

Computer Science 210 s1c  
**Computer Systems 1**  
2008 Semester 1  
Lecture Notes

Lecture 10, 27Mar08:

# The von Neumann Computer

*James Goodman*



Credits: Slides prepared by Gregory T. Byrd, North Carolina State University

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## The Stored Program Computer

### 1943: ENIAC

- Presper Eckert and John Mauchly – first general electronic computer.  
(or was it John V. Atanasoff in 1939?)
- Hard-wired program – settings of dials and switches.

### 1944: Beginnings of EDVAC

- Among other improvements, includes program stored in memory

### 1945: John von Neumann

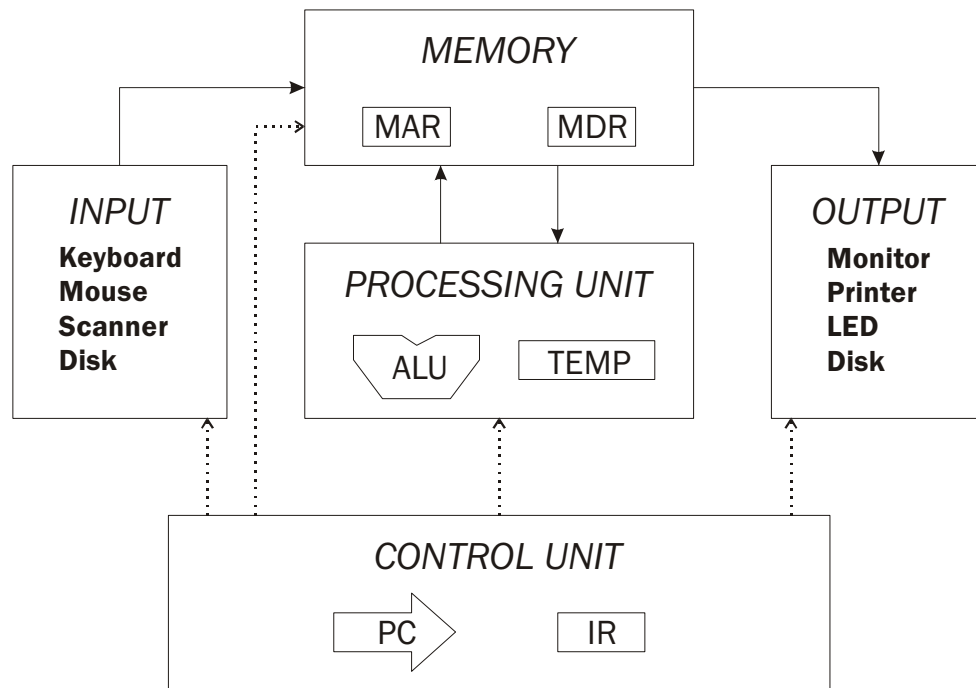
- Wrote a report on the stored program concept,  
known as the *First Draft of a Report on EDVAC*

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

For more history, see <http://www.maxmon.com/history.htm>

# Von Neumann Model



30-Mar-08

CS210

167

## Memory

$2^k \times m$  array of stored bits

Address

- unique ( $k$ -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

0000	
0001	
0010	
0011	00101101
0100	
0101	
0110	
	⋮
1101	10100010
1110	
1111	

30-Mar-08

CS210

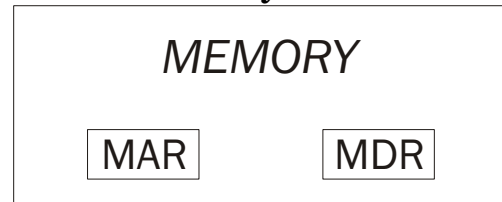
168

## Interface to Memory

How does processing unit get data to/from memory?

**MAR:** Memory Address Register

**MDR:** Memory Data Register



To **LOAD** 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.

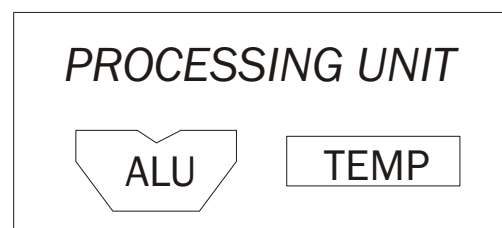
To **STORE** a value (X) to a location (A):

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 registers (R0, ..., R7), each 16 bits wide

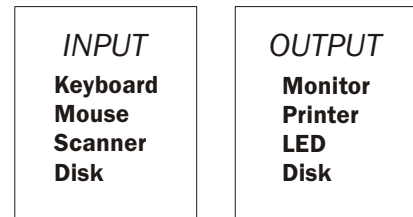
### Word Size

- 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 monitor (output)
- keyboard: data register (KBDR) and status register (KBSR)
- monitor: data register (DDR) and status register (DSR)

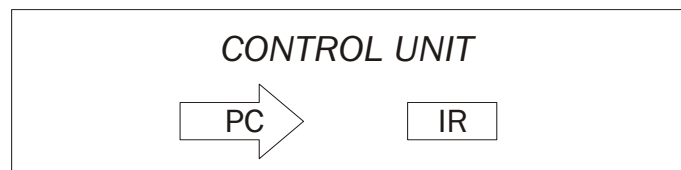
Some devices provide both input and output

- disk, network

Program that controls access to a device is usually called a *driver*.

# Control Unit

Orchestrates execution of the program



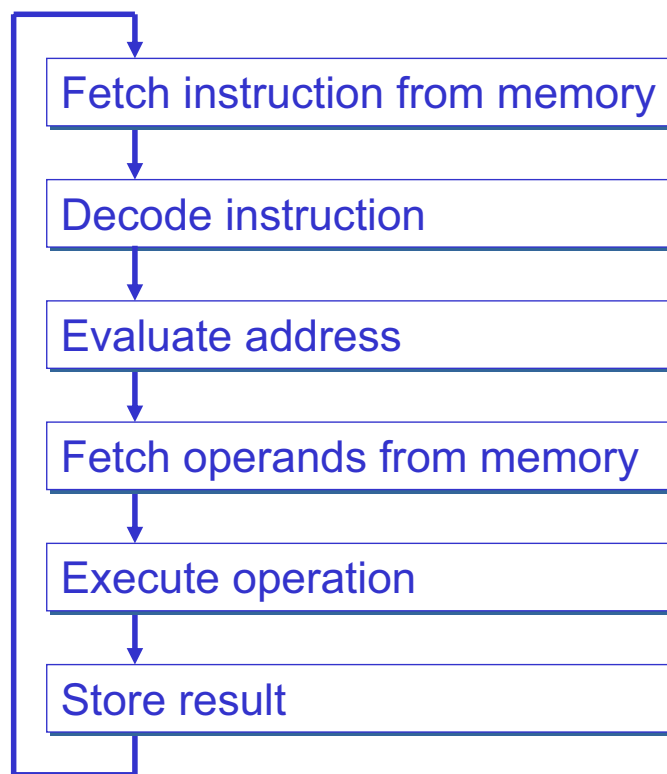
**Instruction Register** (IR) contains the *current instruction*.

**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

# Instruction Processing



30-Mar-08

CS210

173

## 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)***.

30-Mar-08

CS210

174

## 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.

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

*“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

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

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

*“Add the value 6 to the contents of R3 to form a memory address. Load the contents of that memory location to R2.”*

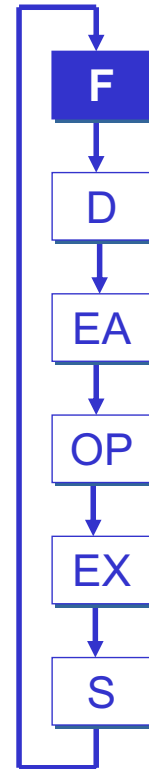
## Instruction Processing: FETCH

Load next instruction (at address stored in PC) from memory into Instruction Register (IR).

- Copy contents of PC into MAR.
- Send “read” signal to memory.
- Copy contents of MDR into IR.

Then increment PC, so that it points to the next instruction in sequence.

- PC becomes PC+1.



30-Mar-08

CS210

177

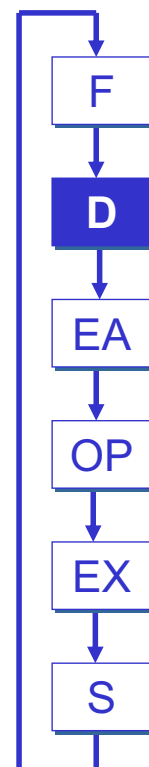
## Instruction Processing: DECODE

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 LDR, last six bits is offset
  - for ADD, last three bits is source operand #2



30-Mar-08

CS210

178

## 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
- add offset to zero

