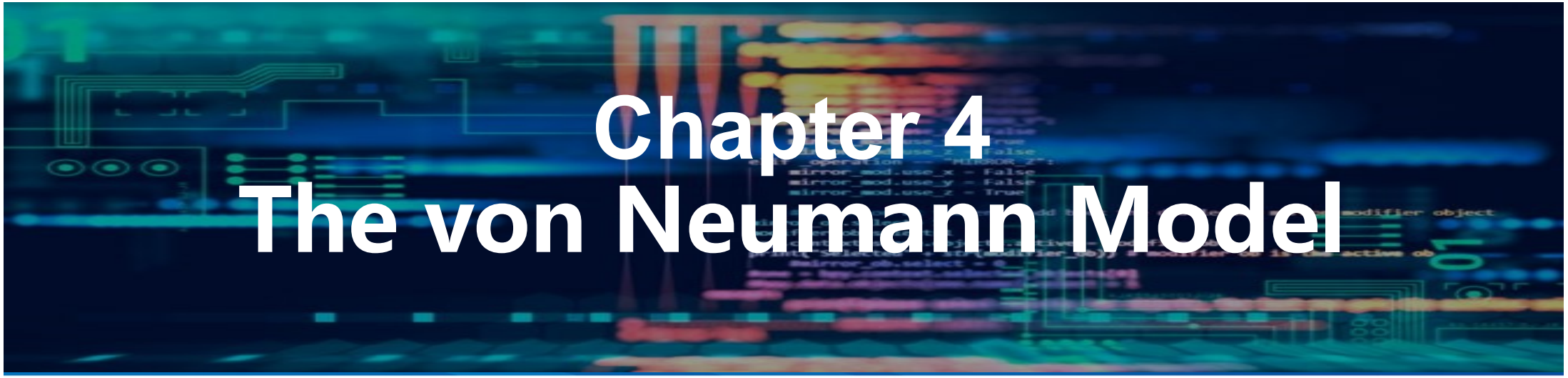
中国科学技术大学
University of Science and Technology of China

计算系统概论A
Introduction to Computing Systems
(CS1002A.03)



Chapter 4
The von Neumann Model

陈俊仕
cjuns@ustc.edu.cn
2023 Fall
计算机科学与技术学院
School of Computer Science and Technology

Review

- **MOS transistors are used as switches to implement logic functions.**
  - **N-type: connect to GND, turn on (with 1) to pull down to 0**
  - **P-type: connect to +2.9V, turn on (with 0) to pull up to 1**
- **Basic gates: NOT, NOR, NAND**
  - **Logic functions are usually expressed with AND, OR, and NOT**
- **Properties of logic gates**
  - **Completeness**
    - — can implement any truth table with AND, OR, NOT
  - **DeMorgan's Law**
    - — convert AND to OR by inverting inputs and output

Review

- We’ ve touched on basic digital logic
  - Transistors
  - Gates
  - Storage (latches, flip-flops, memory)
  - State machines
- Built some simple circuits
  - adder, subtracter, adder/subtracter, Incrementer
  - Counter (consisting of register and incrementer)
  - Hard-coded traffic sign state machine
  - Programmable traffic sign state machine
- Up next: a computer as a state machine

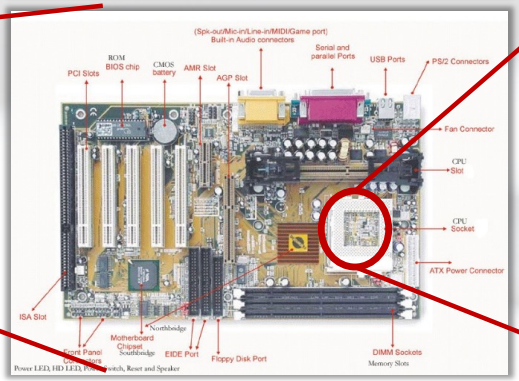
Today

- Great Idea #2: Stored program computer(Von Neumann Model--A Machine Structure
  - Basic Components for a machine
  - The LC-3: An Example von Neumann Machine
  - Instruction Processing

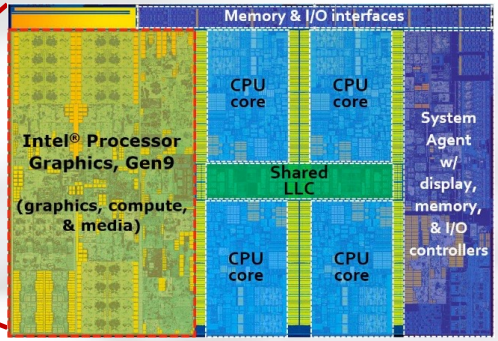
Bottom up approach

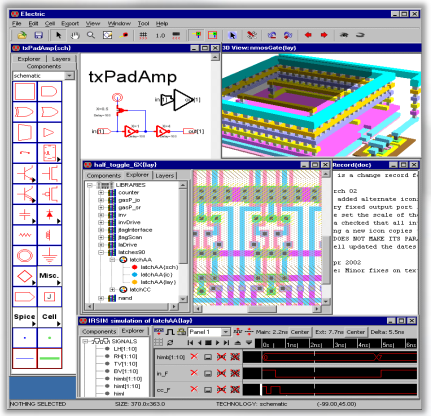
Personal Computer:
Hardware & Software Design
1~10PCBs/System

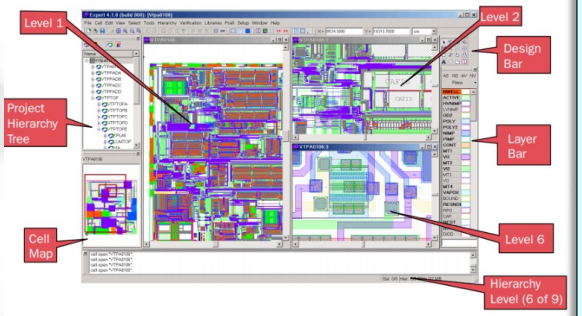
Motherboard Circuit Design
10 ICs/ PCB
1~50G Devices

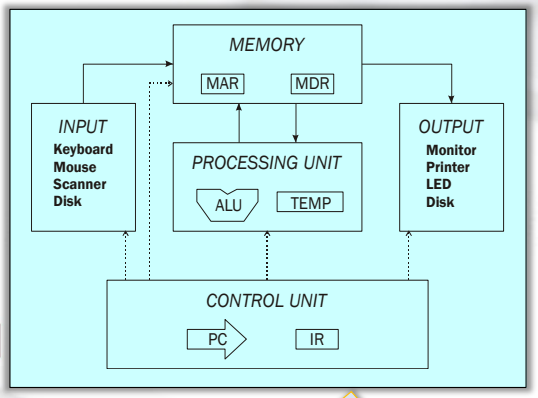


Integrated Circuit Design
100 Modules/ IC
0.25M~20G Devices





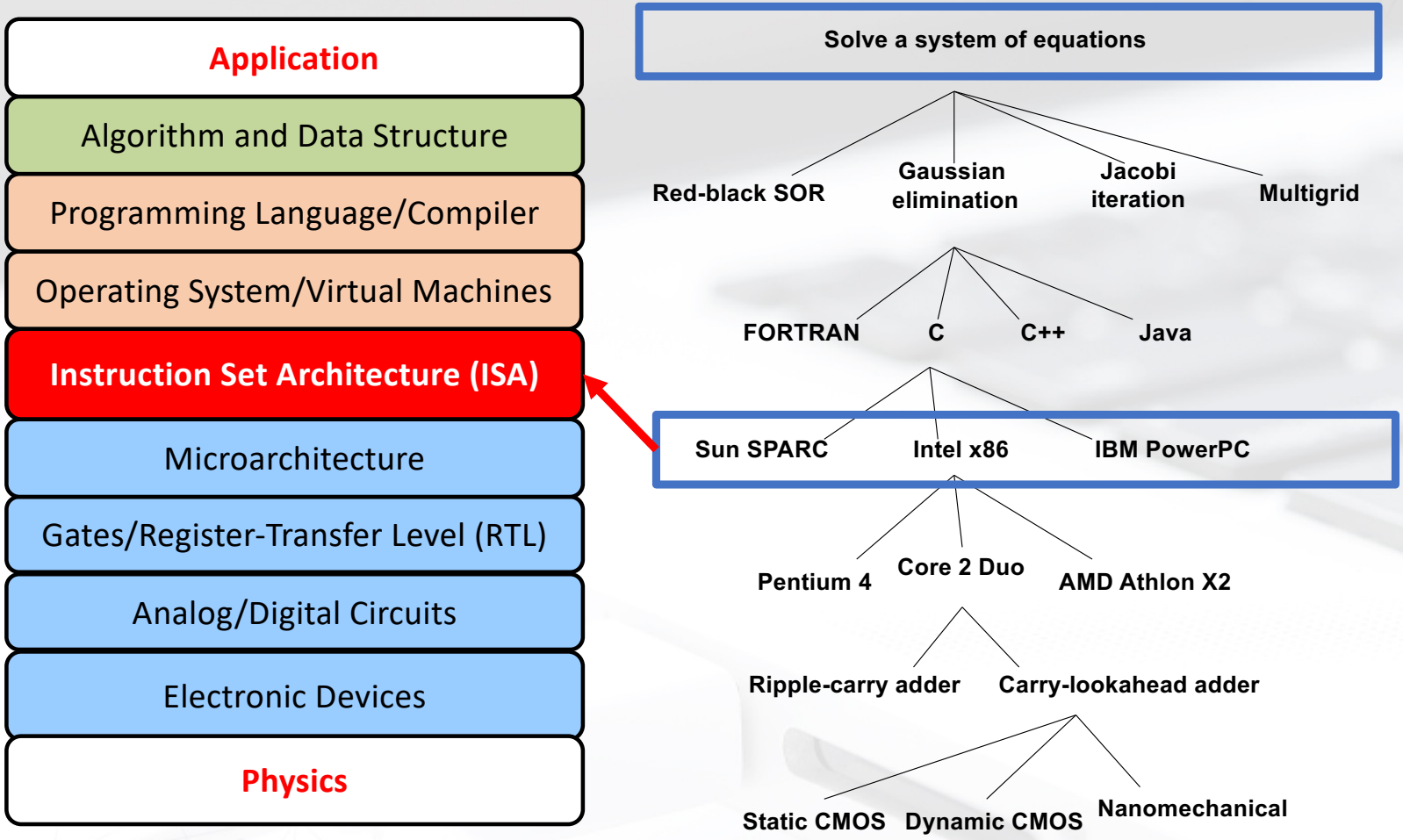




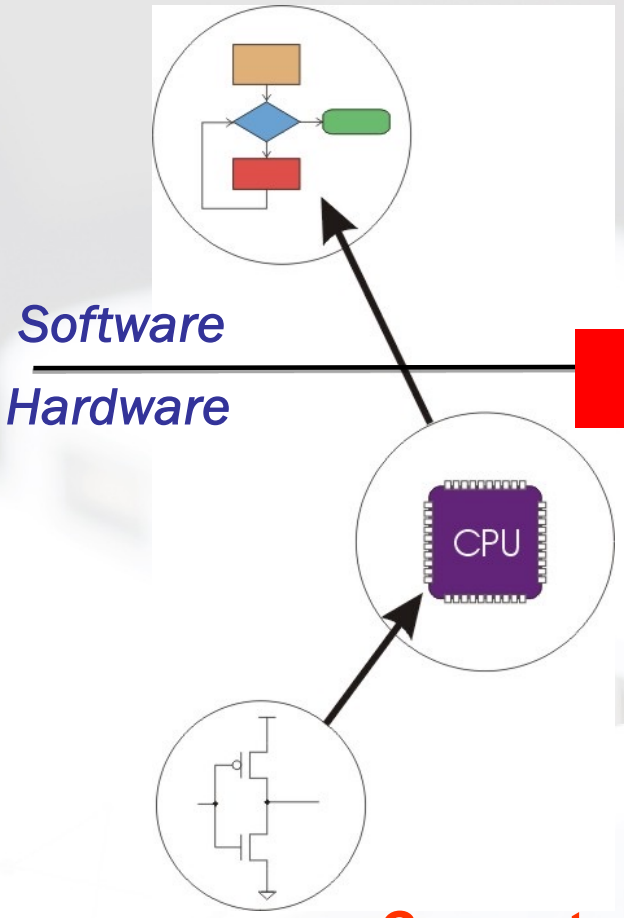
Electronic System Level (ESL)Design

Now, You are Here.

Great Idea #3: Abstraction Helps Us Manage Complexity



Great Idea #4: Software and Hardware Co-design



Application

Algorithm & Data Structure

Language

Machine Architecture, ISA

Microarchitecture

Logic and IC

Now, You are Here.

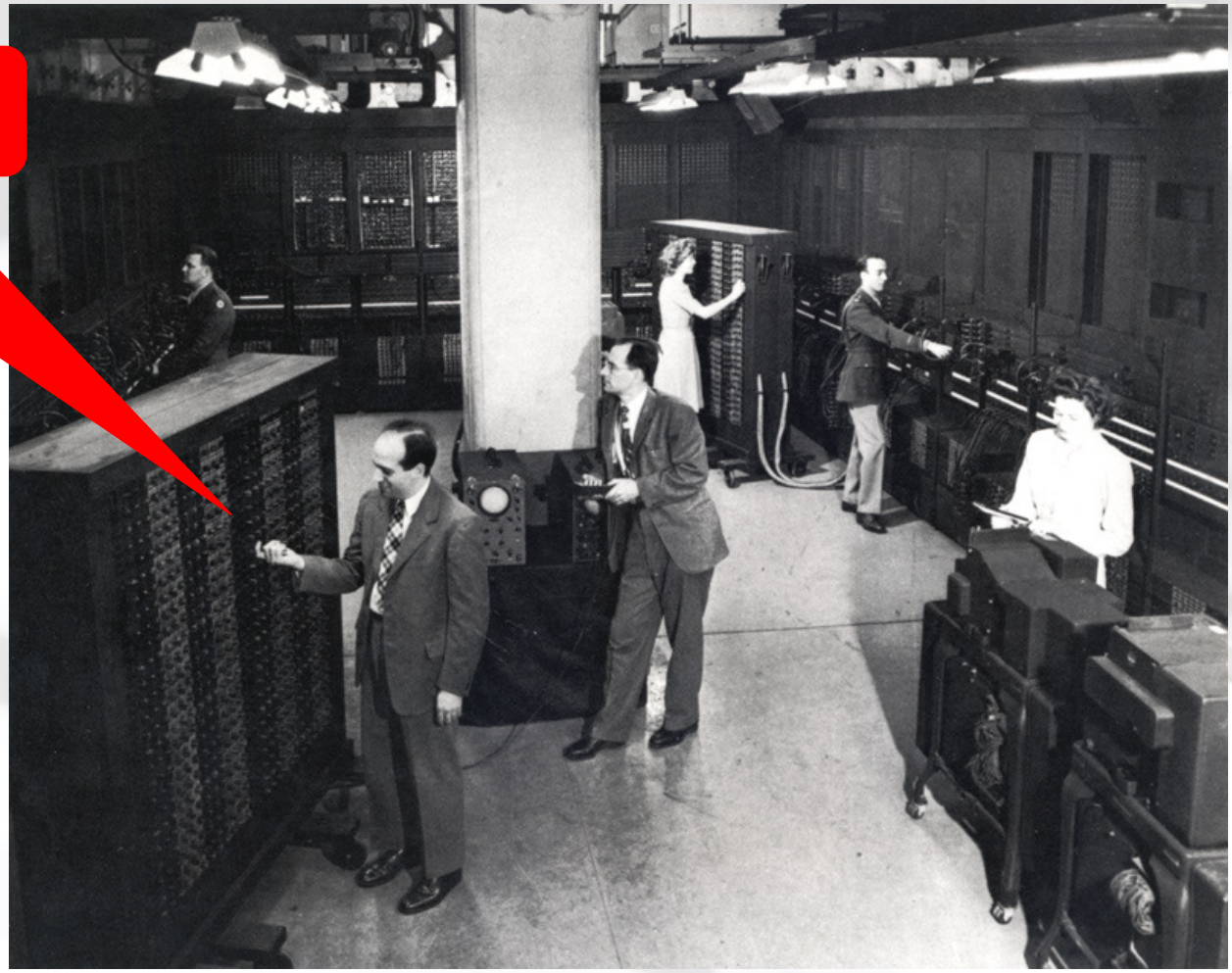
Device

Computer System: Layers of Abstraction

Outline

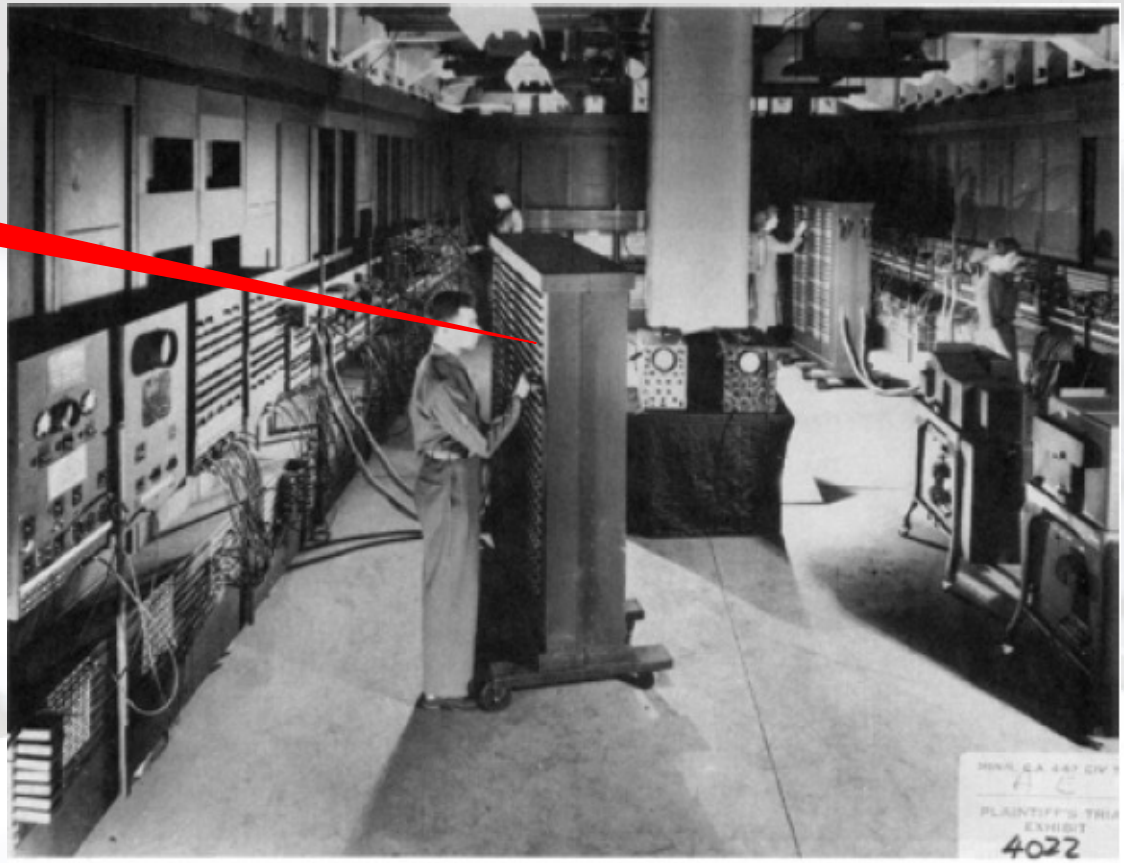
ENIAC - The first electronic computer ,1946年

?



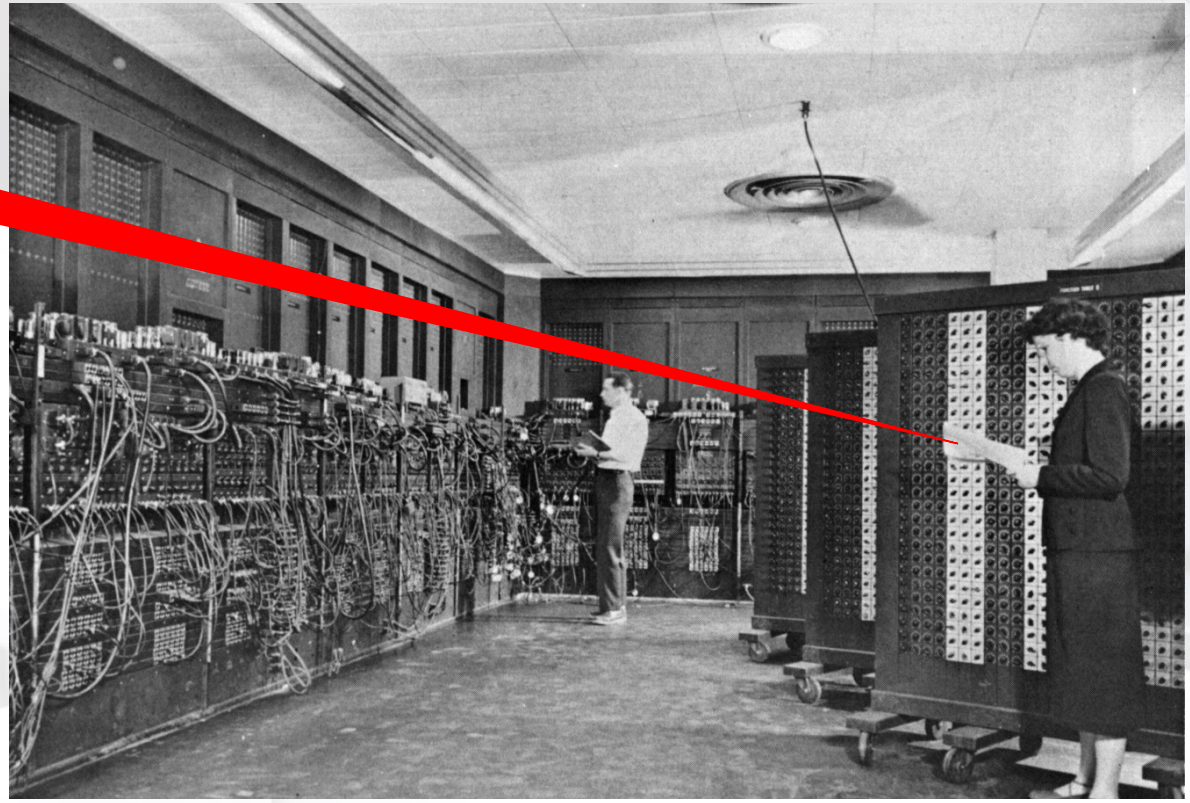
ENIAC - The first electronic computer ,1946年

?



ENIAC - The first electronic computer ,1946年

Programmed by plugboard and switches, time consuming!



Changing the program could take days!

The Origin of the Stored Program Computer

John von Neumann, c. 1955
Credit: Computer History Museum

1944: ENIAC

- Presper Eckert and John Mauchly -- first general electronic computer.
- Hard-wired program -- settings of dials and switches.

1944: Beginnings of EDVAC

- John von Neumann joined ENIAC team and proposed a stored program computer called EDVAC

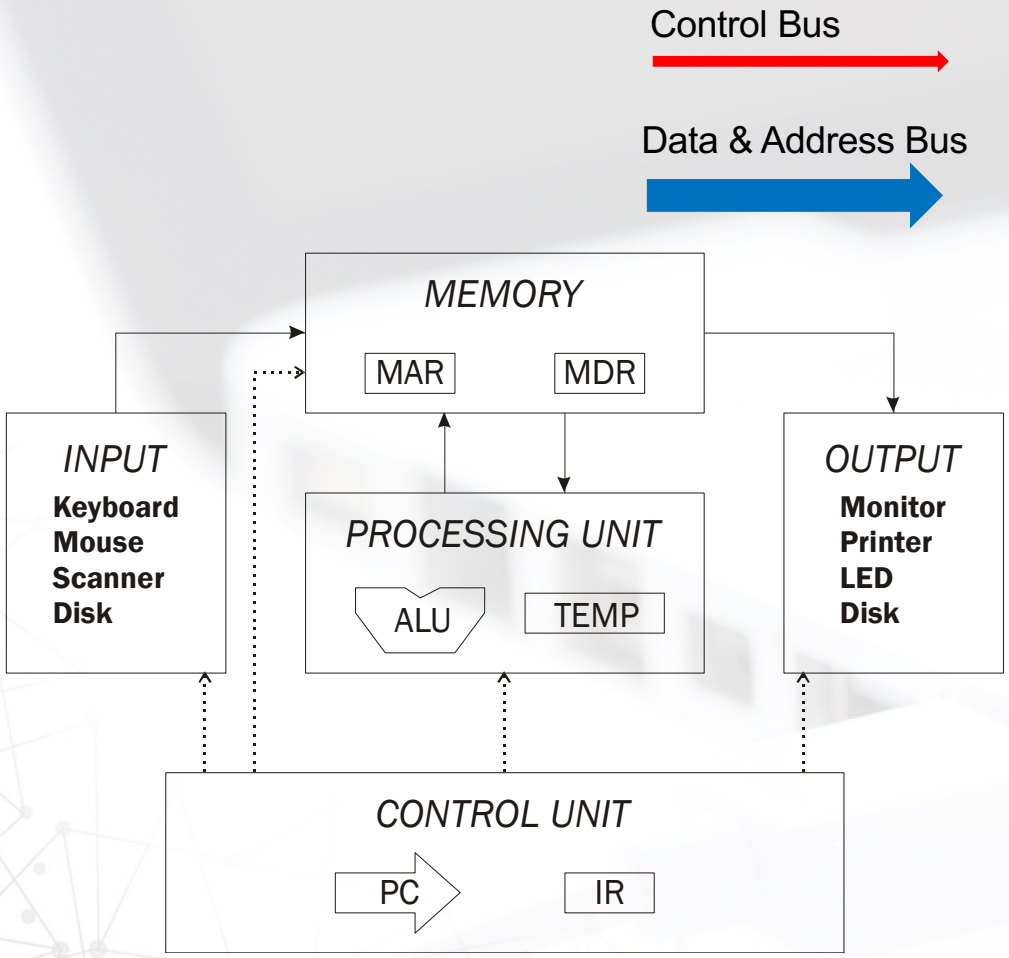
1945: John von Neumann

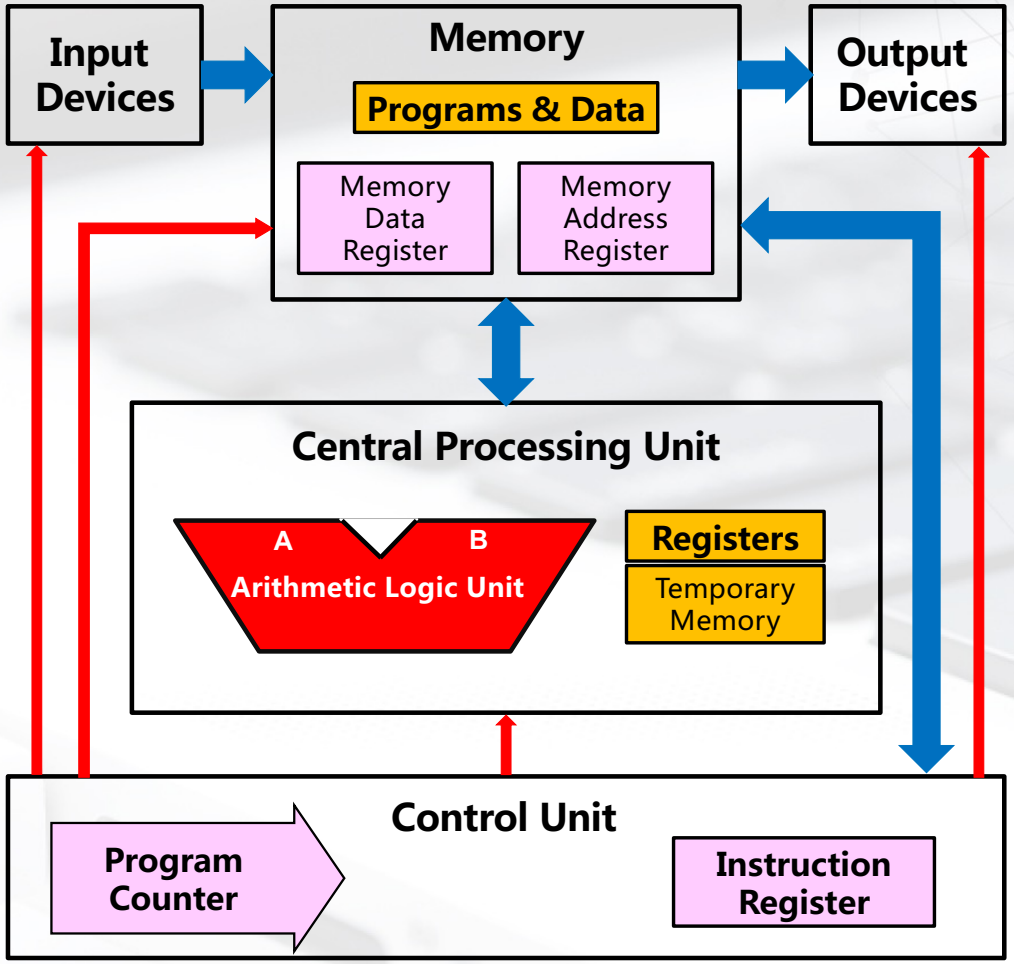
- John von Neumann wrote "First Draft of a Report on the EDVAC" in which he outlined the architecture of a stored-program computer.
- failed to credit designers, ironically still gets credit

The basic structure proposed in the draft became known as the "von Neumann machine" (or model).

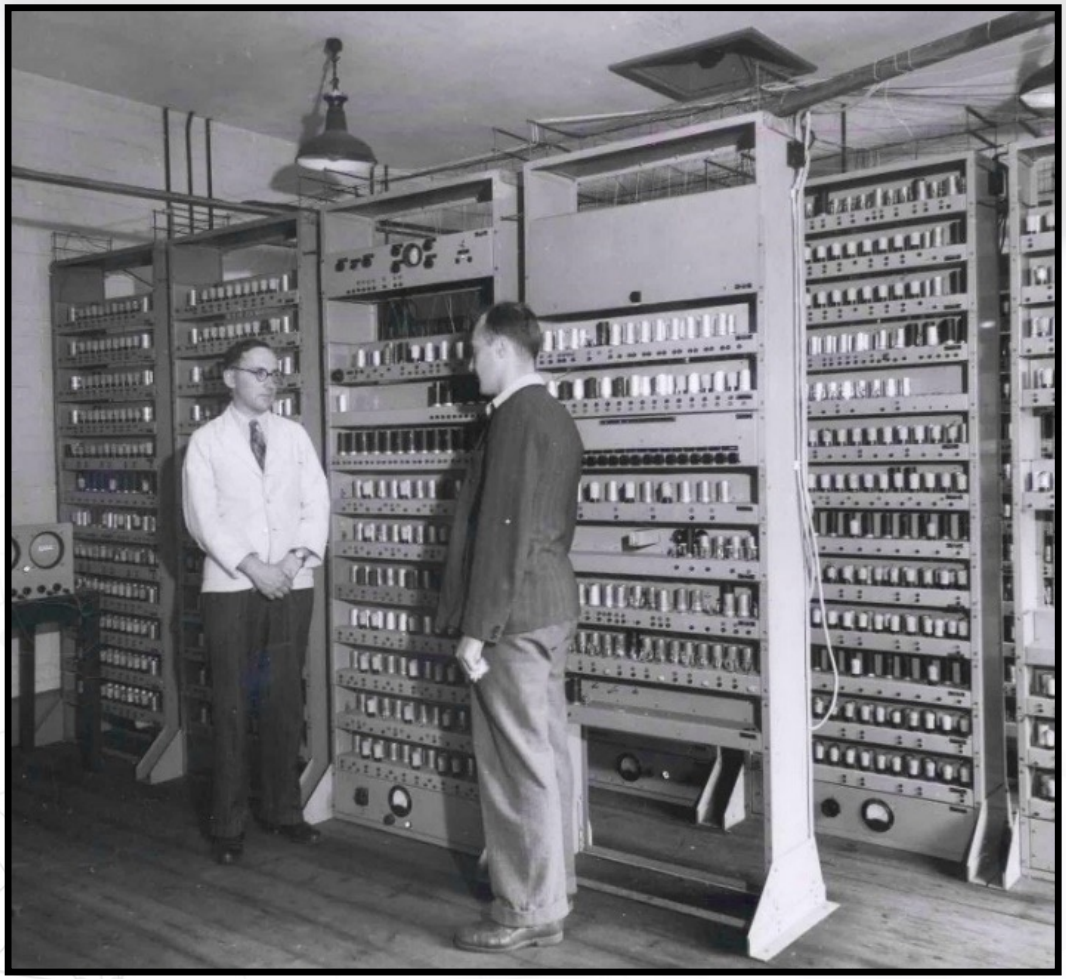
- a *memory*, containing instructions and data
- a *processing unit*, for performing arithmetic and logical operations
- a *control unit*, for interpreting instructions

The Stored Program Computer Architecture (von Neumann Machine Architecture or Model)





The Stored Program Computer



EDSAC
University of Cambridge
UK, 1949

Electronic storage of programming information and data eliminated the need for the more clumsy methods of programming, such as punched paper tape — a concept that has characterized mainstream computer development since 1945.

Maurice Vincent Wilkes

Two major inventions of the microprocessor chip

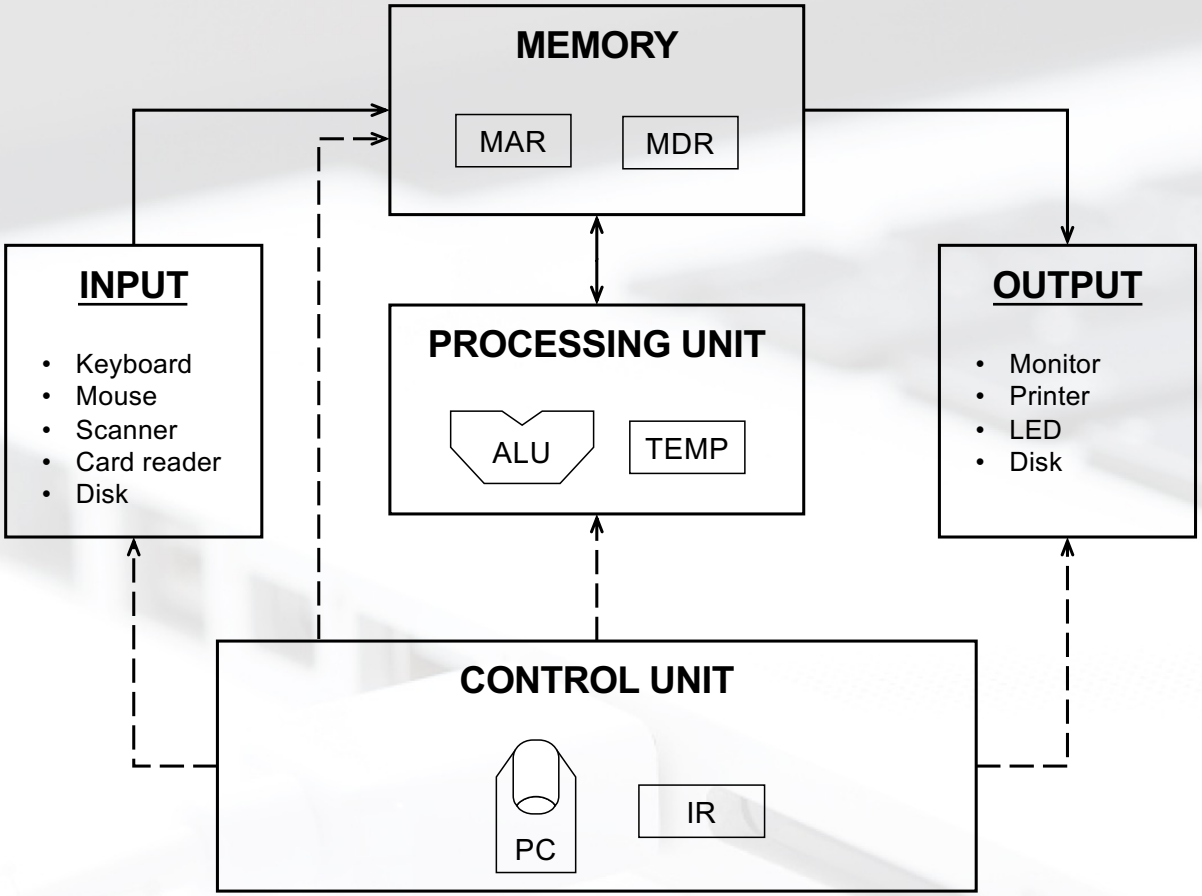
Stored program + **Transistor technology**

Change the program so that you can do all kinds of tasks **on the same hardware**

The device is **smaller** and **faster** than a vacuum tube

Outline

von Neumann Model



Memory

k x *m* array of stored bits (*k* is usually 2^n)

Address

- unique (n-bit) identifier of location

Contents

- m-bit value stored in location

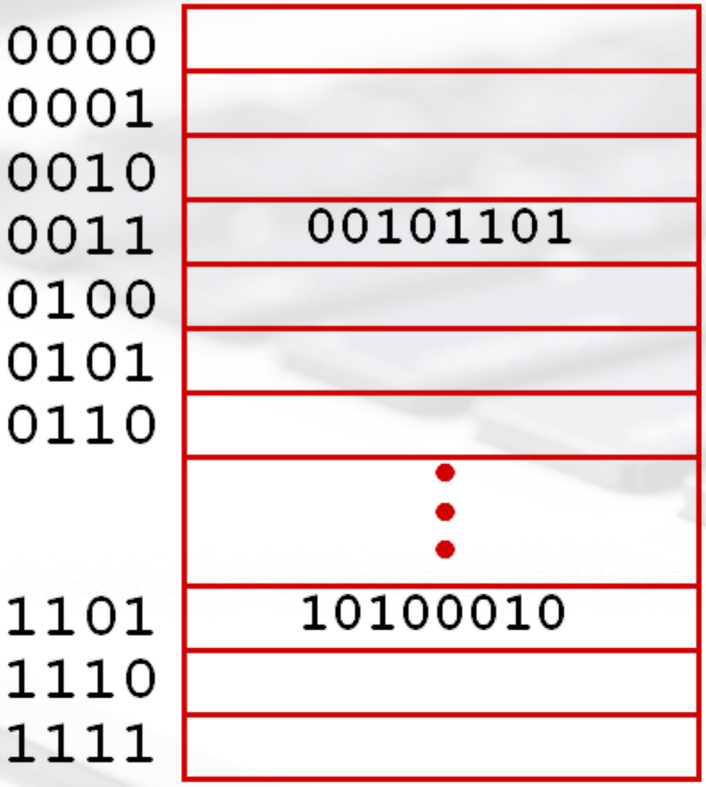
Basic Operations:

LOAD

- read a value from a memory location

STORE

- write a value to a memory location

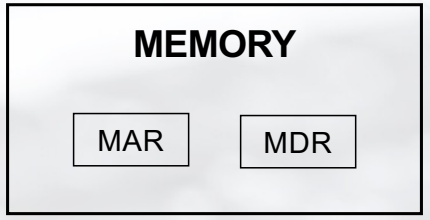


Interface to Memory

How does processing unit get data to/from memory?

MAR: Memory Address Register

MDR: Memory Data Register



To read a location (A):

1. Write the address (A) into the MAR.
2. Send a "read" signal to the memory.
3. Read the data from MDR.

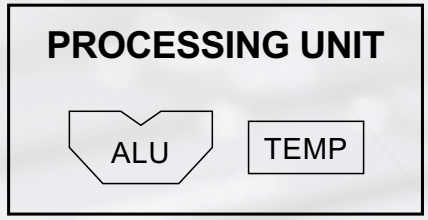
To write a value (X) to a location (A):

1. Write the data (X) to the MDR.
2. Write the address (A) into the MAR.
3. Send a "write" signal to the memory.

Processing Unit

Functional Units

- ALU = Arithmetic and Logic Unit
- could have many functional units. some of them special-purpose (multiply, square root, …)
- LC-3 performs ADD, AND, NOT



Registers

- Small, temporary storage
- Operands and results of functional units
- LC-3 has eight register (R0, …, R7)

Word Size

- number of bits normally processed by ALU in one instruction
- also width of registers
- LC-3 is 16 bits

Input and Output

- Devices for getting data into and out of computer memory
- Each device has its own interface, usually a set of registers like the memory's MAR and MDR
  - `LC-3 supports keyboard (input) and console (output)`
  - `keyboard: data register (KBDR) and status register (KBSR)`
  - `console: data register (CRTDR) and status register (CRTSR)`
  - `frame buffer: memory-mapped pixels`
- Some devices provide both input and output
  - `disk, network`
- Program that controls access to a device is usually called a driver.

INPUT

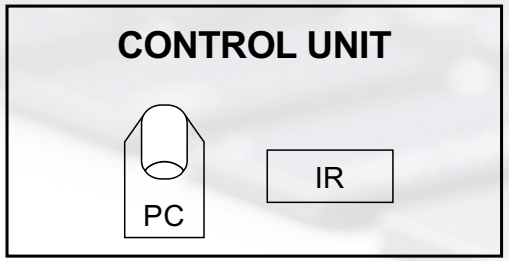
- Keyboard
- Mouse
- Scanner
- Card reader
- Disk

OUTPUT

- Monitor
- Printer
- LED
- Disk

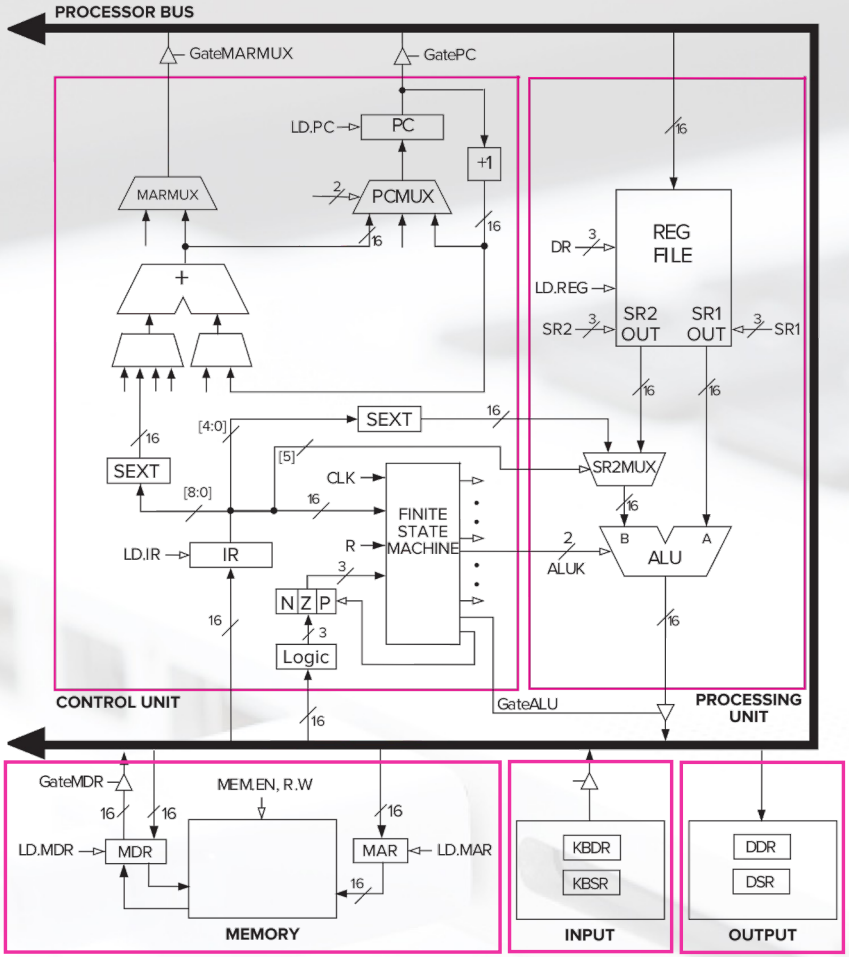
Control Unit

- **Orchestrates execution of the program**
- **Instruction Register (IR) contains the *current instruction*.**
- **Program Counter (PC) contains the *address* of the next instruction to be executed.**
- **Control unit:**
  - **`reads an instruction from memory`**
    - — the instruction's address is in the PC
  - **`interprets the instruction, generating signals that tell the other components what to do`**
    - — an instruction may take many *machine cycles* to complete

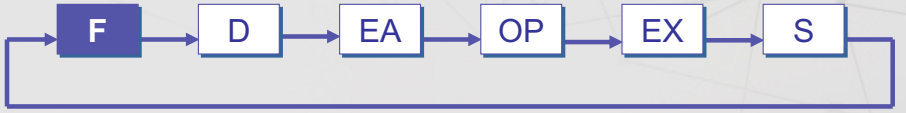


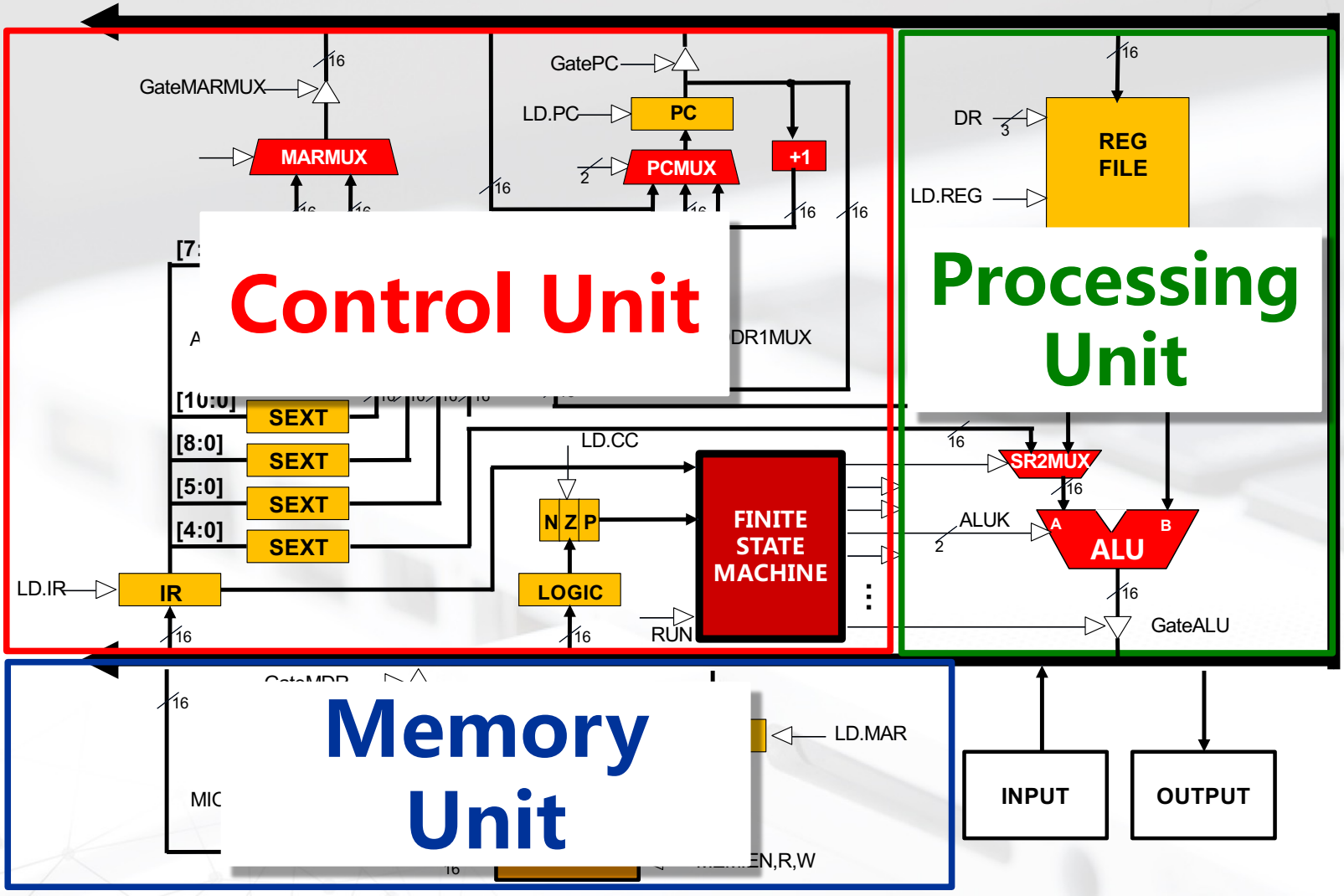
Outline

LC-3 Data Path



LC-3 Data Path





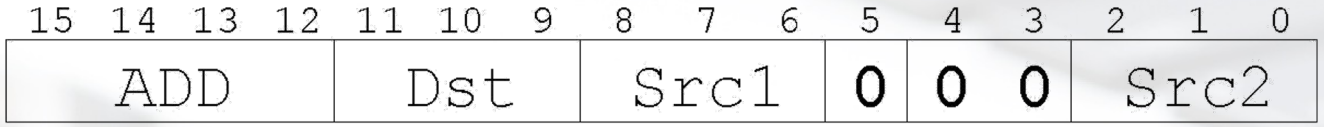
Outline

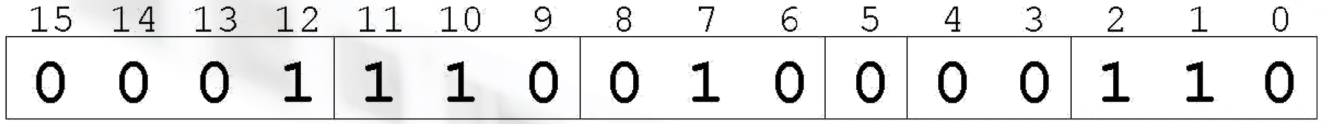
Instruction

- The instruction is the fundamental unit of work.
- Specifies two things:
  - *opcode*: operation to be performed
  - *operands*: data/locations to be used for operation
- An instruction is encoded as a sequence of bits. *(Just like data!)*
  - Often, but not always, instructions have a fixed length, such as 16 or 32 bits.
  - Control unit interprets instruction: generates sequence of control signals to carry out operation.
  - Operation is either executed completely, or not at all.
- A computer's instructions and their formats is known as its *Instruction Set Architecture (ISA).*
  - Persistent ISA invented by UW grad Gene Amdahl (IBM 360)

Example: LC-3 ADD Instruction

- **LC-3 has 16-bit instructions.**
  - Each instruction has a four-bit opcode, bits [15:12].
- **LC-3 has eight *registers* (R0-R7) for temporary storage.**
  - Sources and destination of ADD are registers.

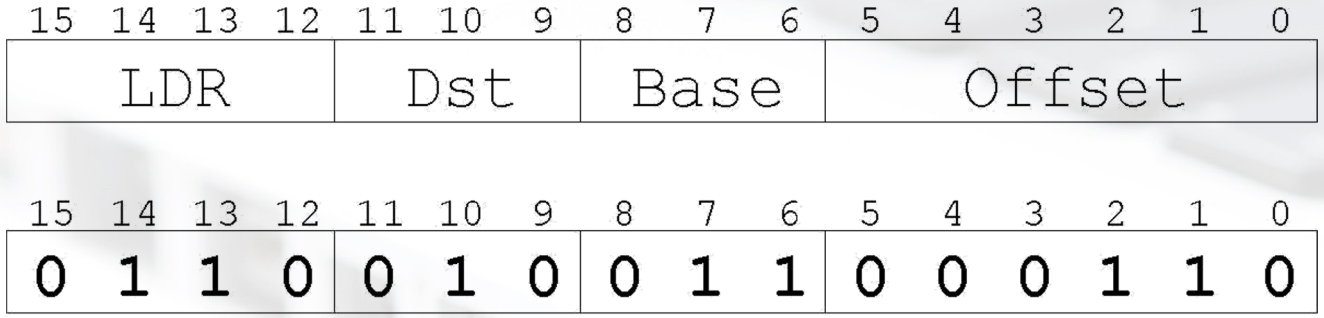




"Add the contents of R2 to the contents of R6, and store the result in R6."

Example: LC-3 LDR Instruction

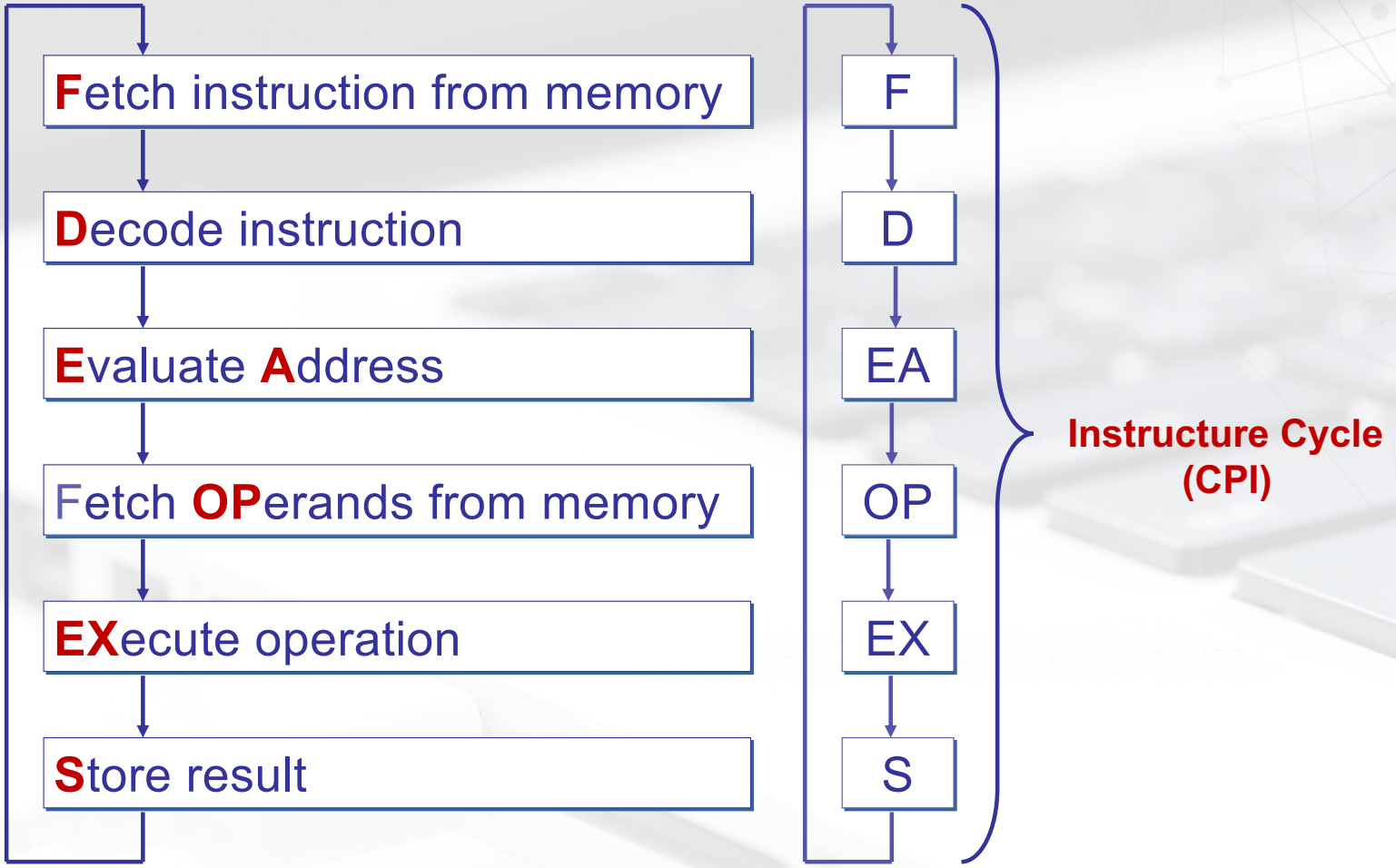
- **Load instruction -- reads data from memory**
- **Base + offset mode:**
  - **`add offset to base register -- result is memory address`**
  - **`load from memory address into destination register`**



"Add the value 6 to the contents of R3 to form a memory address. Load the contents stored in that address to R2."

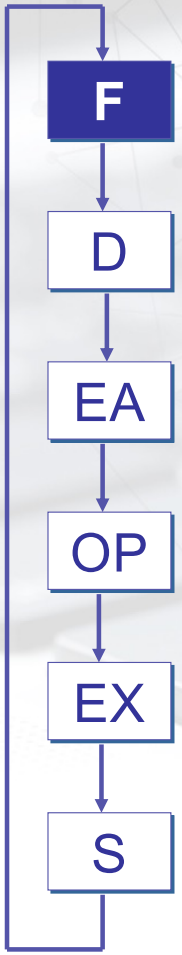
Instruction Processing（State Transition）

Instruction	1
Instruction	2
Instruction	3
.....	
Instruction	n
Instruction	n+1
Instruction	n+2
.....	



Instruction Processing: FETCH

- Load next instruction (at address stored in PC) from memory into Instruction Register (IR).
  - Load contents of PC into MAR.
  - Send "read" signal to memory.
  - Read contents of MDR, store in IR.
- Then increment PC, so that it points to the next instruction in sequence.
  - PC becomes PC+1.



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD				Dst			Src1			0	0	0	Src2		

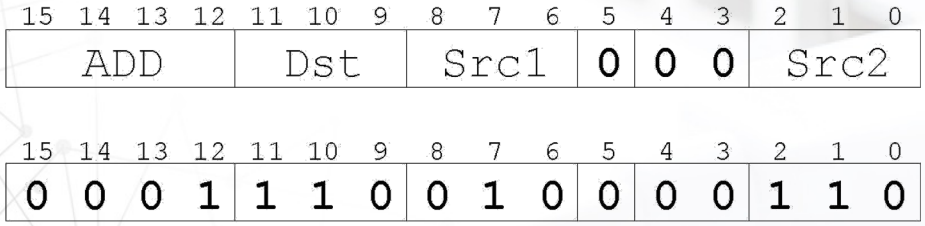
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0

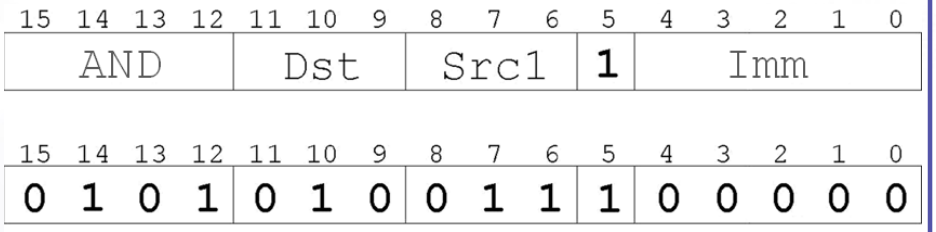
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AND				Dst			Src1			1	Imm				

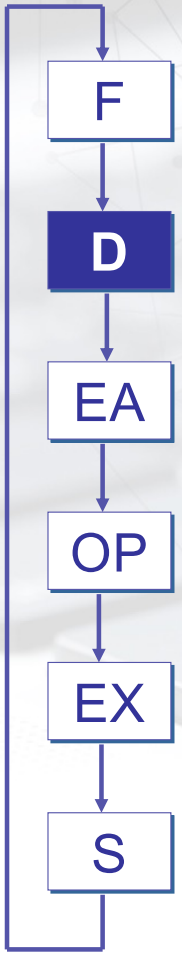
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0

Instruction Processing: DECODE

- **First identify the opcode.**
  - **In LC-3, this is always the first four bits of instruction.**
  - **A 4-to-16 decoder asserts a control line corresponding to the desired opcode.**
- **Depending on opcode, identify other operands from the remaining bits.**
  - **Example:**
    - for ADD, last three bits is source operand #2
    - for LDR, last six bits is offset

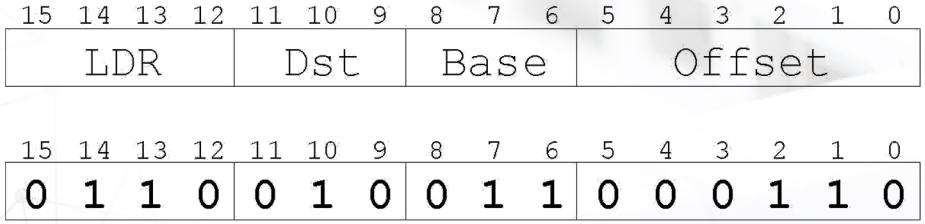


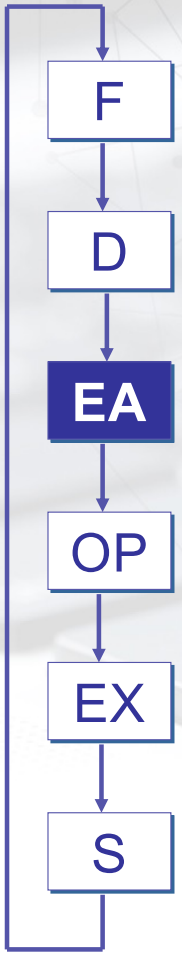




Instruction Processing: EVALUATE ADDRESS

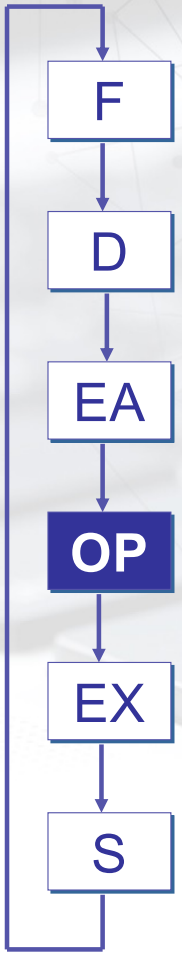
- For instructions that require memory access, compute address used for access.
- Examples:
  - add offset to base register (as in LDR)
  - add offset to PC (or to part of PC)
  - add offset to zero





Instruction Processing: FETCH OPERANDS

- Obtain source operands needed to perform operation.
- Examples:
  - read data from register file (ADD)
  - load data from memory (LDR)



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD				Dst			Src1			0	0	0	Src2		

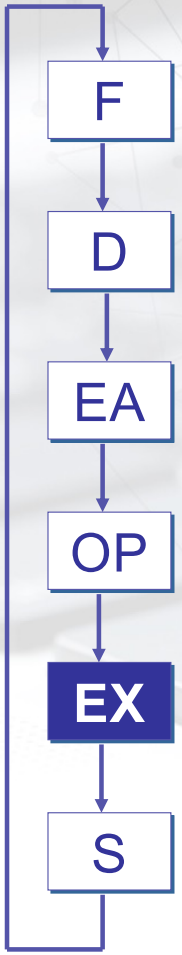
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0

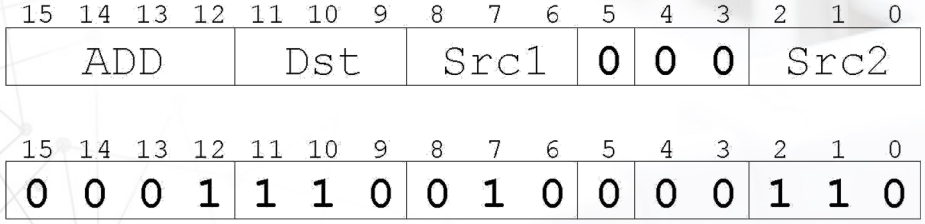
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AND				Dst			Src1			1	Imm				

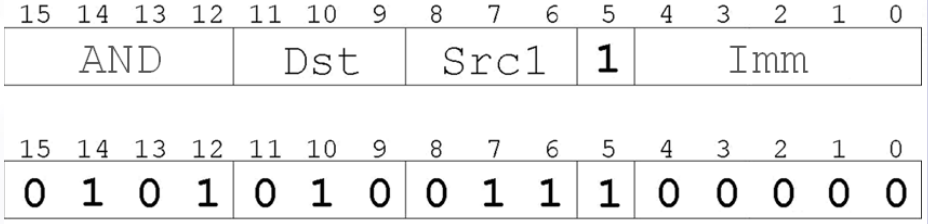
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0

Instruction Processing: EXECUTE

- Perform the operation, using the source operands.
- Examples:
  - send operands to ALU and assert ADD signal
  - do nothing (e.g., for loads and stores)

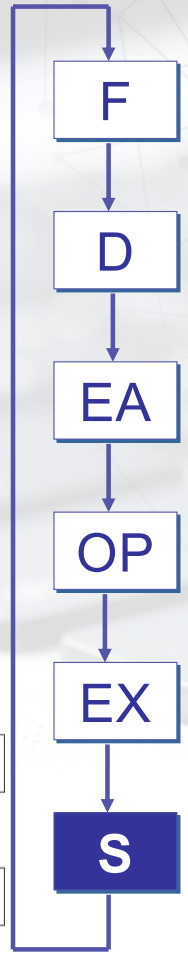






Instruction Processing: STORE

- **Write results to destination. (register or memory)**
- **Examples:**
  - **result of ADD is placed in destination register**
  - **result of memory load is placed in destination register**
  - **for store instruction, data is stored to memory**
    - write address to MAR, data to MDR
    - assert WRITE signal to memory



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD				Dst			Src1			0	0	0	Src2		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AND				Dst			Src1			1	Imm				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0

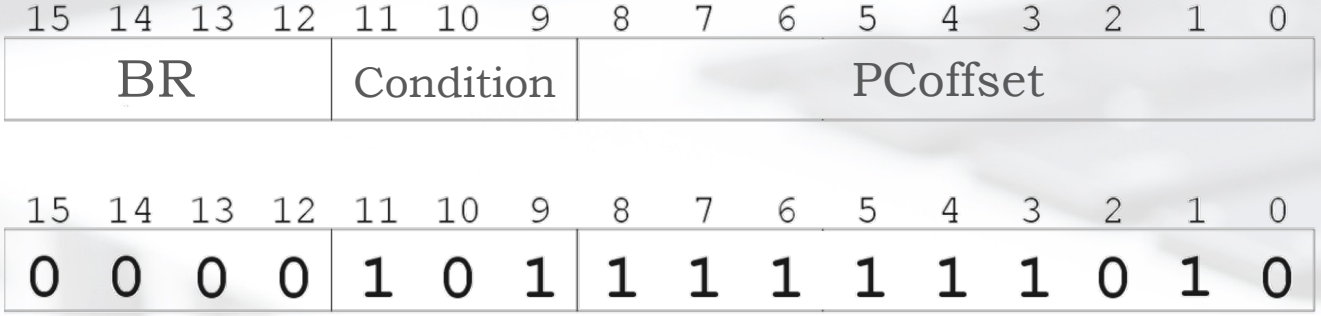
Changing the Sequence of Instructions

- In the FETCH phase, we incremented the Program Counter by **1.**
- What if we don’ t want to always execute the instruction that follows this one?
  - `examples: loop, if-then-else, function call`

  --Need special instructions that change the contents of the PC.
- These are called ***jumps*** and ***branches***.
  - ***jumps*** `are unconditional -- they always change the PC`
  - ***branches*** `are conditional -- they change the PC only if some condition is true (e.g., the contents of a register is zero)`

Example: LC-3 BR Instruction

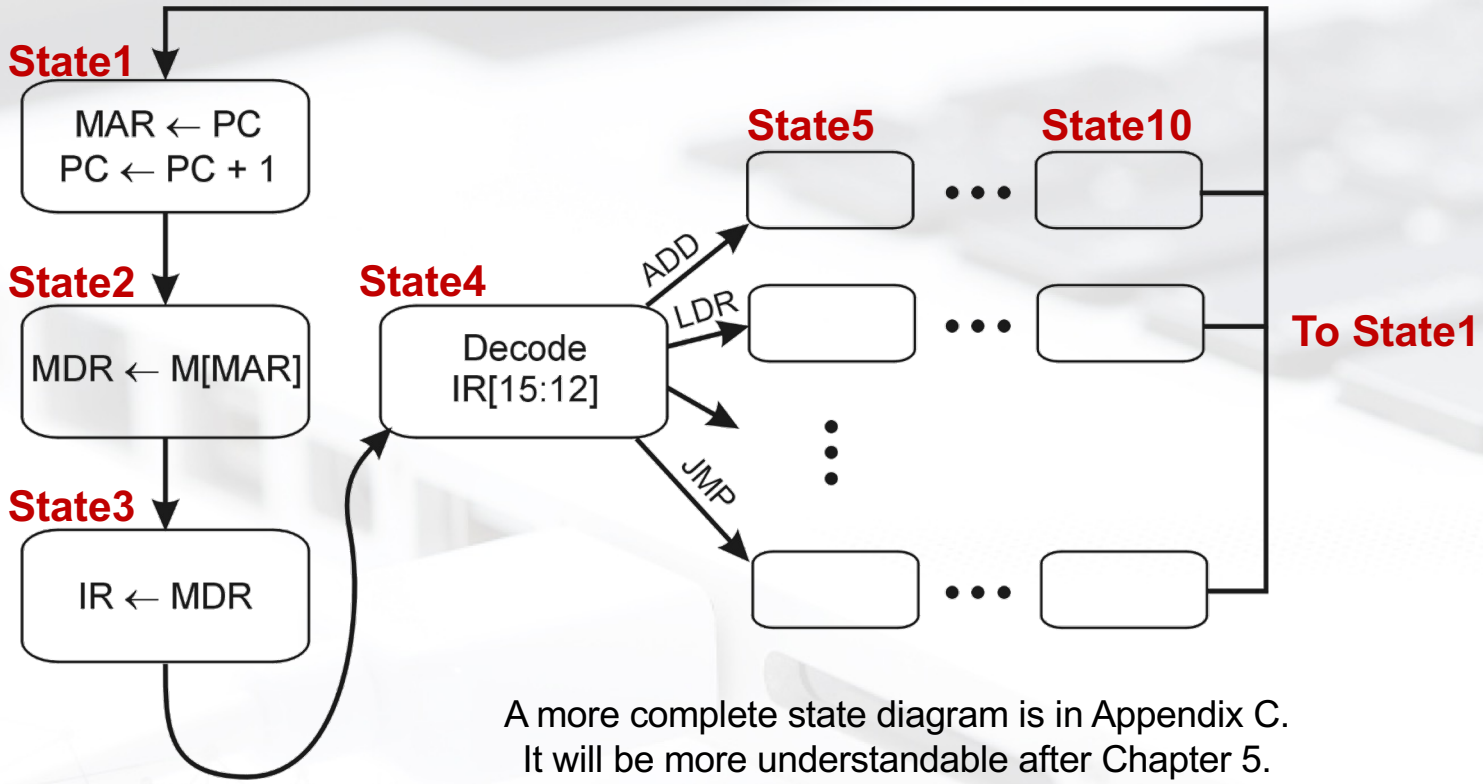
- Set the PC to the value PC+PCoffset. This becomes the address of the next instruction to fetch.



"Load the contents of (PC + PCoffset) into the PC."

Control of the Instruction Cycle

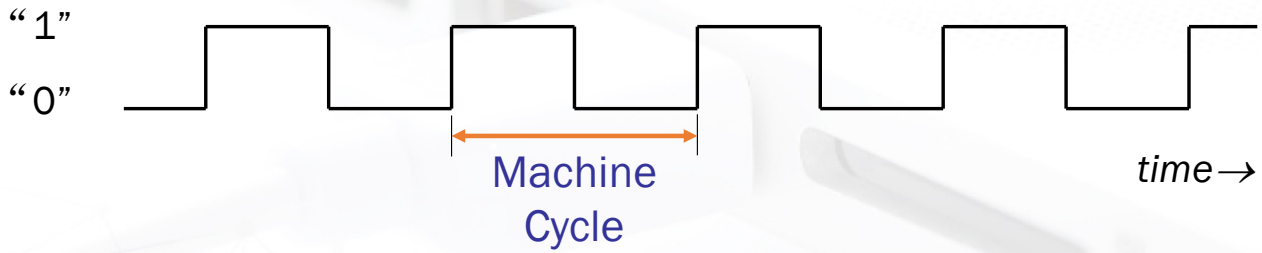
- The control unit is a state machine. Here is part of a simplified state diagram for the LC-3:



A more complete state diagram is in Appendix C.
It will be more understandable after Chapter 5.

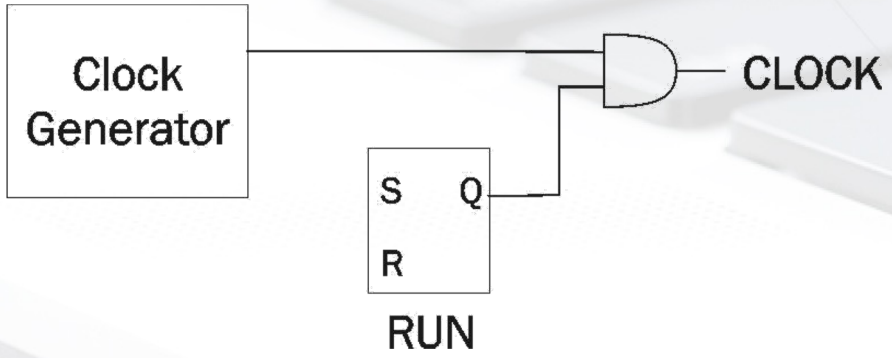
Driving Force: The Clock

- The clock is a signal that keeps the control unit moving.
  - At each clock "tick," control unit moves to the next machine cycle
    -- may be next instruction or next phase of current instruction.
- Clock generator circuit:
  - Based on crystal oscillator
  - Generates regular sequence of "0" and "1" logic levels
  - *Clock Cycle (or Machine Cycle)* -- rising edge to rising edge



Halting the Computer: Stopping the Clock(by TRAP Instruction)

- **Control unit will repeat instruction processing sequence as long as clock is running.**
  - If not processing instructions from your application, then it is processing instructions from the Operating System (OS).
  - The OS is a special program that manages processor and other resources.
- **To stop the computer**
  - AND the clock generator signal with ZERO
  - when control unit stops seeing the CLOCK signal, it stops processing



Outline

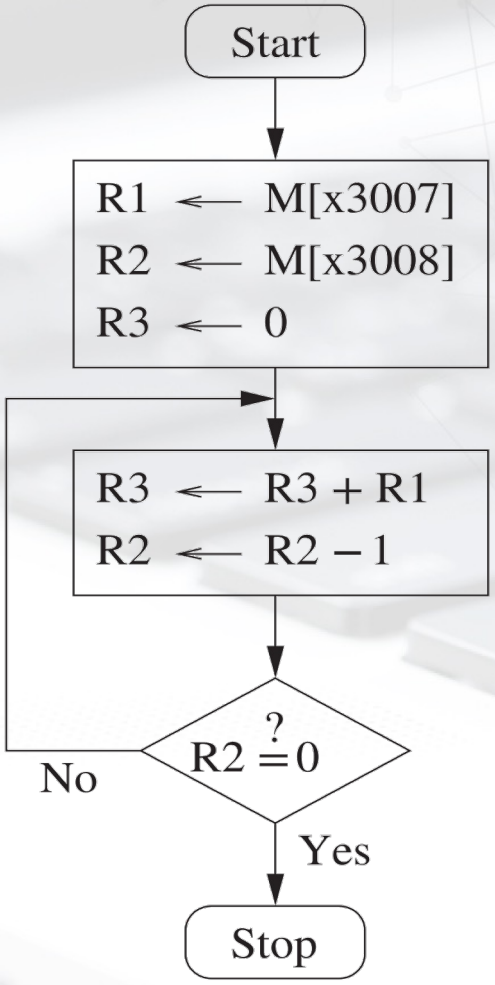
An algorithm for 5 x 4

- We had ADD, AND, LD, BR, HALT(TRAP)
- We had ADD instructions, but did not have multiply instructions. So, we do

$$5 \times 4 = 5+5+5+5$$

M[x3007]=5

M[x3008]=4

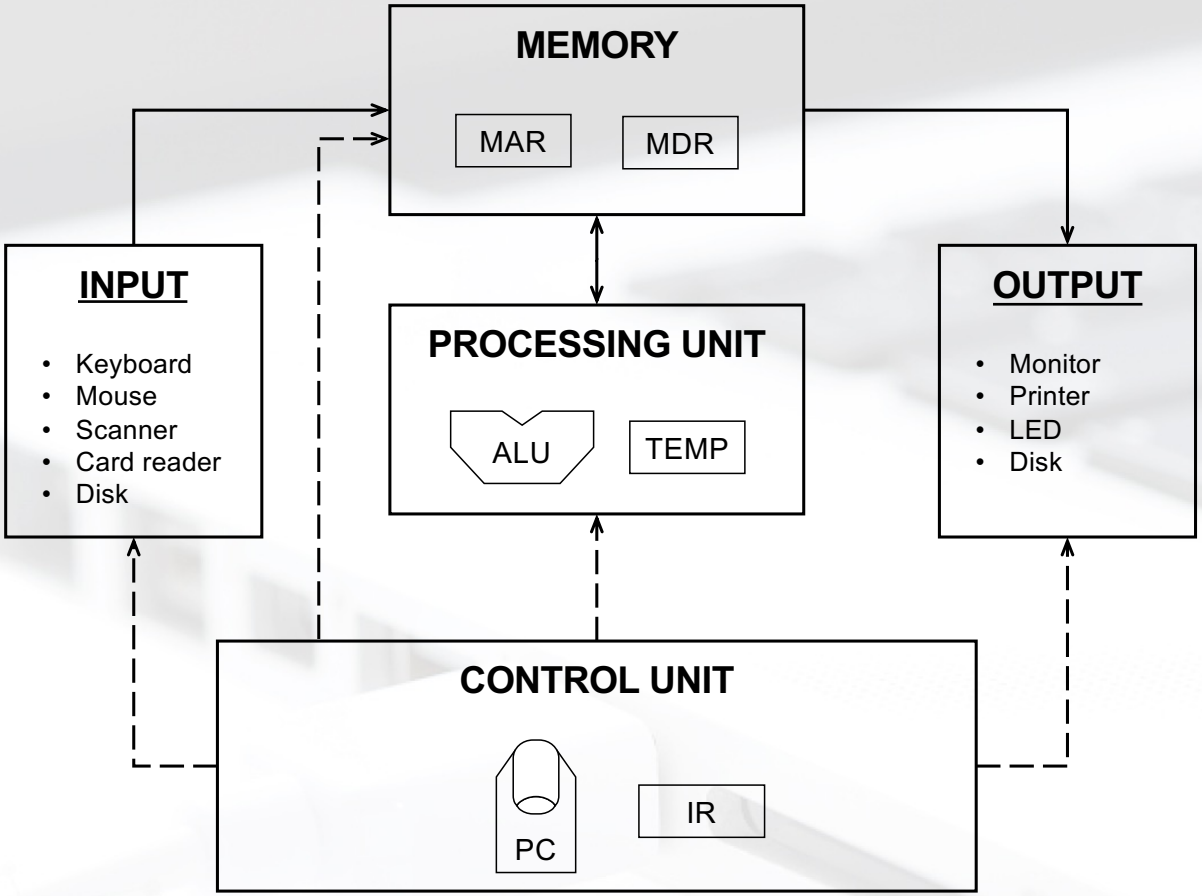


A program that multiplies without a multiply instruction

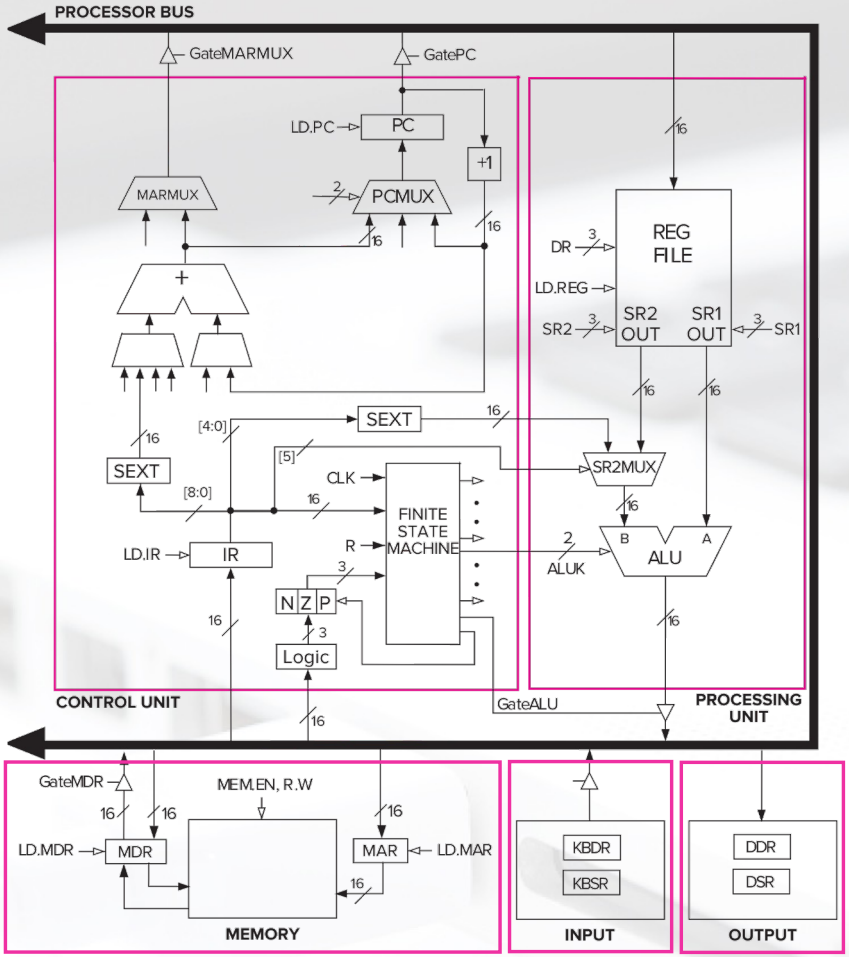
Address	*Instruction*	*Comments*
x3000	0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0	*R1 ← M[x3007]*
x3001	0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 0	*R2 ← M[x3008]*
x3002	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	*R3 ← 0*
x3003	0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 0	*R3 ← R3+R1*
x3004	0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1	*R2 ← R2-1*
x3005	0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 1	*BR not zero M[x3003]*
x3006	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1	*HALT*
x3007	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1	*The value 5*
X3008	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	*The value 4*

Outline

Von Neumann Model



LC-3 Data Path



Instruction Processing Summary

- Instructions look just like data -- it's all interpretation.
- Three basic kinds of instructions:
  - computational instructions (ADD, AND, …)
  - data movement instructions (LD, ST, …)
  - control instructions (JMP, BRnz, …)
- Six basic phases of instruction processing:
  - not all phases are needed by every instruction
  - phases may take variable number of machine cycles