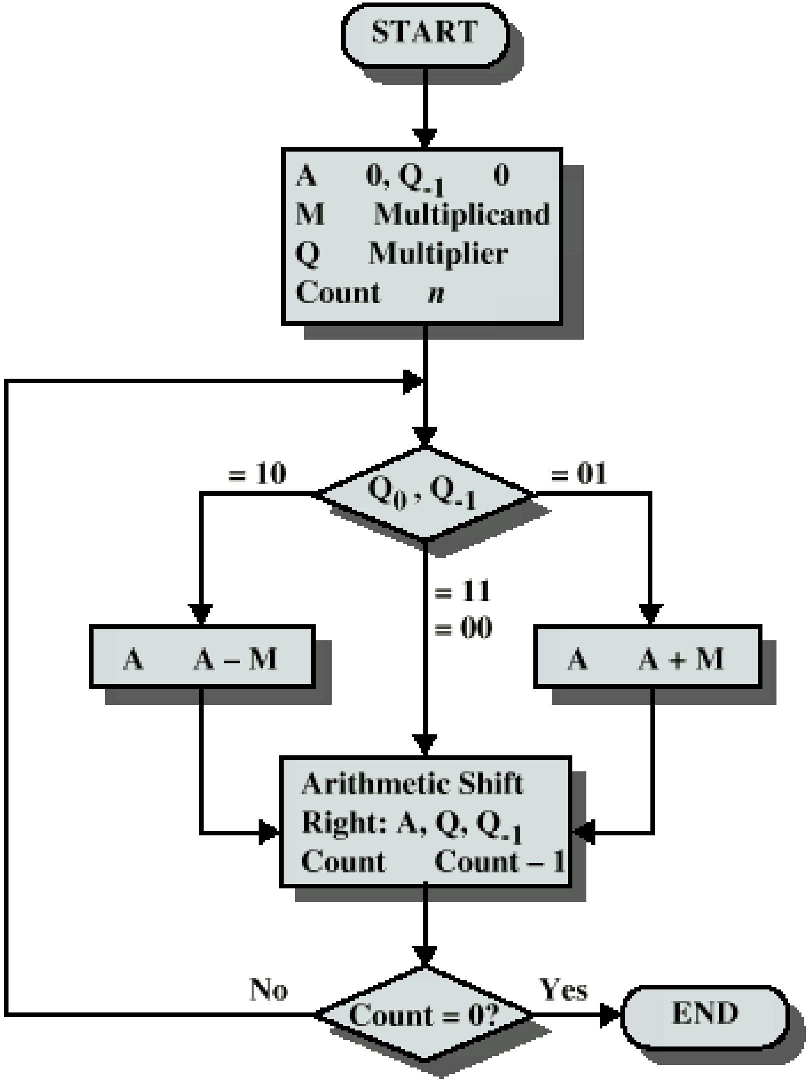
Flowchart for Unsigned Binary Multiplication



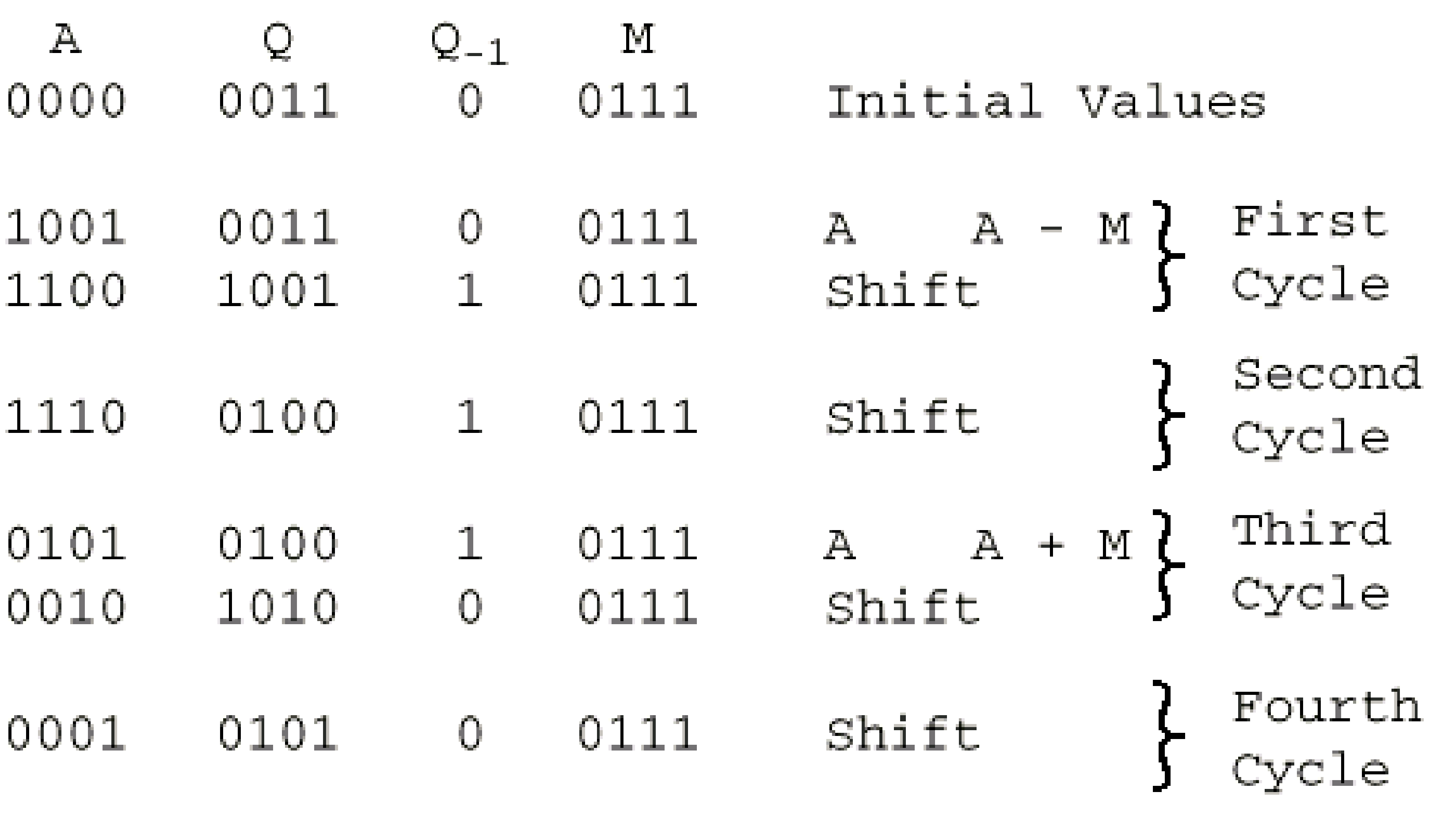
Multiplying Negative Numbers

- This does not work!
- Solution 1
  - Convert to positive if required
  - Multiply as above
  - If signs were different, negate answer
- Solution 2
  - Booth’s algorithm

Booth’s Algorithm



Example of Booth's Algorithm

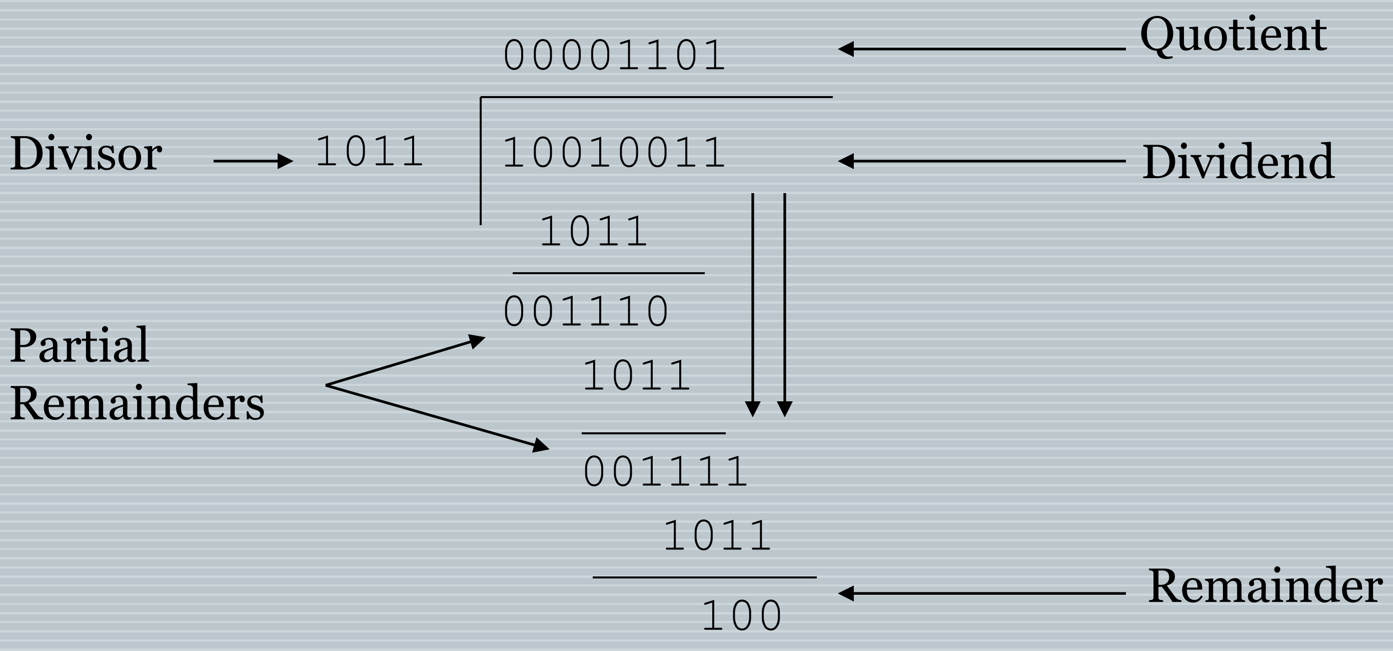


A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

Division

- More complex than multiplication
- Negative numbers are really bad!
- Based on long division

Division of Unsigned Binary Integers



Quotient

00001101

Divisor

1011

10010011

Dividend

1011

001110

Partial Remainders

1011

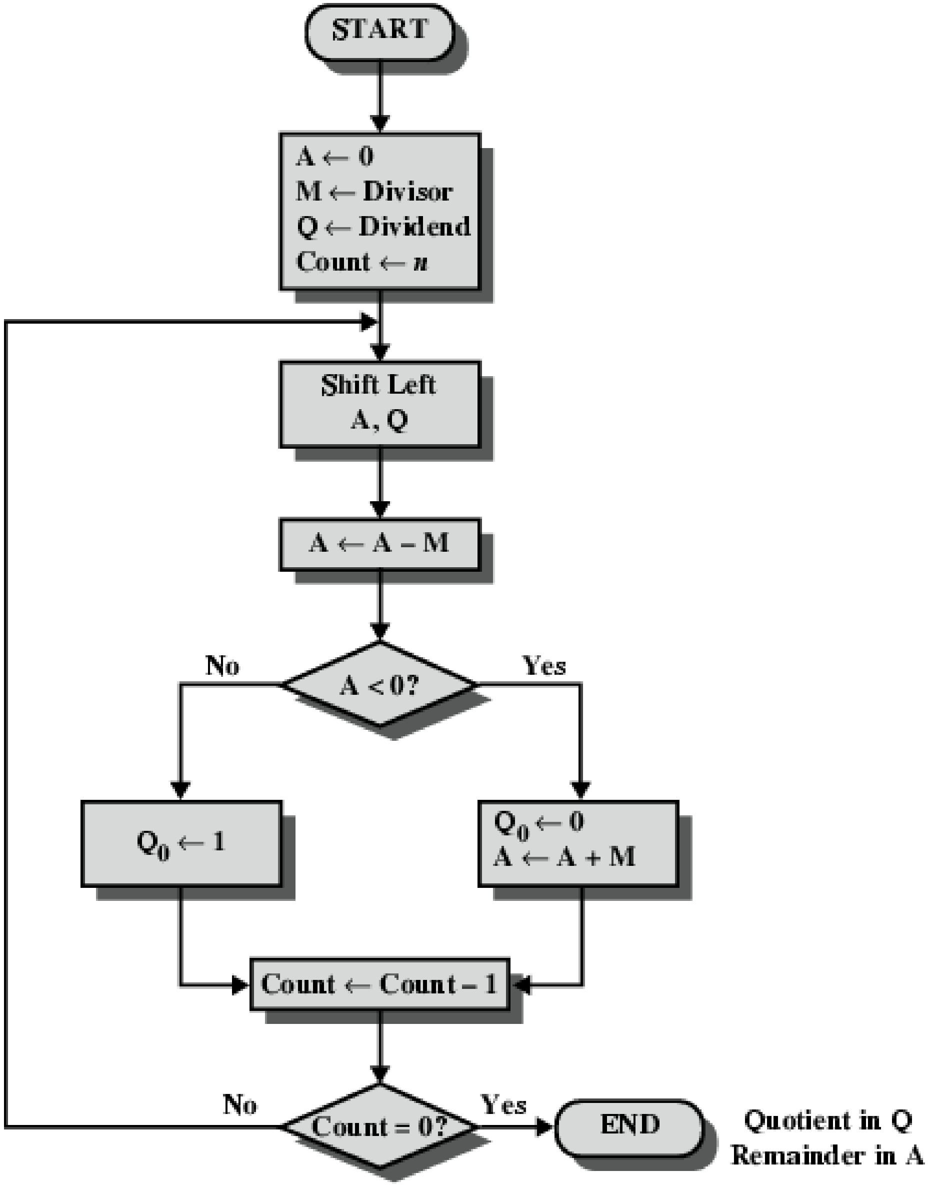
001111

1011

100

Remainder

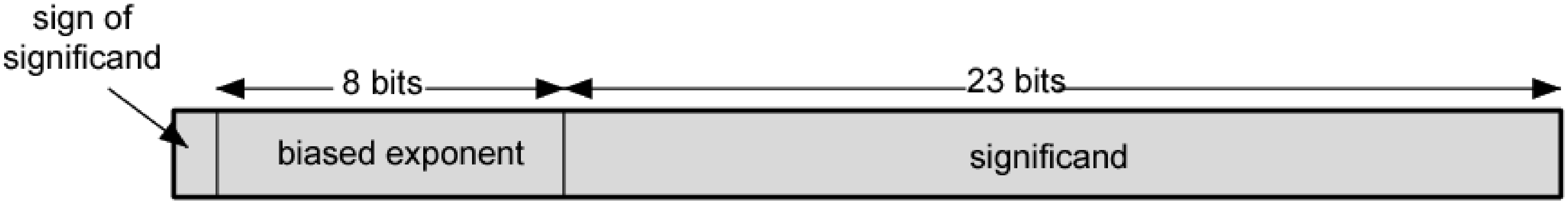
Flowchart for Unsigned Binary Division



Real Numbers

- Numbers with fractions
- Could be done in pure binary
  - 1001.1010 = $2^4 + 2^0 + 2^{-1} + 2^{-3}$ =9.625
- Where is the binary point?
- Fixed?
  - Very limited
- Moving?
  - How do you show where it is?

Floating Point



(a) Format

- $+/-$ .significand x 2^{exponent}
- Misnomer
- Point is actually fixed between sign bit and body of mantissa
- Exponent indicates place value (point position)

Normalization

- FP numbers are usually normalized
- i.e. exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- (c.f. Scientific notation where numbers are normalized to give a single digit before the decimal point
- e.g. 3.123×10^3)