Computer Organization and Design

Chapter 1
Introduction

Architecture & Organization 1

- Architecture is those attributes visible to the programmer
  - Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques.
- Organization is how features are implemented
  - Control signals, interfaces, memory technology.

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.
- Organization differs between different versions.

Structure & Function

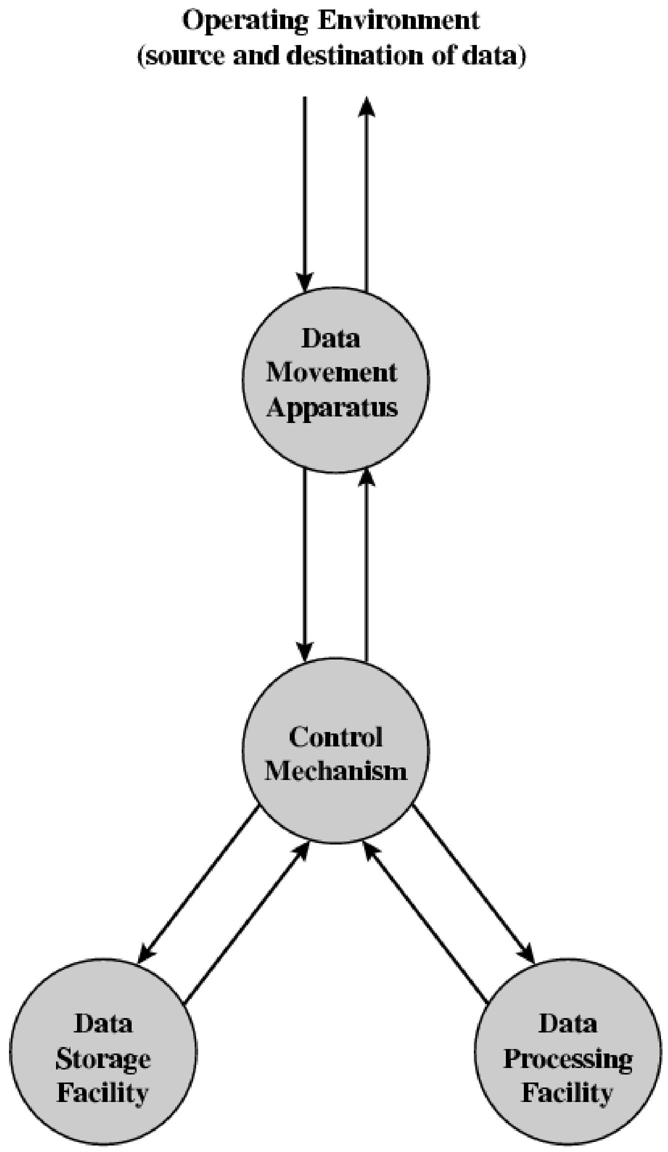
- Structure is the way in which components relate to each other.

- Function is the operation of individual components as part of the structure.

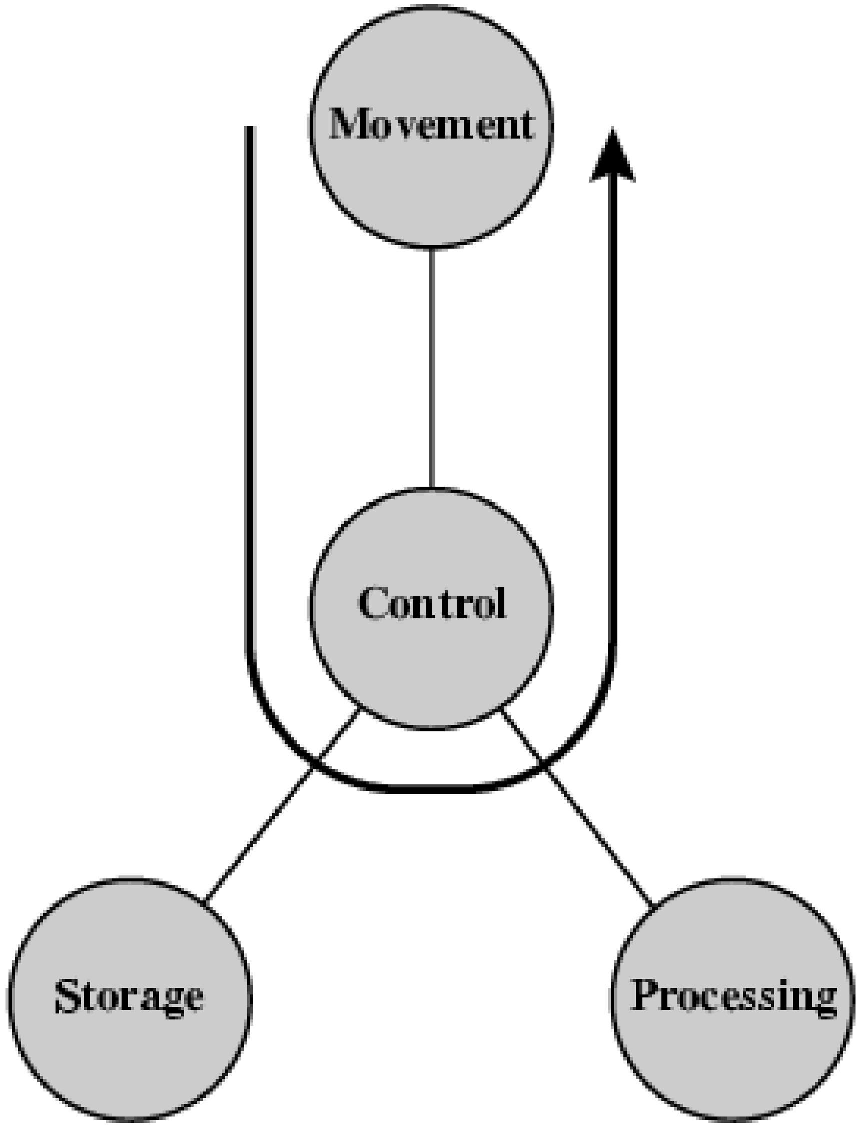
Function

- All computer functions are:
  - Data processing
  - Data storage
  - Data movement
  - Control

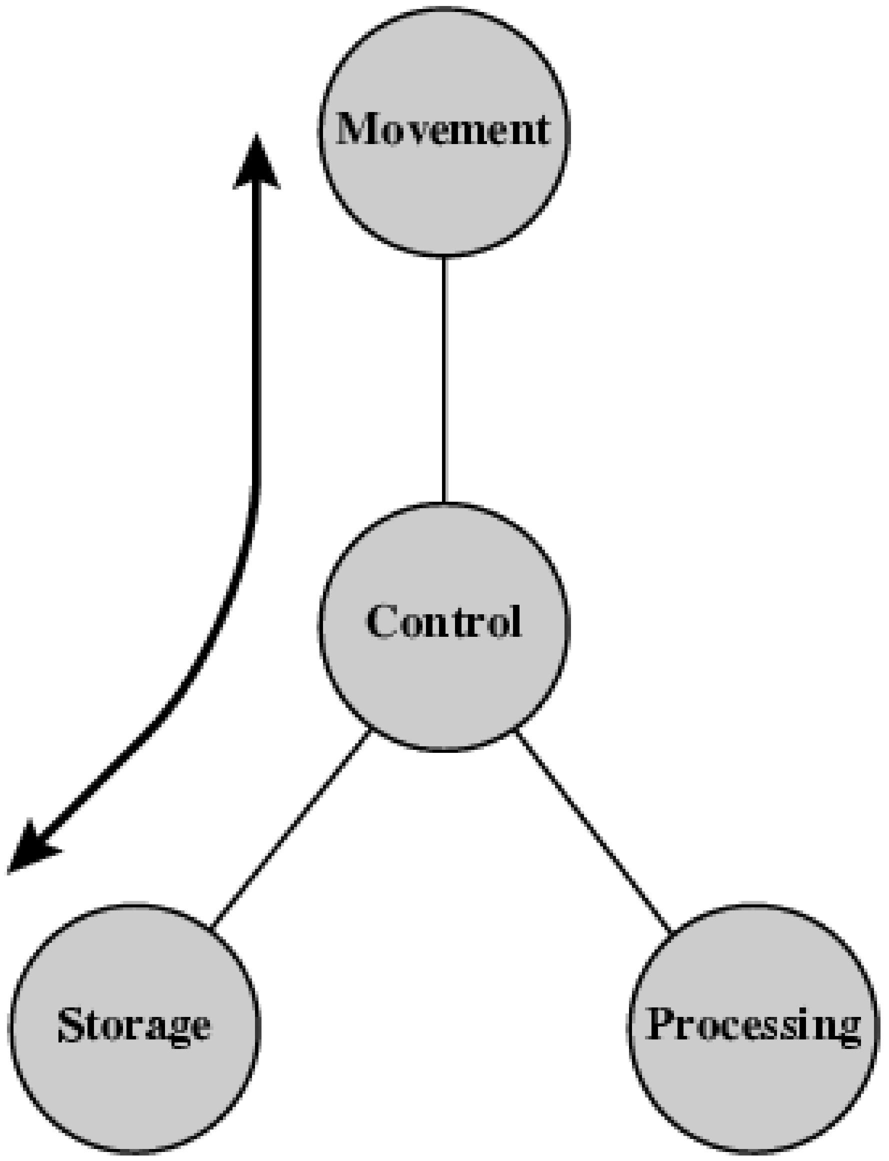
Functional view



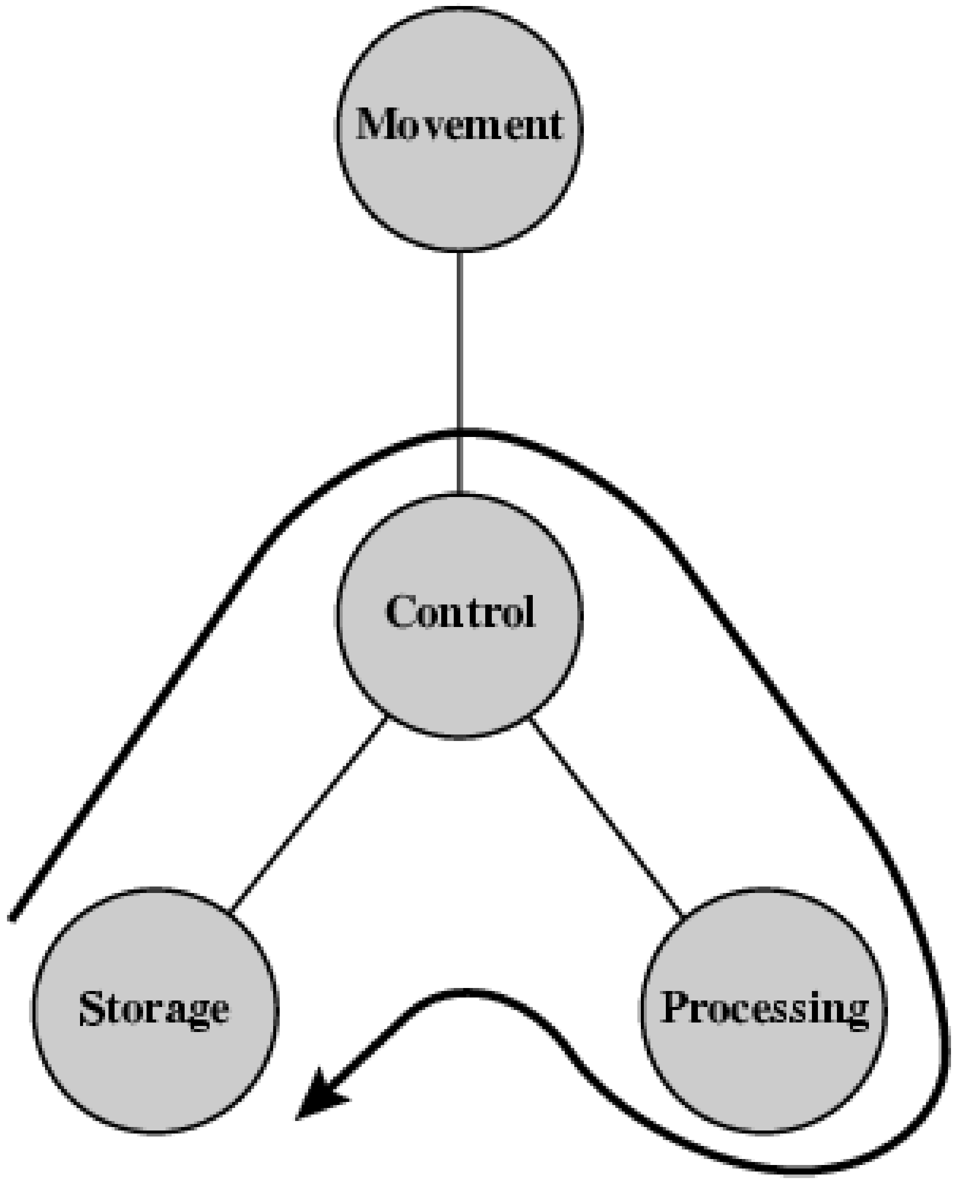
Operations (1) Data movement



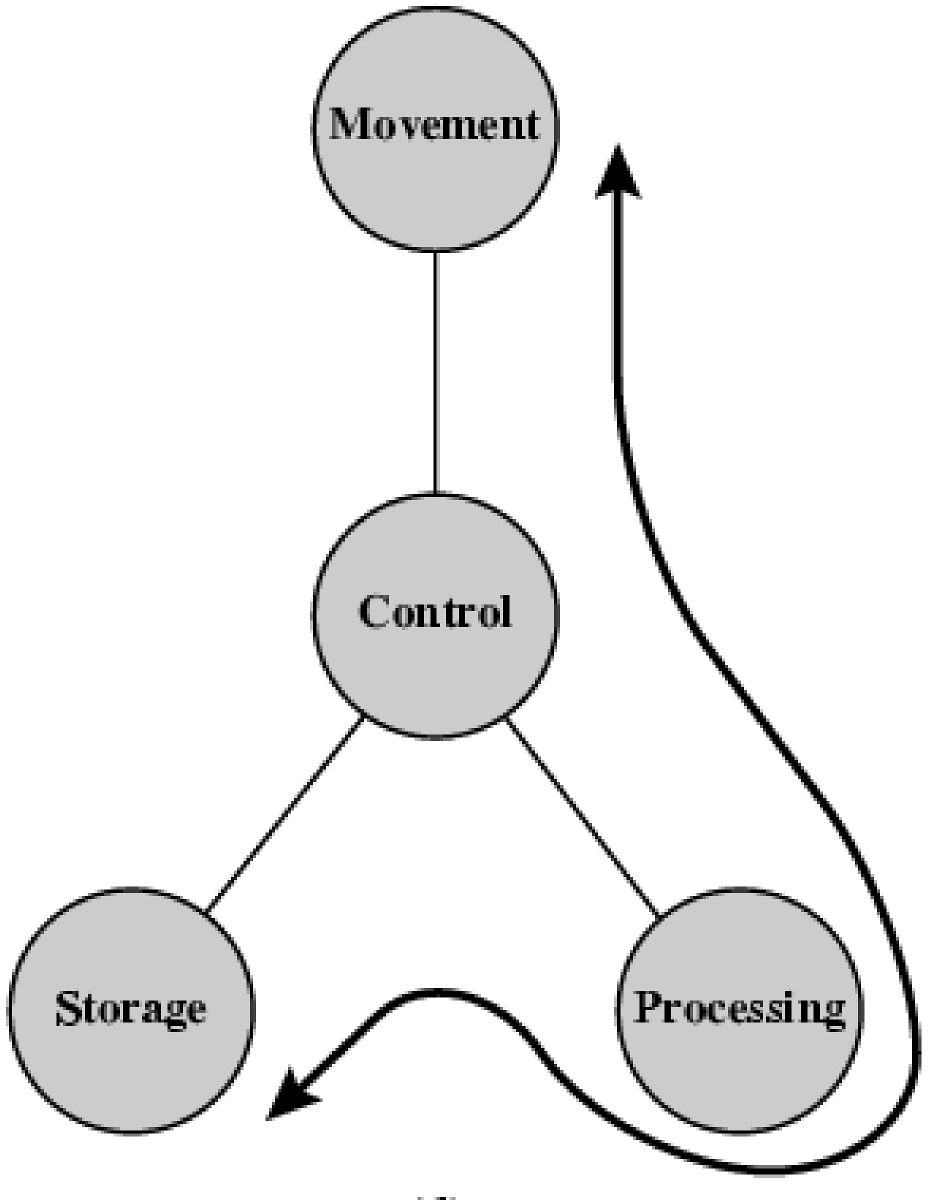
Operations (2) Storage



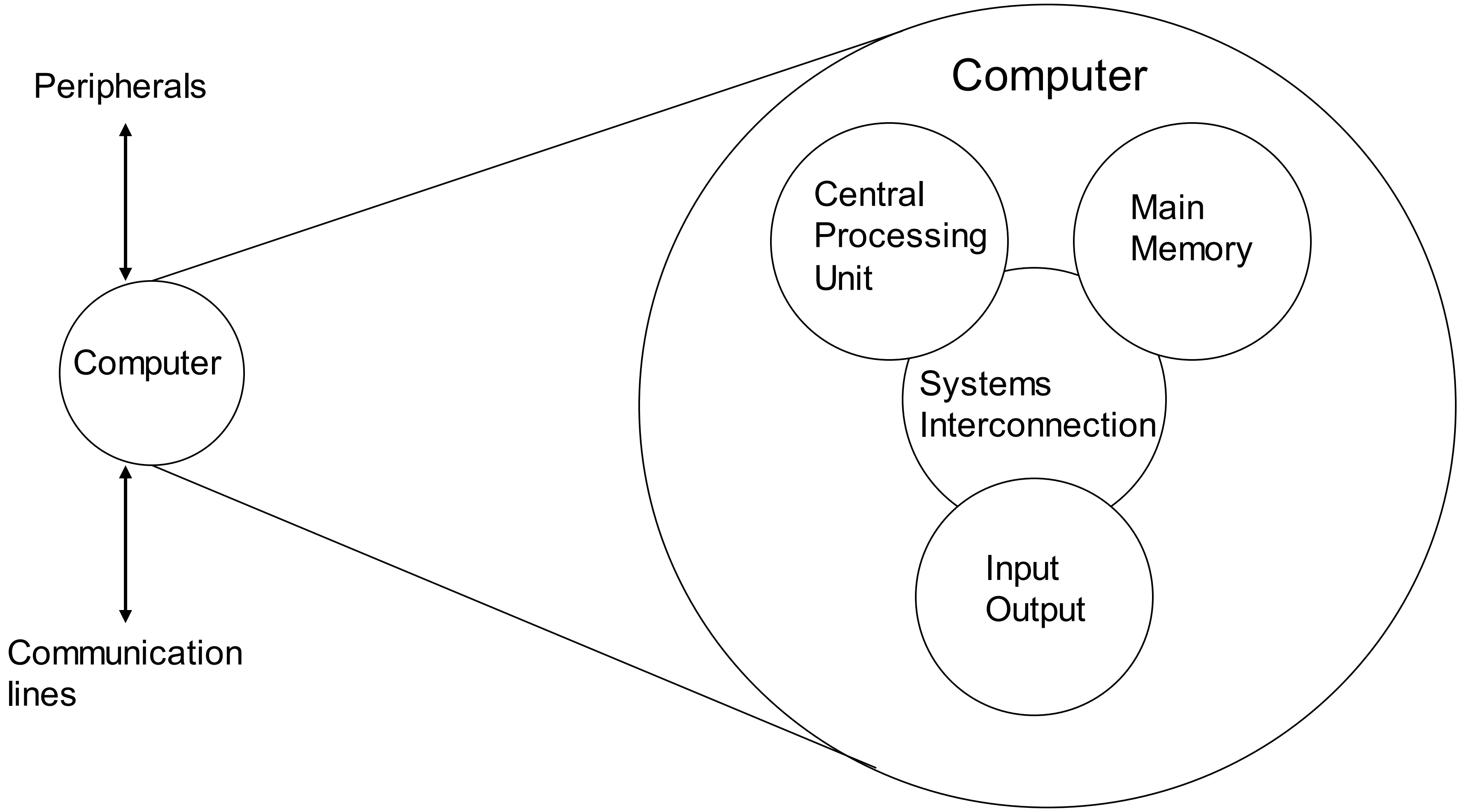
Operation (3) Processing from/to storage



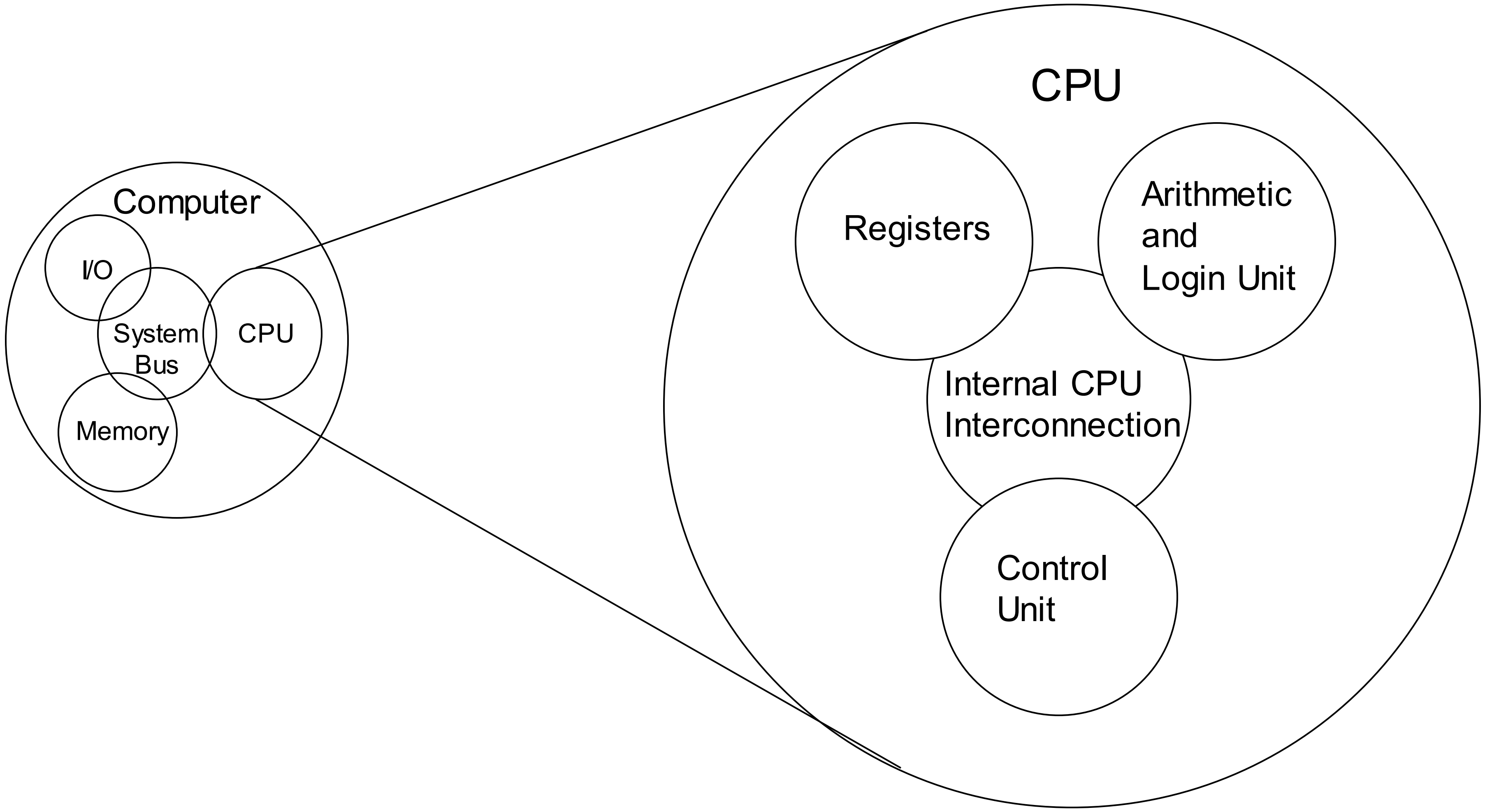
Operation (4)
Processing from storage to I/O



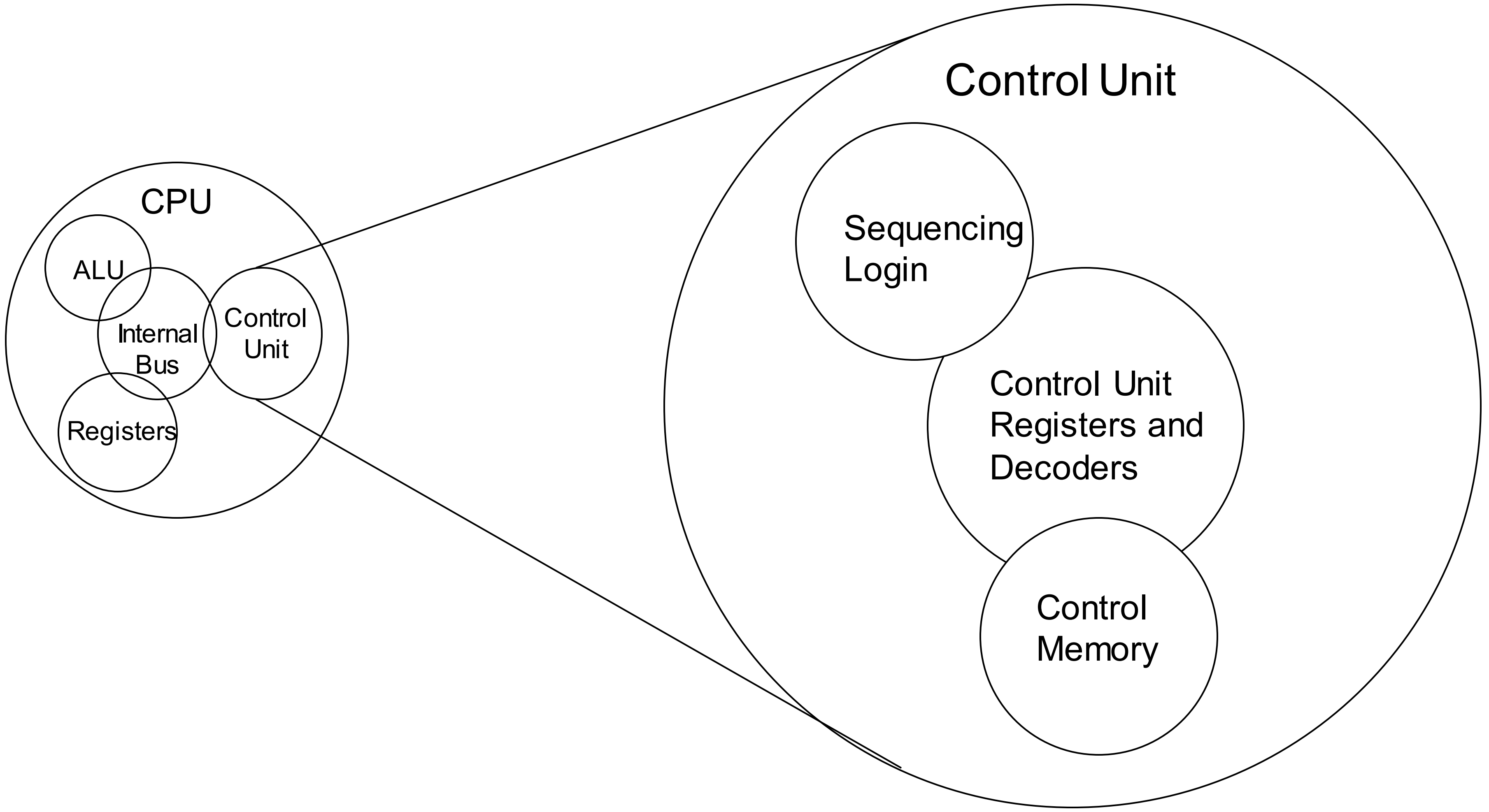
Structure - Top Level



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.
- Control Unit Operation.
- Multiprocessors and Vector Processing.
- Digital Logic.

Internet Resources
- Web site for book

- http://WilliamStallings.com/COA6e.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

Internet Resources
- Web sites to look for

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

Basics of Digital Logic Design

From transistors to chips

- Chips from the bottom up:
  - Basic building block: the transistor = "on/off switch"
    - Digital signals – voltage levels high/low
  - Transistors are used to build logic gates
  - Logic gates make up functional and control units
  - Microprocessors contain several functional and control units
- This section provides an introduction into digital logic
  - Combinatorial and sequential logic
  - Boolean algebra and truth tables
  - Basic logic circuits:
    - Decoders, multiplexers, latches, flip-flops
    - Simple register design

2

Signals, Logic Operations and Gates

- Rather than referring to voltage levels of signals, we shall consider signals that are logically 1 or 0 (or asserted or de-asserted).

Logic operation

NOT

A	A
0	1
1	0

AND

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

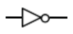
OR

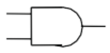
A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

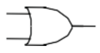
XOR

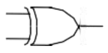
A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Gates









Output is 1 iff:	Input is 0	Both inputs are 1s	At least one input is 1	Inputs are not equal

- Gates are simplest digital logic circuits, and they implement basic logic operations (functions).
- Gates are designed using transistors.
- Gates are used to build more complex circuits that implement more complex logic functions.

3

Classification of Logic Functions/Circuits

- Combinational logic functions (circuits):
  - –any number of inputs and outputs
  - –outputs y_i depend only on current values of inputs x_i

  Logic equations may be used to define a logic function.

  Example: A logic function with 4 inputs and 2 outputs

  $y_1 = (x_1 + (x_2 * x_3)) + ((\overline{x_3} * x_4) * x_1)$ "*" used for "and", "+" used for "or"

  $y_2 = (\overline{x_1} + (x_2 * x_4)) + ((x_1 * x_2) * \overline{x_3})$

- For sequential functions (circuits):
  - – outputs depend on current values of inputs and some internal states.
- Any logic function (circuit) can be realized using only and, or and not operations (gates).
- nand and nor operations (gates) are universal.

4

Basic Laws of Boolean Algebra

- Identity laws: $A + 0 = A$
  $A * 1 = A$
- Inverse laws: $A + \overline{A} = 1$
  $A * \overline{A} = 0$
- Zero and one laws: $A + 1 = 1$
  $A * 0 = 0$
- Commutative laws: $A + B = B + A$
  $A * B = B * A$
- Associative laws: $A + (B + C) = (A + B) + C$
  $A * (B * C) = (A * B) * C$
- Distributive laws : $A * (B + C) = (A * B) + (A * C)$
  $A + (B * C) = (A + B) * (A + C)$
- DeMorgan's laws: $\overline{(A + B)} = \overline{A} * \overline{B}$
  $(A * B) = A + B$

5

Simple Circuit Design: Example

Given logic equations, it is easy to design a corresponding circuit

$$y_1 = (x_1 + (x_2 * x_3)) + ((\overline{x_3} * x_4) * x_1) = x_1 + (x_2 * x_3) + (\overline{x_3} * x_4 * x_1)$$

$$y_2 = (\overline{x_1} + (x_2 * x_4)) + ((x_1 * x_2) * \overline{x_3}) = \overline{x_1} + (x_2 * x_4) + (x_1 * x_2 * \overline{x_3})$$

6

Truth Tables

- Another way (in addition to logic equations) to define functionality
- Problem: their sizes grow exponentially with number of inputs.

inputs			outputs	
x_1	x_2	x_3	y_1	y_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

What are logic equations corresponding to this table?

$$y_1 = x_1 + x_2 + x_3$$

$$y_2 = x_1 * x_2 * x_3$$

Design corresponding circuit.

7

Logic Equations in Sum of Products Form

- Systematic way to obtain logic equations from a given truth table.

inputs			outputs	
x_1	x_2	x_3	y_1	y_2
0	0	0	1	1
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

- **A product term is included for each row where y_i has value 1**
- **A product term includes all input variables.**
- **At the end, all product terms are ored**

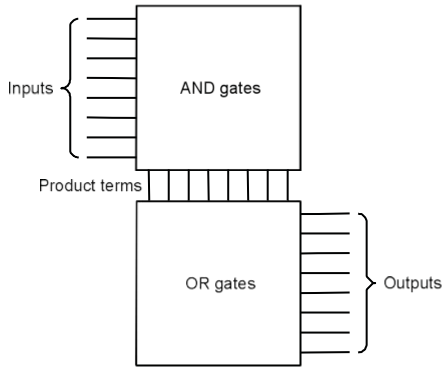
$$y_1 = \overline{x_1}*\overline{x_2}*\overline{x_3} + \overline{x_1}*\overline{x_2}*x_3 + x_1*\overline{x_2}*x_3 + x_1*x_2*x_3$$

$$y_2 = \overline{x_1}*\overline{x_2}*\overline{x_3} + \overline{x_1}*\overline{x_2}*x_3 + \overline{x_1}*x_2*\overline{x_3} + x_1*\overline{x_2}*\overline{x_3} + x_1*\overline{x_2}*x_3$$

8

Programmable Logic Array - PLA

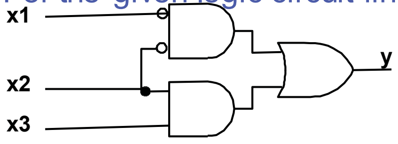
• PLA – structured logic implementation



9

Circuit ⓪ ⓪ Logic Equation Truth Table

• For the given logic circuit find its logic equation and truth table.



$y = \overline{x}_1 * \overline{x}_2 + x_2 * x_3$

x_1	x_2	x_3	y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

• Note that y column above is identical to y_1 column Slide 8.

• Thus, the given logic function may be defined with different logic equations and then designed by different circuits.

10

Minimization Using Boolean Laws

- Consider one of previous logic equations:

$$y_1 = \overline{x_1}*\overline{x_2}*\overline{x_3} + \overline{x_1}*\overline{x_2}*x_3 + x_1*\overline{x_2}*x_3 + x_1*x_2*x_3$$

$$= \overline{x_1}*\overline{x_2}*(\overline{x_3} + x_3) + x_2*x_3*(\overline{x_1} + x_1)$$

$$= \overline{x_1}*\overline{x_2} + x_2*x_3$$

But if we start grouping in some other way we may not end up with the minimal equation.

Gray codes

•a.k.a. reflected code – binary numeral system in which two successive values differ by only one digit

2-bit Gray code	3-bit Gray code	4-bit Gray code
00	000	0000
01	001	0001
11	011	0011
10	010	0010
	110	0110
	111	0111
	101	0101
	100	0100
		1100
		1101
		1111
		1110
		1010
		1011
		1001
		1000

Minimization Using Karnaugh Maps (1/4)

- Provides more formal way to minimization
- Includes 3 steps

1. **Form Karnaugh maps from the given truth table. There is one Karnaugh map for each output variable.**
2. **Group all 1s into as few groups as possible with groups as large as possible.**
3. **each group makes one term of a minimal logic equation for the given output variable.**

Forming Karnaugh maps – using "Gray code"

- **The key idea in forming the map is that horizontally and vertically adjacent squares correspond to input variables that differ in one variable only. Thus, a value for the first column (row) can be arbitrary, but labeling of adjacent columns (rows) should be such that those values differ in the value of only one variable.**

13

Minimization Using Karnaugh Maps (2/4)

Grouping (This step is critical)

When two adjacent squares contain 1s, they indicate the possibility of an algebraic simplification and they may be combined in one group of two. Similarly, two adjacent pairs of 1s may be combined to form a group of four, then two adjacent groups of four can be combined to form a group of eight, and so on. In general, the number of squares in any valid group must be equal to 2^k. Note that one 1 can be a member of more than one group and keep in mind that you should end up with as few groups as possible, which are as large as possible.

Finding Product Terms

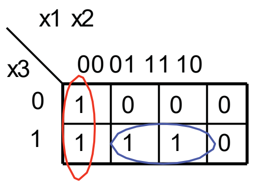
The product term that corresponds to a given group is the product of variables whose values are constant in the group. If the value of input variable x_i is 0 for the group, then $\overline{x_i}$ is entered in the product, while if x_i has value 1 for the group, then x_i is entered in the product.

14

Minimization Using Karnaugh Maps (3/4)

Example 1: Given truth table, find minimal circuit

x_1	x_2	x_3	y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



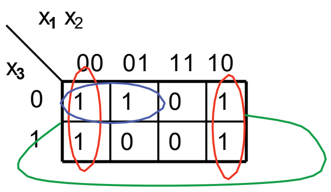
x3 \ x1 x2	00	01	11	10
0	1	0	0	0
1	1	1	1	0

$y = \overline{x_1}*\overline{x_2} + x_2*x_3$

15

Minimization Using Karnaugh Maps (4/4)

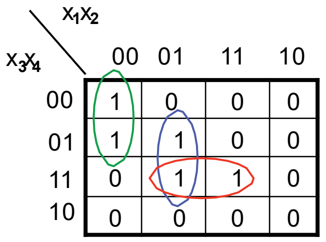
Example 2:



x_3 \ $x_1 x_2$	00	01	11	10
0	1	1	0	1
1	1	0	0	1

$y = \overline{x_1}*\overline{x_3} + \overline{x_2}$

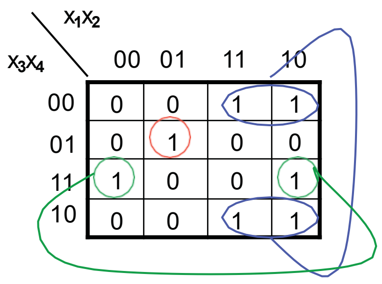
Example 3:



x_3x_4 \ x_1x_2	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	0	1	1	0
10	0	0	0	0

$y = \overline{x_1*x_2}*x_3 + \overline{x_1}*x_2*x_4 + x_2*x_3*x_4$

Example 4:



x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	1	0	0	1
10	0	0	1	1

$y = x_1*\overline{x}_4 + \overline{x_2}*x_3*x_4 + x\,\overline{*}x_2{}_3{*}x_4\overline{*}x$

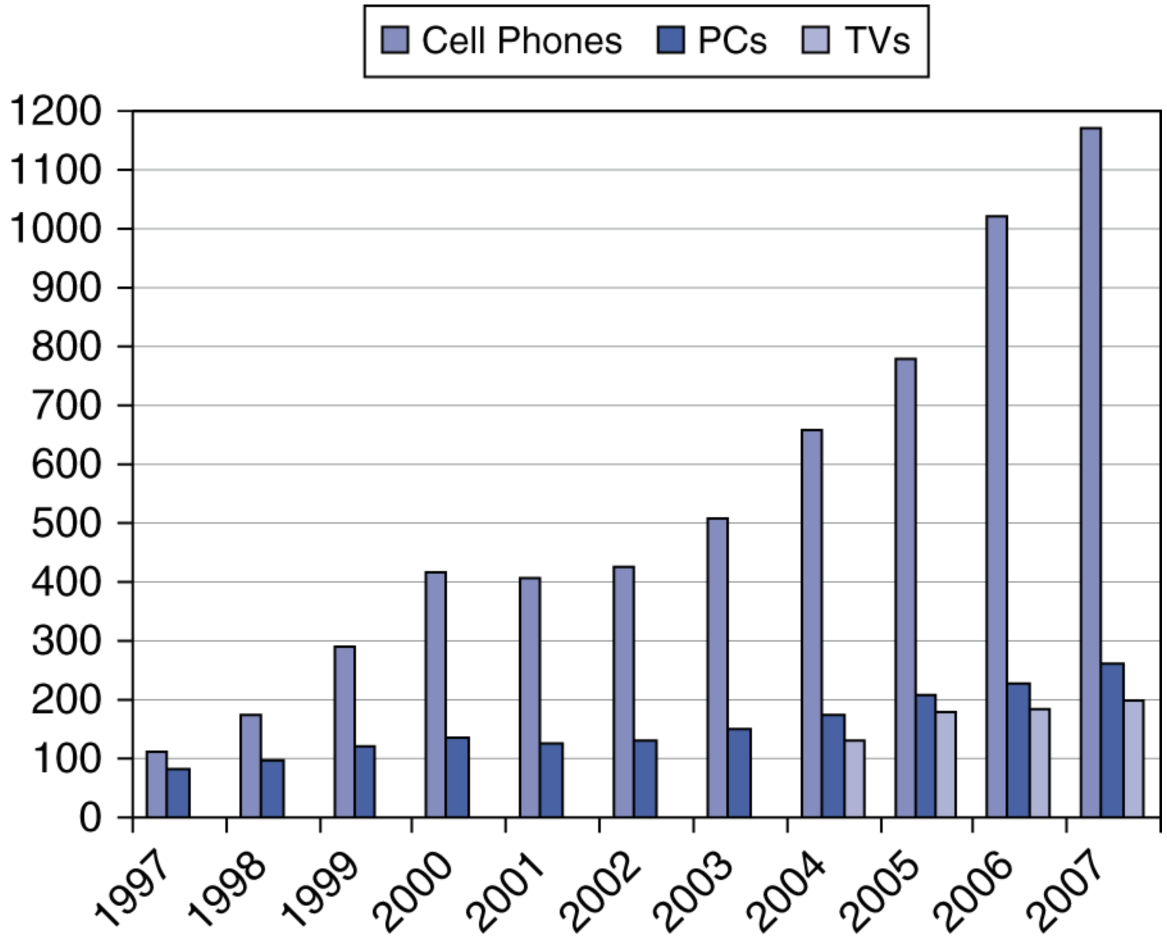
16

Computer Abstractions and Technology

Classes of Computers

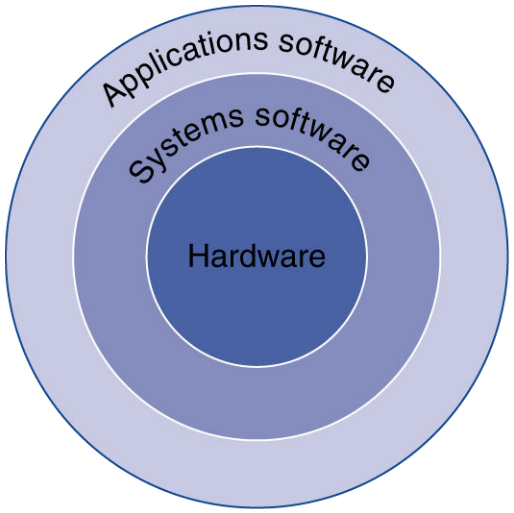
- Desktop computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

The Processor Market

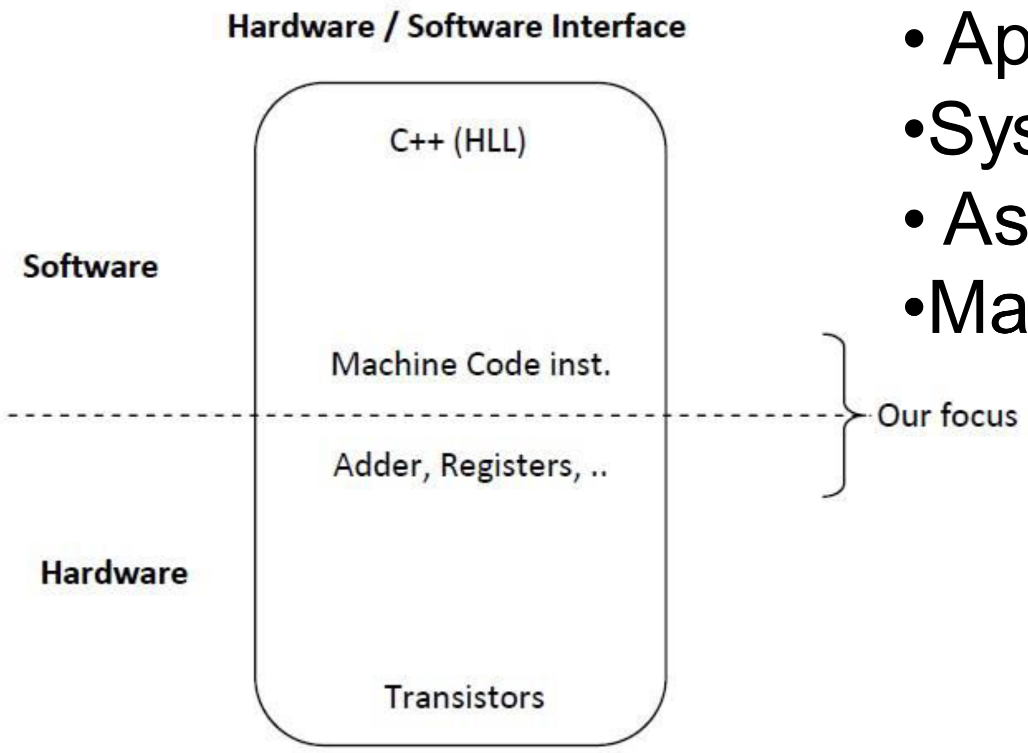


Below Your Program

- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers



How do computers work?

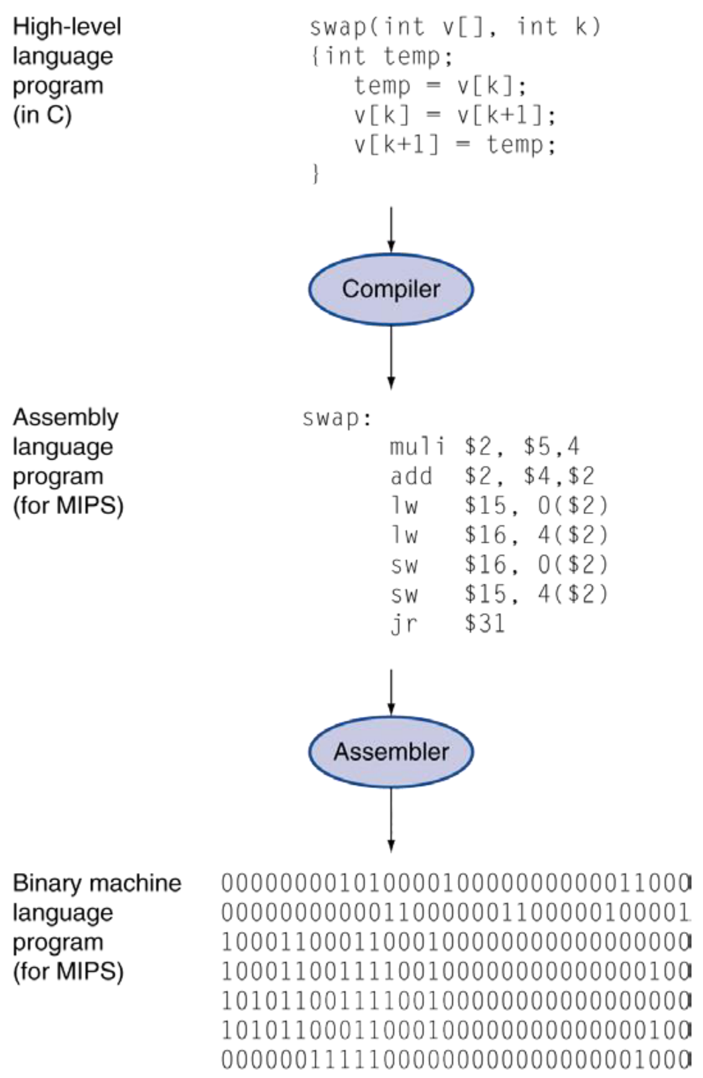


Need to understand abstractions such as:

- Application software
- System software
- Assembly language
- Machine language

Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data


High-level language program (in C)
Compiler
Assembly language program (for MIPS)
Assembler
Binary machine language program (for MIPS)

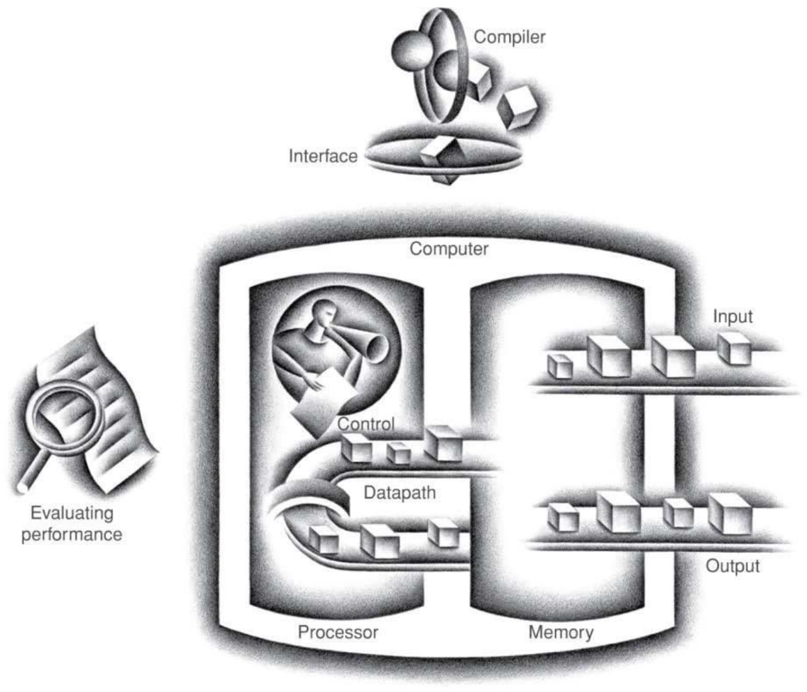
```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

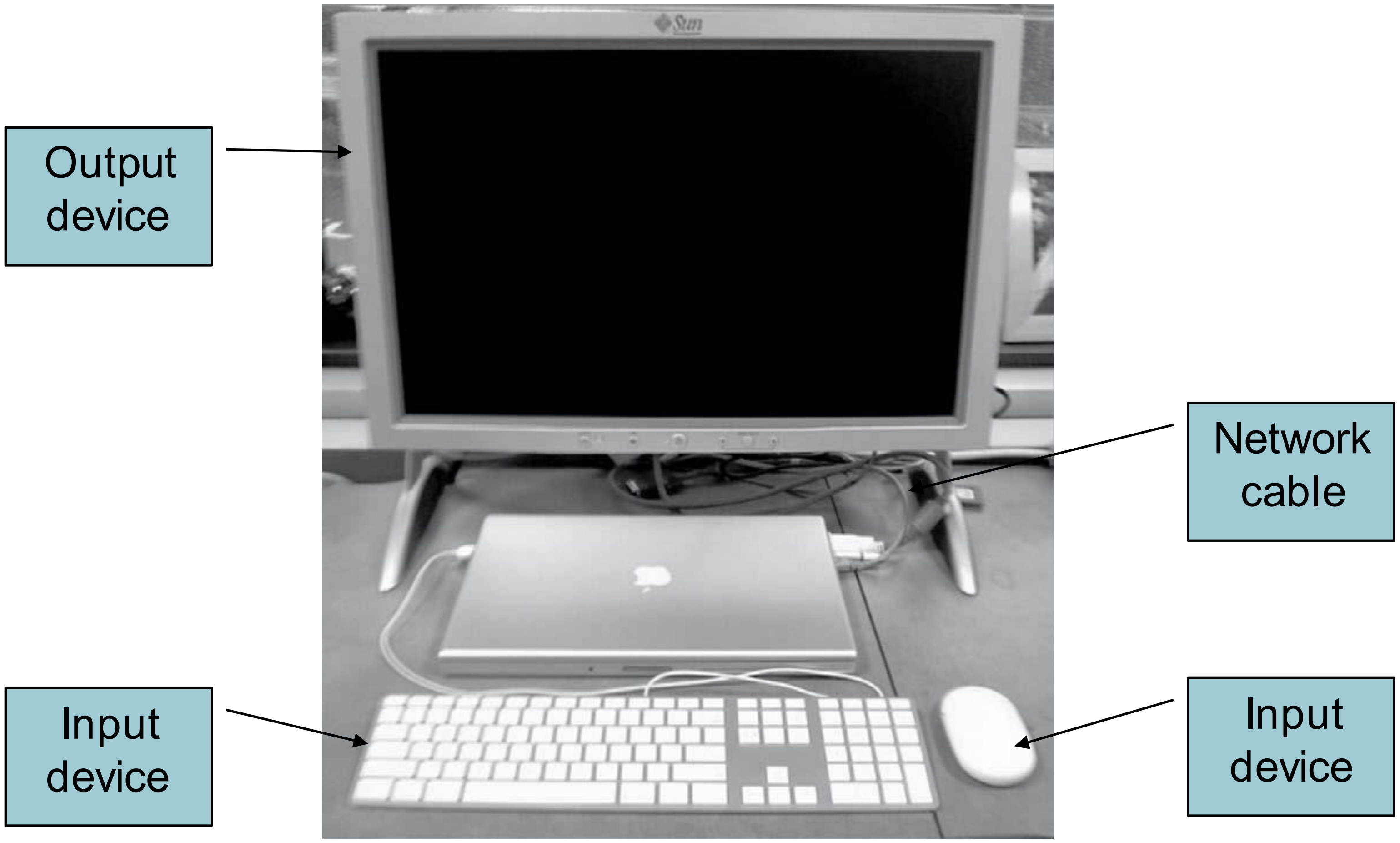
Components of a Computer

The BIG Picture



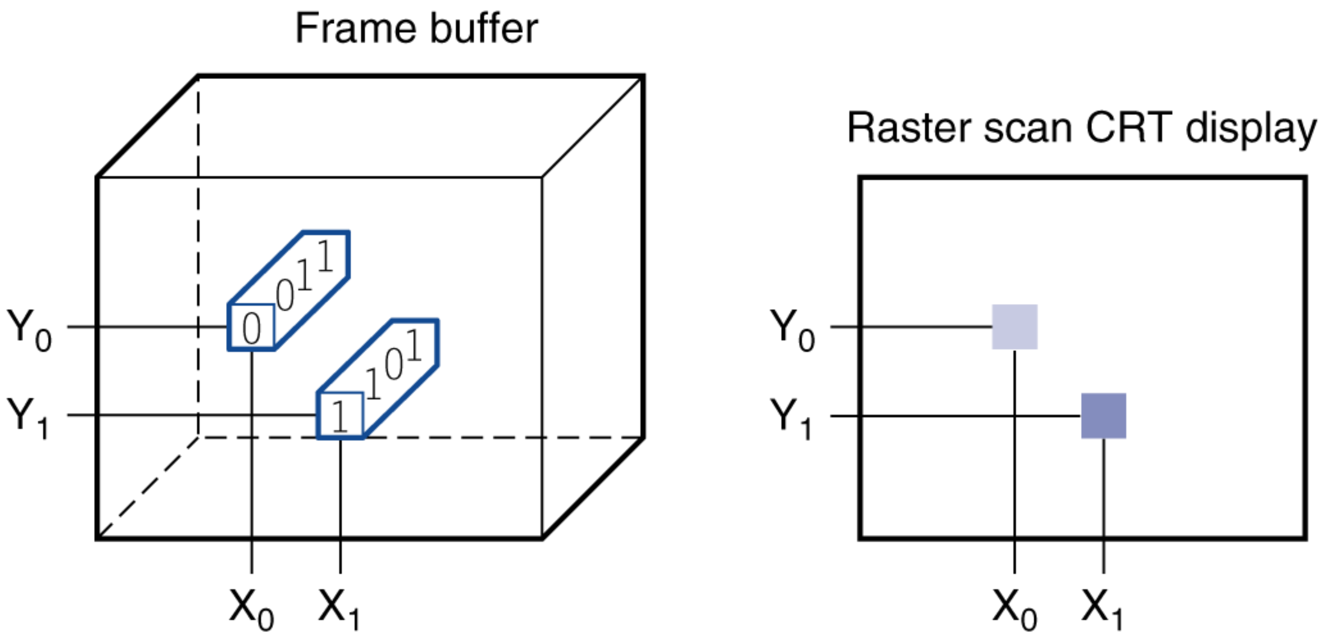
- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

Anatomy of a Computer

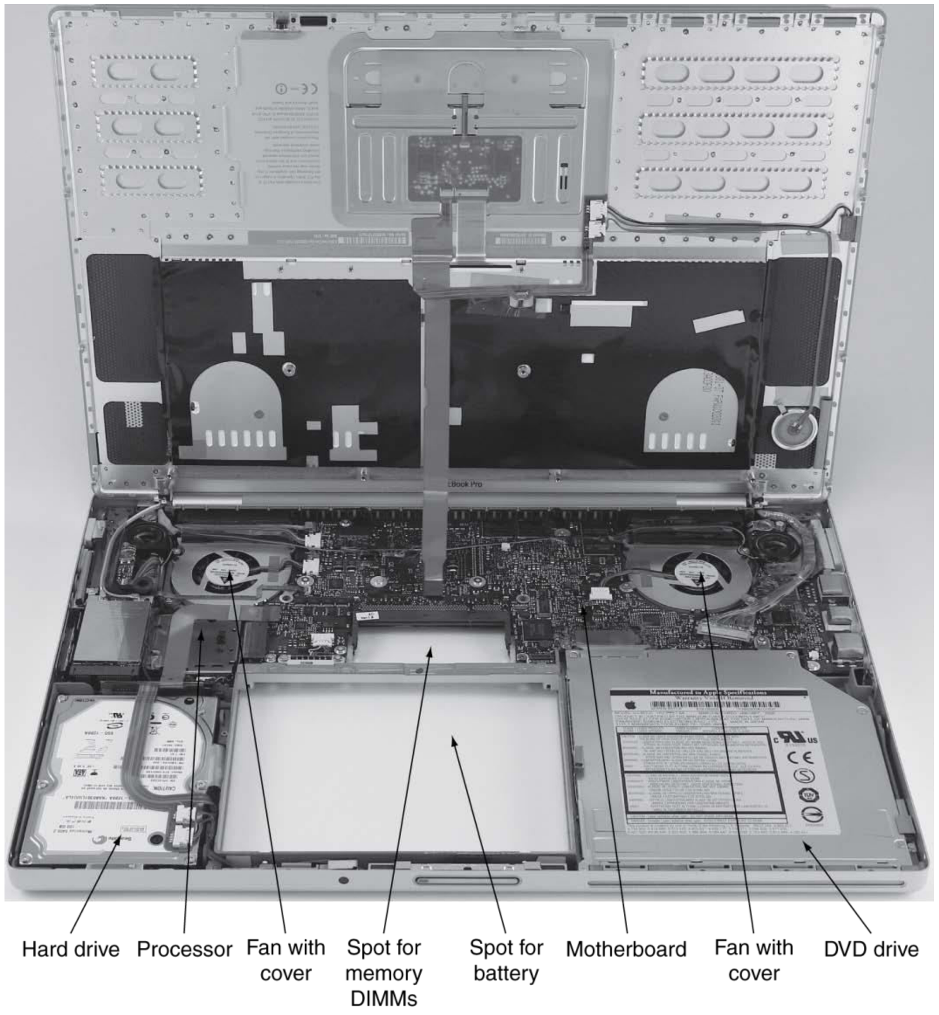


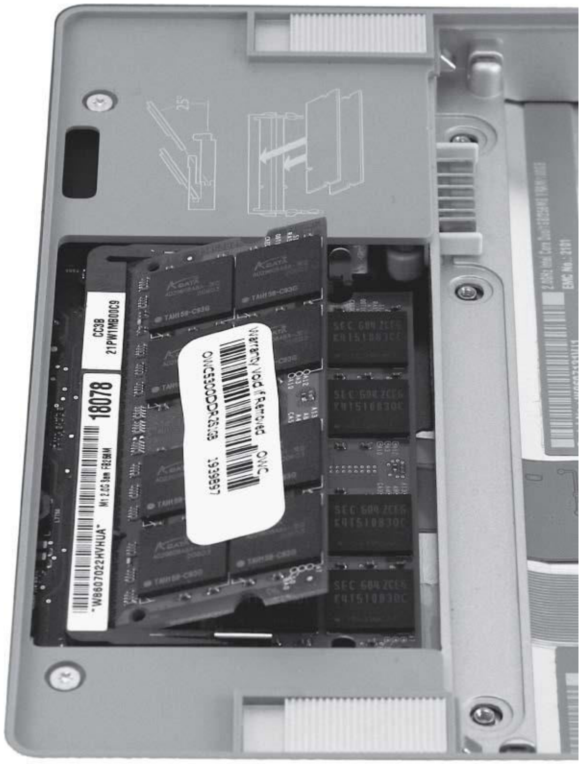
Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory



Opening the Box



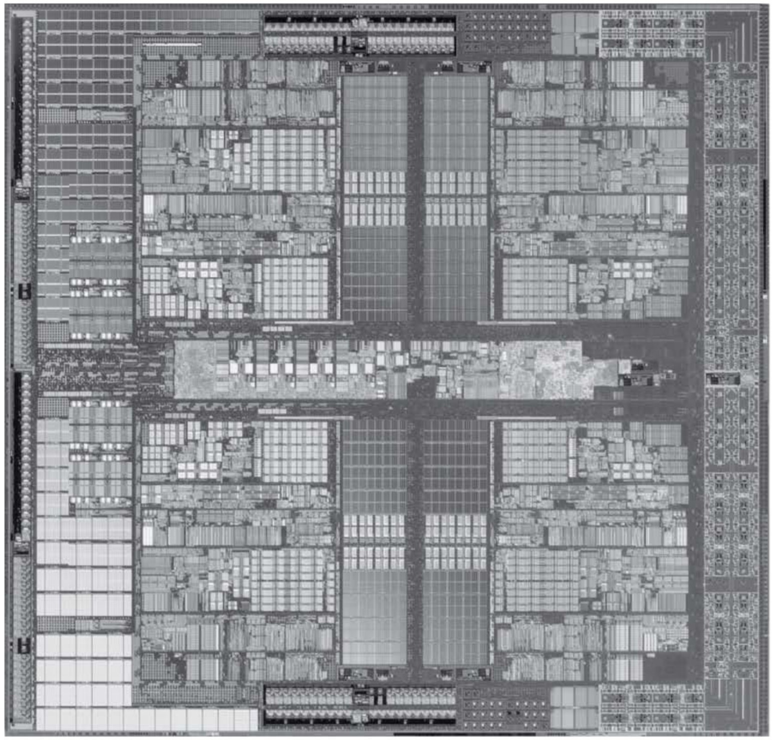


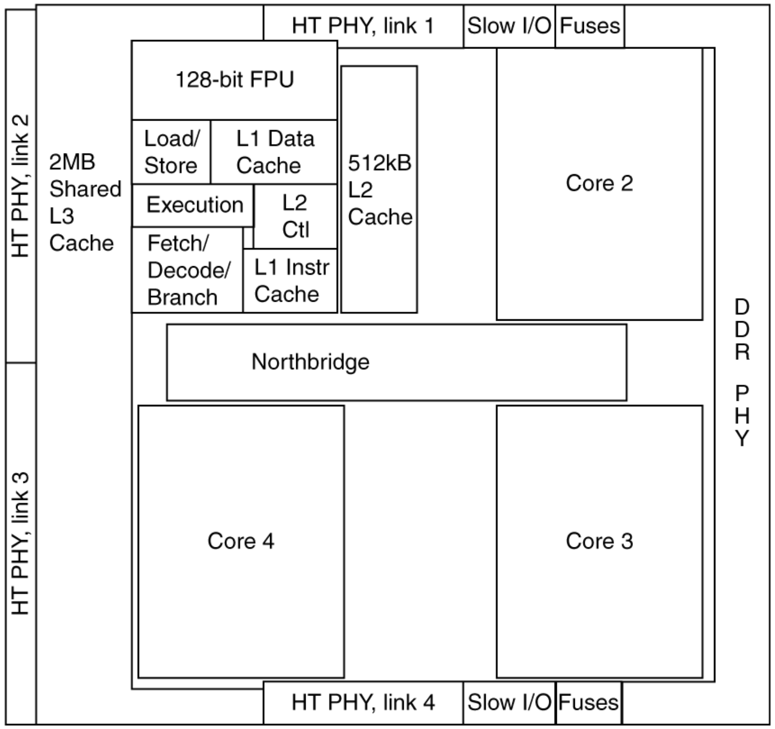
Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

Inside the Processor

- AMD Barcelona: 4 processor cores





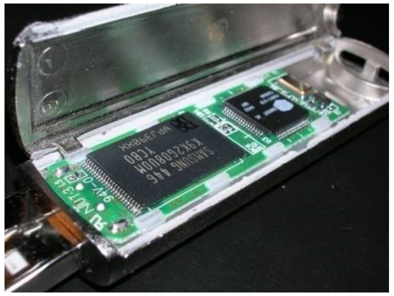
Abstractions

The BIG Picture

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

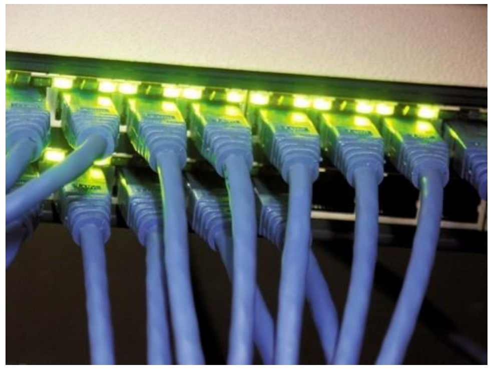
A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



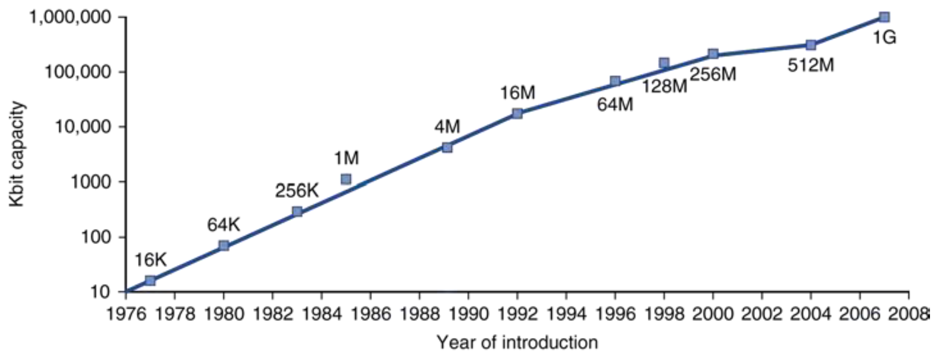
Networks

- Communication and resource sharing
- Local area network (LAN): Ethernet
  - Within a building
- Wide area network (WAN: the Internet
- Wireless network: WiFi, Bluetooth



Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost

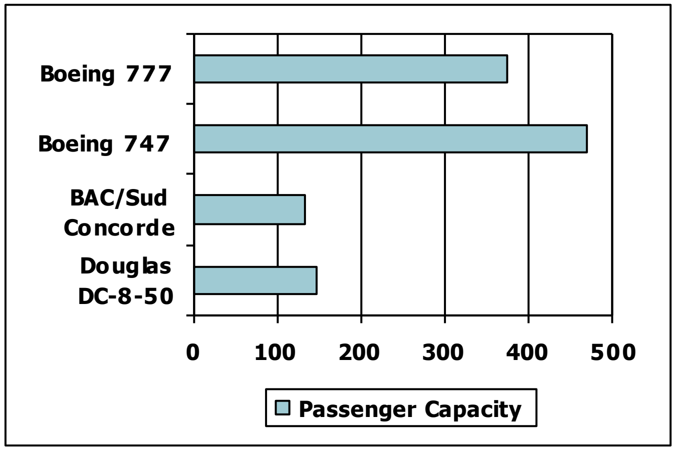


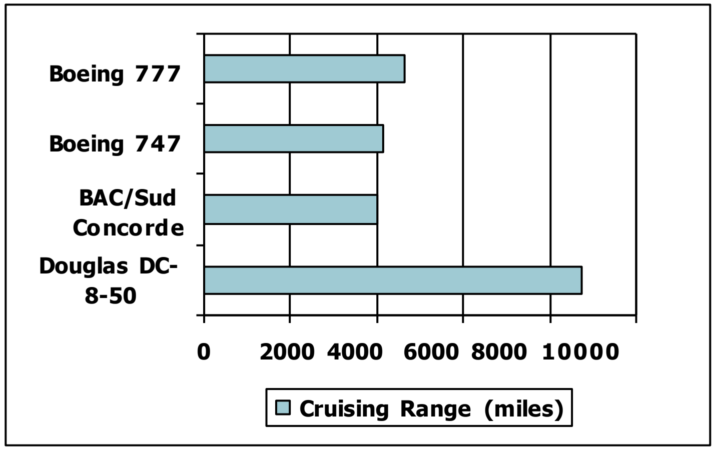
DRAM capacity

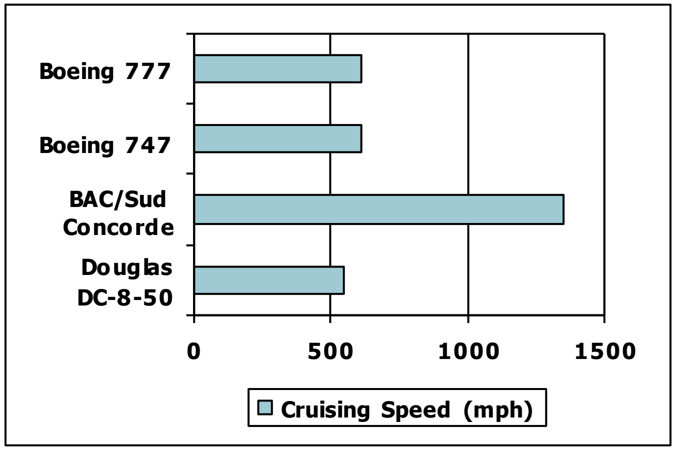
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2005	Ultra large scale IC	6,200,000,000

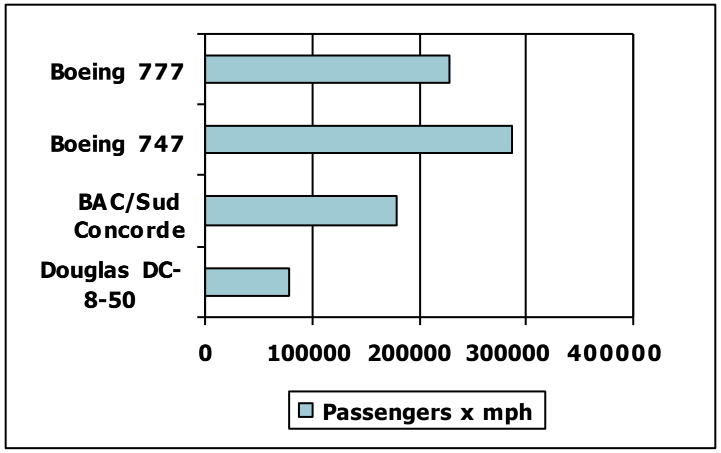
Defining Performance

- Which airplane has the best performance?









Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

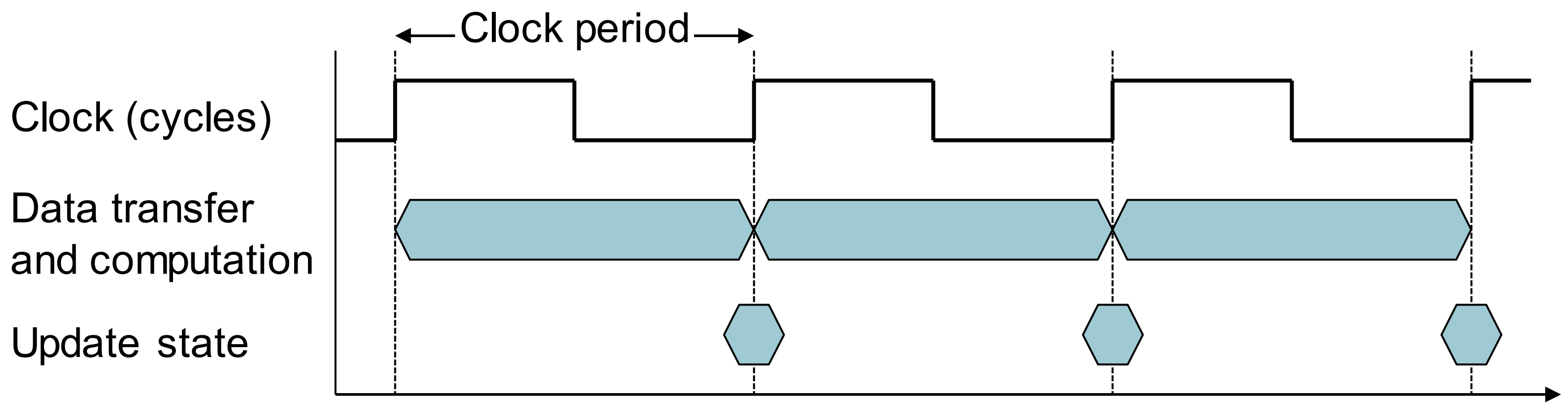
- Define Performance = 1/Execution Time
- "X is n time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

Chapter 4: Microprogrammed and Hardwired Control Unit

Dr. Moaath Shatnawi

Hardwired control unit

- Hardwired control units are generally faster than microprogrammed designs.

- In hardwired control, we will saw how all the control signals required inside the CPU can be generated using a state counter and a PLA circuit.

A microprogrammed control unit

- A microprogrammed control unit is a relatively simple logic circuit that is capable of:

1. sequencing through microinstructions and
2. generating control signals to execute each microinstruction.

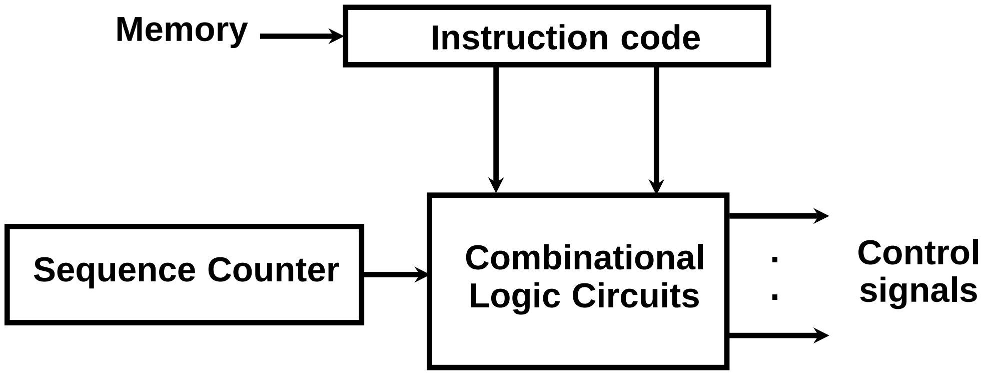
some Important Terms

- **Control Word :** A control word is a word whose individual bits represent various control signals.
- **Micro-routine :** A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.

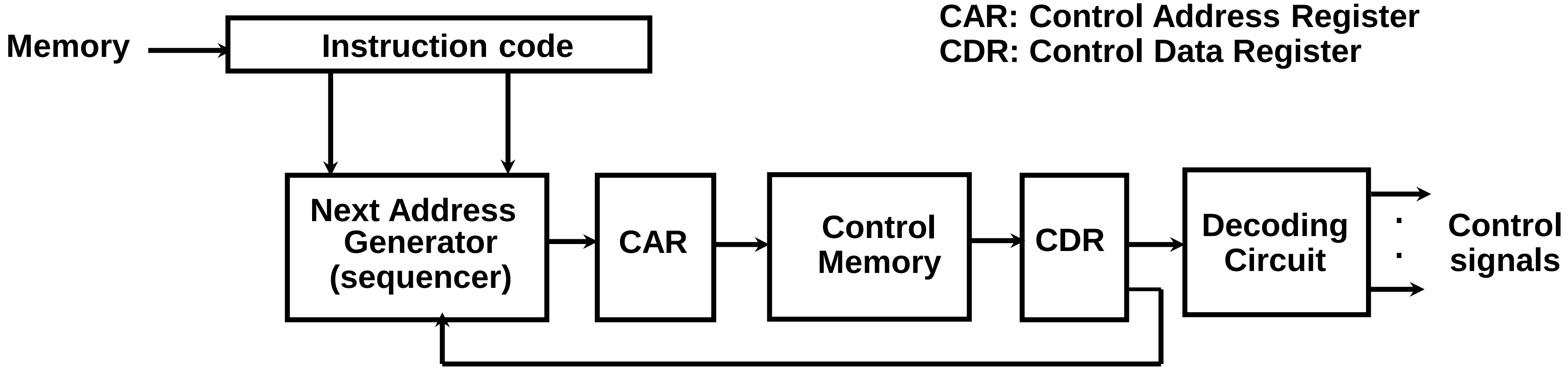
- **Microinstruction :** Individual control words in this micro-routine are referred to as microinstructions.
- **Micro-program :** A sequence of micro-instructions is called a micro-program, which is stored in a ROM or RAM called a Control Memory (CM).
- **Control Store :** the micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the Control Store.

Control Unit Implementation

- Hardwired



- Microprogrammed



CAR: Control Address Register
CDR: Control Data Register

Microprogrammed Control Unit

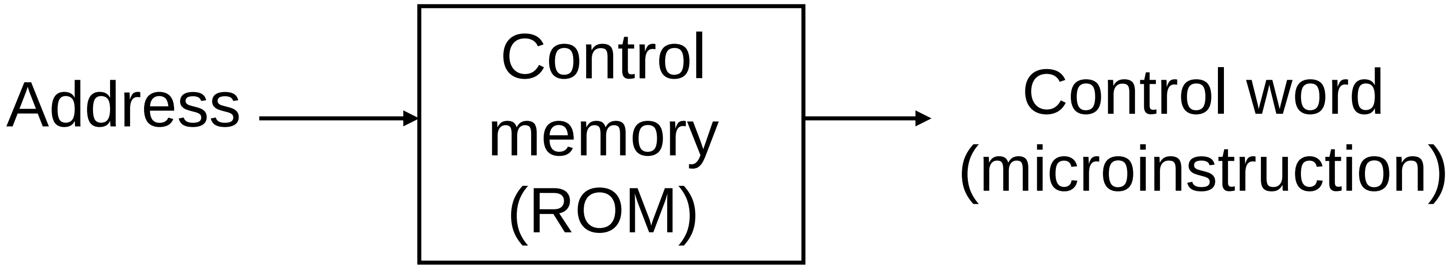
- Control signals
  - Group of bits used to select paths in multiplexers, decoders, arithmetic logic units
- Control variables
  - Binary variables specify microoperations
    - Certain microoperations initiated while others idle
- Control word
  - String of 1’s and 0’s represent control variables

Microprogrammed Control Unit

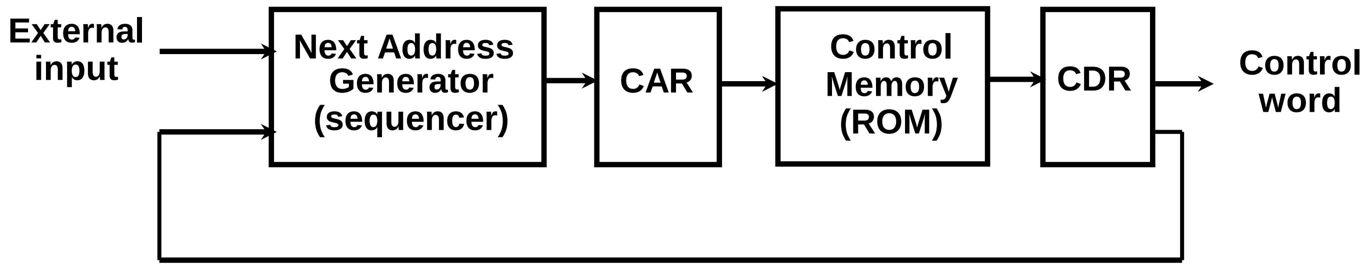
- Control memory
  - Memory contains control words
- Microinstructions
  - Control words stored in control memory
  - Specify control signals for execution of micro-operations
- Microprogram
  - Sequence of microinstructions

Control Memory

- Read-only memory (ROM)
- Content of word in ROM at given address specifies microinstruction
- Each computer instruction initiates series of microinstructions (microprogram) in control memory
- These microinstructions generate microoperations to
  - Fetch instruction from main memory
  - Evaluate effective address
  - Execute operation specified by instruction
  - Return control to fetch phase for next instruction



Microprogrammed Control Organization



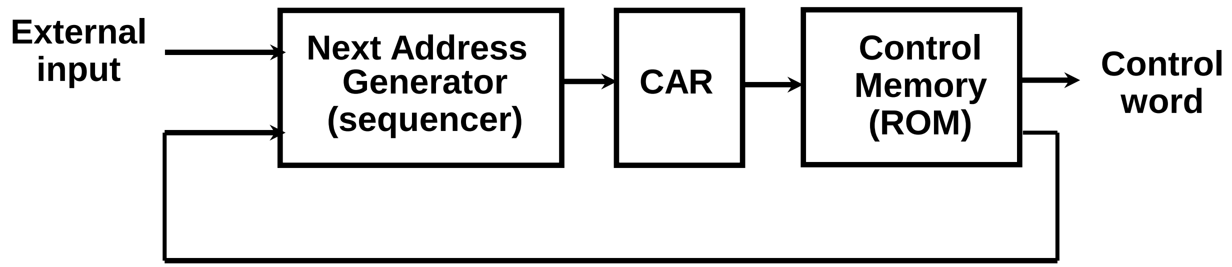
- Control memory
  - Contains microprograms (set of microinstructions)
  - Microinstruction contains
    - Bits initiate microoperations
    - Bits determine address of next microinstruction
- Control address register (CAR)
  - Specifies address of next microinstruction

Microprogrammed Control Organization

- Next address generator (microprogram sequencer)
  - Determines address sequence for control memory
- Microprogram sequencer functions
  - Increment CAR by one
  - Transfer external address into CAR
  - Load initial address into CAR to start control operations

Microprogrammed Control Organization

- Control data register (CDR)- or pipeline register
  - Holds microinstruction read from control memory
  - Allows execution of microoperations specified by control word simultaneously with generation of next microinstruction
- Control unit can operate without CDR



Microprogram Routines

- Routine
  - Group of microinstructions stored in control memory
- Each computer instruction has its own microprogram routine to generate micro-operations that execute the instruction

Microprogram Routines

- Subroutine
  - Sequence of microinstructions used by other routines to accomplish particular task
- Example
  - Subroutine to generate effective address of operand for memory reference instruction
- Subroutine register (SBR)
  - Stores return address during subroutine call

Conditional Branching

- Branching from one routine to another depends on status bit conditions
- Status bits provide parameter info such as
  - Carry-out of adder
  - Sign bit of number
  - Mode bits of instruction
- Info in status bits can be tested and actions initiated based on their conditions: 1 or 0
- Unconditional branch
  - Fix value of status bit to 1

Mapping of Instruction

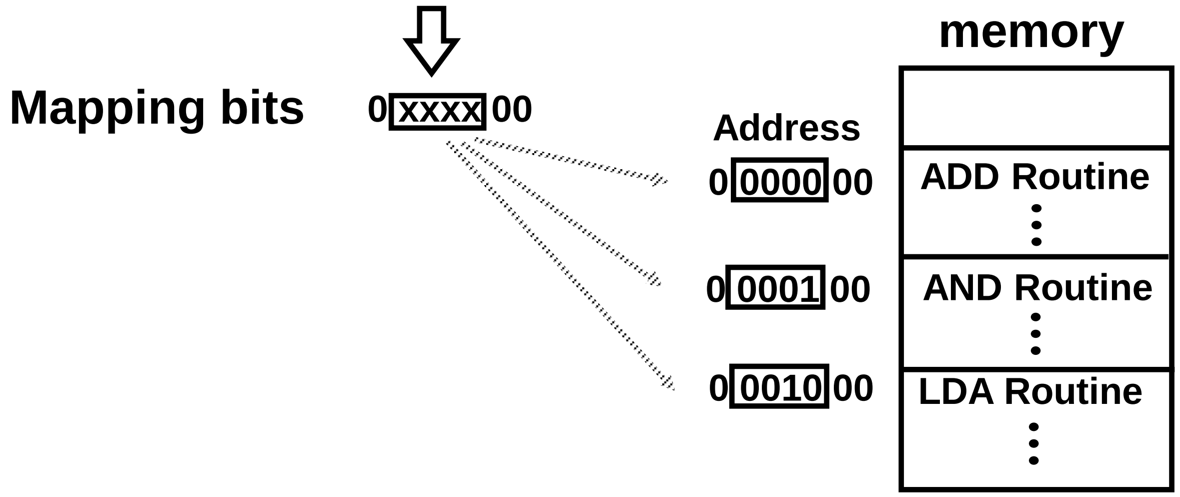
- Each computer instruction has its own microprogram routine stored in a given location of the control memory
- Mapping
  – Transformation from instruction code bits to address in control memory where routine is located

Mapping of Instruction

- Example
  - Mapping 4-bit operation code to 7-bit address

OP-codes of Instructions

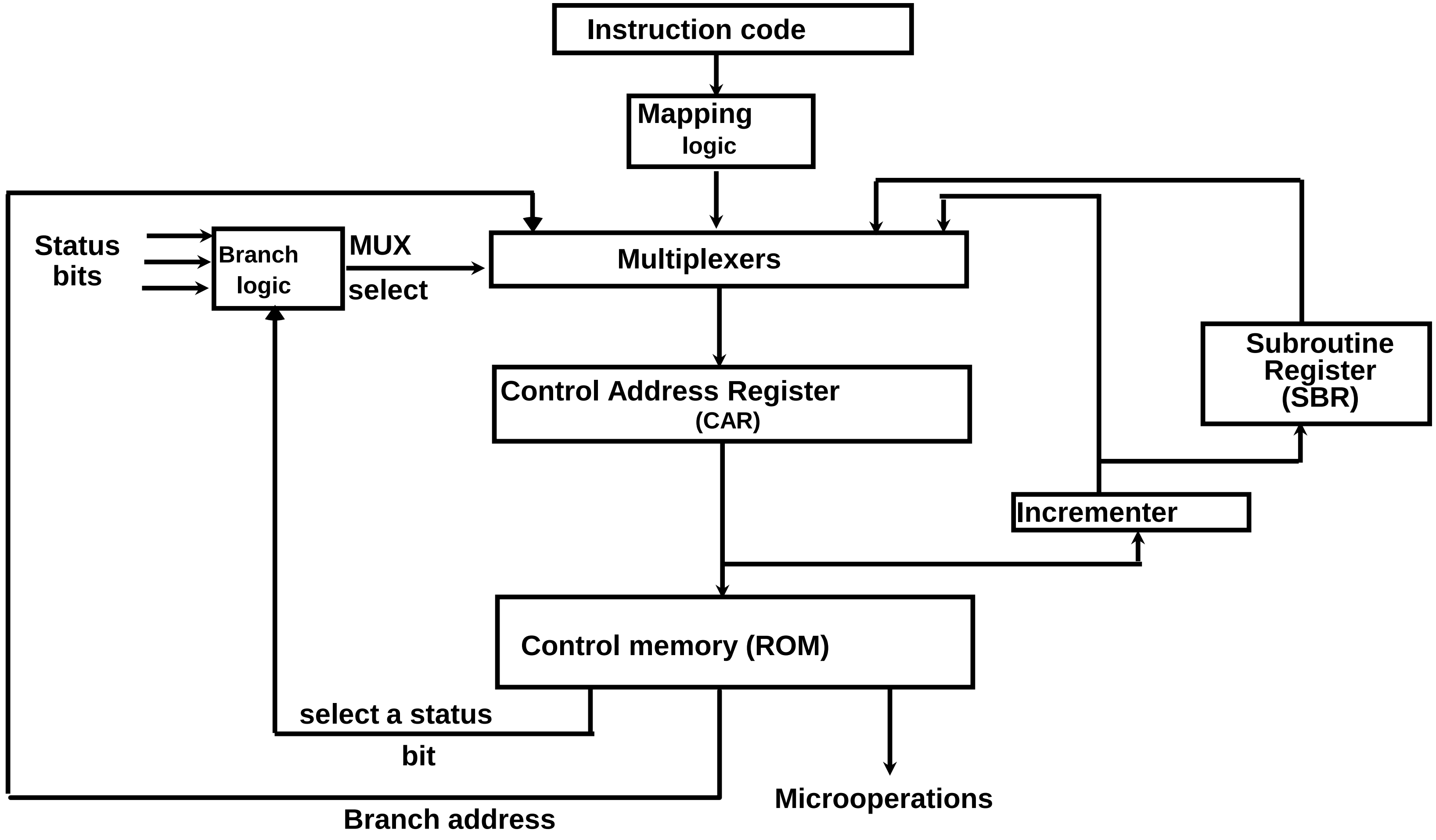
ADD	0000
AND	0001
LDA	0010



Address Sequencing

- Address sequencing capabilities required in control unit
  - Incrementing CAR
  - Unconditional or conditional branch, depending on status bit conditions
  - Mapping from bits of instruction to address for control memory
  - Facility for subroutine call and return

Address Sequencing



Microprogram Example

Computer instruction format

15	14 11	10 0
I	Opcode	Address

Microinstruction Format

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

Hardwired control unit characteristics

- Hardwired control unit generates the control signals needed for the processor using logic circuits
- Hardwired control unit is faster when compared to microprogrammed control unit as the required control signals are generated with the help of hardware's
- Difficult to modify as the control signals that need to be generated are hard wired
- More costlier as everything has to be realized in terms of logic gates
- It cannot handle complex instructions as the circuit design for it becomes complex
- Only limited number of instructions are used due to the hardware implementation
- Used in computer that makes use of Reduced Instruction Set Computers(RISC)

Microprogrammed control unit characteristics

- Microprogrammed control unit generates the control signals with the help of micro instructions stored in control memory
- This is slower than the other as micro instructions are used for generating signals here
- Easy to modify as the modification need to be done only at the instruction level
- Less costlier than hardwired control as only micro instructions are used for generating control signals
- It can handle complex instructions
- Control signals for many instructions can be generated
- Used in computer that makes use of Complex Instruction Set Computers(CISC)

Application of microprogrammed

- Realization of computers
- Emulation
- Operating system support
- Realization of special purpose devices
- High-level language support
- Micro diagnostic