

Chapter 4

The Von Neumann Model

Based on slides © McGraw-Hill
 Additional material © 2004/2005 Lewis/Martin
 Modified by Diana Palsetia

Instructing the Computer

To perform a task, the computer is provided with **program**

Program

- Consists of a **set of instructions** that the computer must do to complete the task

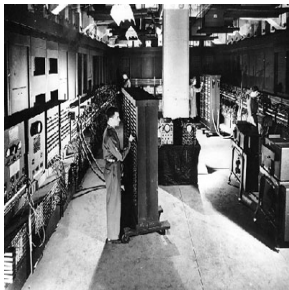
Instruction

- **Smallest piece of work specified (fundamental unit)**
 - One high-level statement in Java/C can be more than one instruction for the machine
- **Either carried completely or not at all**
 - This is because future instructions can be dependent on the previous

How does the computer get the instructions?

■ Before stored program concept

- Plugboards (flip bits manually)



Copyright: ENIAC

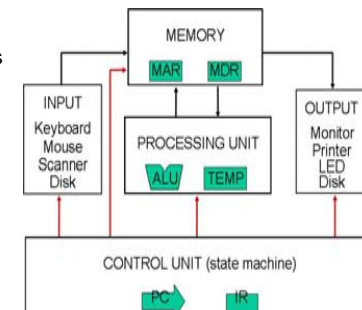
■ Stored program concept

- Program is stored in memory and computer then will get the instructions one-by-one and process them.
- This eliminated physical feeding of instructions
- Proposed by J. Mauchly & J. Eckert
- Popularized by Von Nuemann

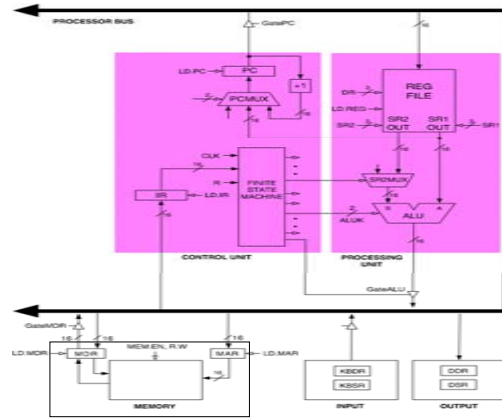
Von Nuemann Model(stored program concept)

■ Components to process a program:

- **Memory:** where instructions and data are stored
- **Control unit:**
 - co-ordinates all other units
 - It also interprets instructions
- **Processing unit:** for performing arithmetic and logical operations
- **Input/Output units:** for interacting with real world



Example of von Nuemann Model: LC-3 (Little Computer- 3)



CIT 593

5/29

Memory

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

Address

- unique (k -bit) identifier of location

Contents

- m -bit value stored in location

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

Basic Operations:

LOAD

- read a value from a memory location

STORE

- write a value to a memory location

CIT 593

6/29

Interface to Memory

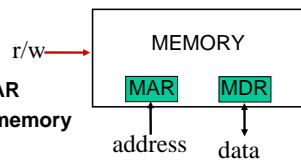
How does processing unit get data to/from memory?

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 (r)" signal to the memory
3. Read the data from MDR



To write a value X to a location A

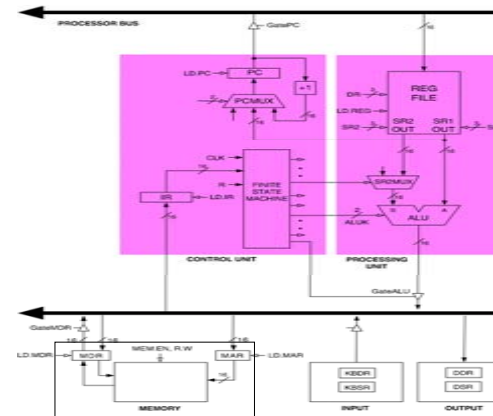
1. Write the data X to the MDR
2. Write the address A into the MAR
3. Send a "write (w)" signal to the memory

So what is the need for the interface ?

CIT 593

7/29

Why have memory Interface ?



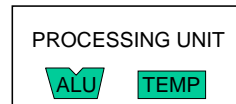
CIT 593

8/29

Processing Unit

Functional Units

- ALU = Arithmetic and Logic Unit
- Could have many functional units (some special-purpose, e.g., multiply, square root, ...)
- LC-3: ADD, AND, NOT



Registers

- Small, temporary storage
- Operands (data to be operated on) and results of functional units
 - LC-3: eight user register (R0, ..., R7)
- Some more for book keeping...more later

Word Size

- Number of bits normally processed by ALU in one instruction
- Also width of registers
- LC-3: 16 bits

CIT 593

9/29

Control Unit

Orchestrates execution of the program (like your Brain)

- Keeps track of where we are in the process of executing
 - The program as well as each instruction

Instruction Register (IR)

- Contains the current instruction

Program Counter (PC)

- Contains the address of the next instruction to execute

Control Unit (Giant State Machine)

- Reads an instruction from memory (at PC)
- Interprets the instruction
- Generates signals that tell the other components what to do
- Instruction may take many *machine cycles* to complete



CIT 593

10/29

Instructions

Instruction: *Fundamental unit of work*

Constituents

- **Opcode**: operation to be performed (e.g. add, and)
- **Operands**: data/locations to be used for operation
 - Source: location that contains the data
 - Destination: location that will store the result of computation
 - Immediate: data values not contained at a particular location

Encoded as a sequence of bits (*just like data!*)

- Sometimes have a fixed length (e.g., 16 or 32 bits)
- Control unit interprets instruction
 - Generates control signals to carry out operation
 - Atomic: operation is either executed completely, or not at all

CIT 593

11/29

Instruction Set Architecture (ISA)

ISA = Programmer-visible components & operations

- Memory organization
 - Address space - # of locations ?
 - Addressability - how many bits per location?
- Register set
 - How many? What size? How are they used?
- Instruction set
 - Opcodes : Which operation ?
 - Data types : int, floating point etc
 - Addressing modes: how to calculate the effective memory address of an operand ?

All information needed to write/generate machine language program

- You care if you write e.g. compilers for high-level language

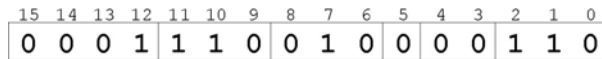
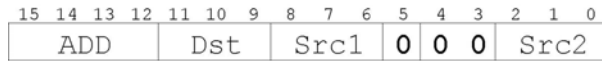
CIT 593

12/29

Example

LC-3 has 16-bit instructions

- Each instruction has a **four-bit opcode**, bits [15:12]
- Has **eight registers (R0-R7)** for temporary storage



LC-3 ADD : Add the contents of R2 to the contents of R6, and store the result in R6.

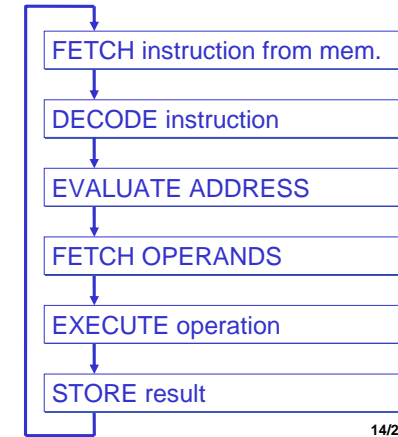
CIT 593

13/29

Instruction Processing

How are instructions executed?

- Each instruction goes to 6 stages
- Some instructions may not use every stage



CIT 593

14/29

Instruction Processing: FETCH

Idea

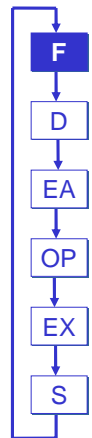
- Put next instruction in IR & increment PC

Steps

- Load contents of PC into MAR
- Increment PC
- Send "read" signal to memory
- Read contents of MDR, store in IR

Who makes all this happen?

- Control unit



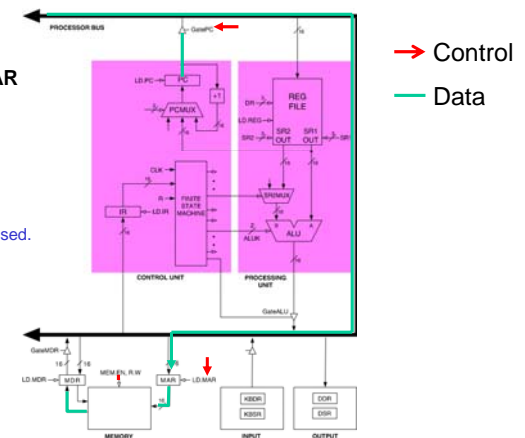
CIT 593

15/29

Example: FETCH in LC-3

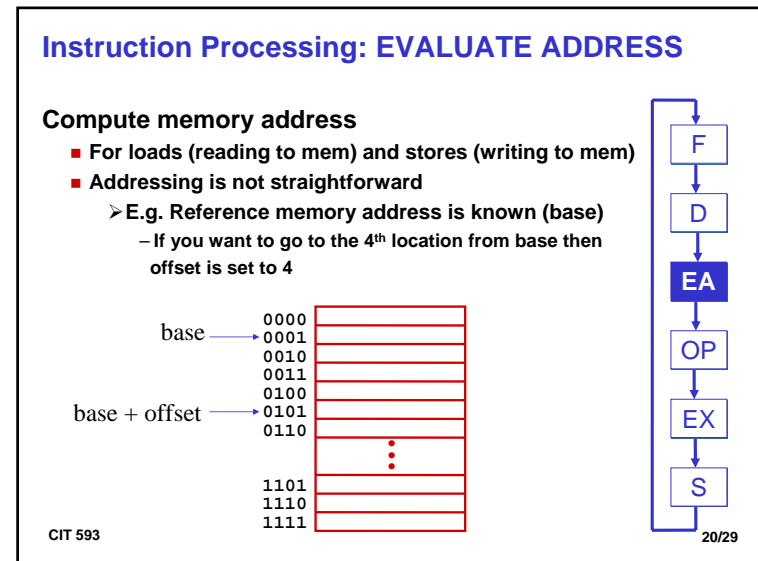
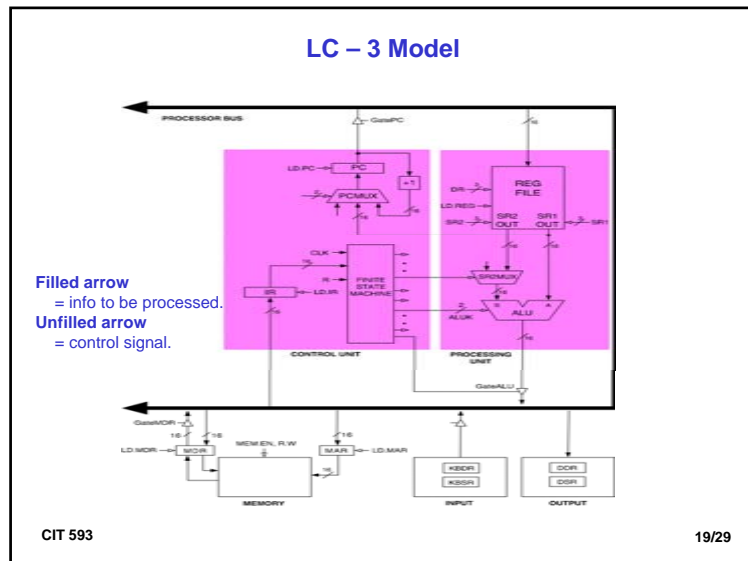
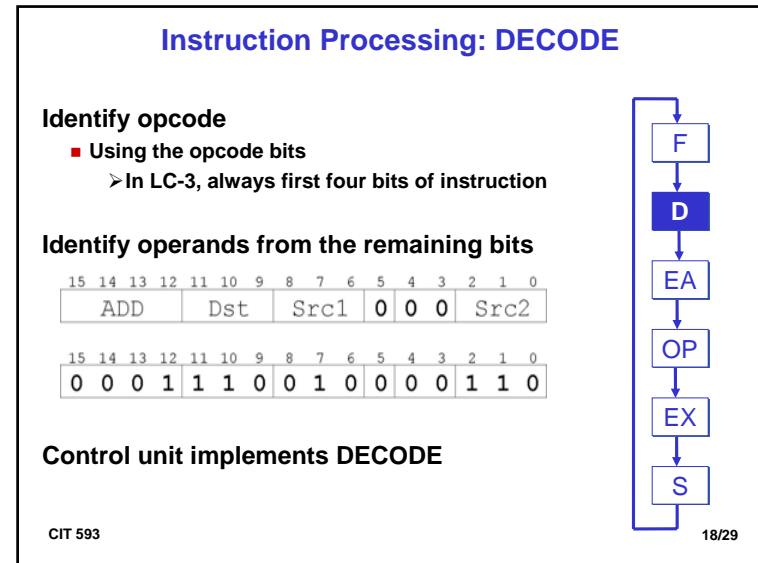
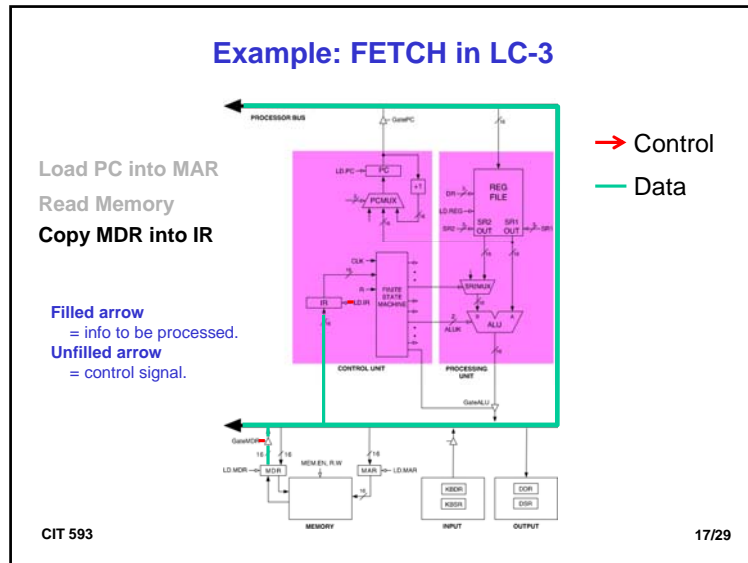
Load PC into MAR
Read Memory

Filled arrow = info to be processed.
Unfilled arrow = control signal.



CIT 593

16/29

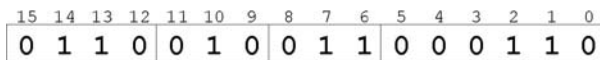
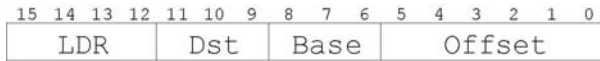


Example: LC-3 LDR Instruction

Reads data from memory

Base + offset addressing mode

- Add offset to base register to produce 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 of memory at that address and place the resulting data in R2.”

CIT 593

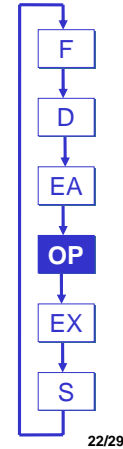
21/29

Instruction Processing: OPERAND fetch

Get source operands for operation

Examples

- Read data from register file (e.g. for an ADD)



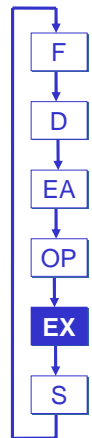
CIT 593

22/29

Instruction Processing: EXECUTE

Actually perform operation

- ALU receives the operands
- E.g. ADD/SUB, AND, OR, NOT, MULT
- Do nothing (e.g., for loads(reads) and stores(writes))



CIT 593

23/29

Instruction Processing: STORE

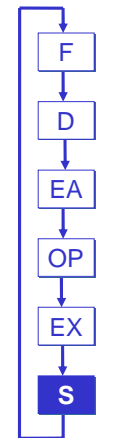
Write results to destination

- Register or memory

Example

- Result of ADD is placed in destination reg.
- If we have limited space
 - Some values from registers are written back to memory via STORE command

Note: The store command will set MDR and assert WRITE signal to memory



CIT 593

24/29

Changing the Sequence of Instructions

Increment of PC

- FETCH phase always increments PC by 1 instruction
- But we can also increment PC by the number of instructions we skipped in certain conditions

We can also skip instructions:

- Programming constructs that change the sequence of instruction execution like *If-then*, *loops* etc.
 - E.g: *If (Overall points equals 90) then Grade equals A*

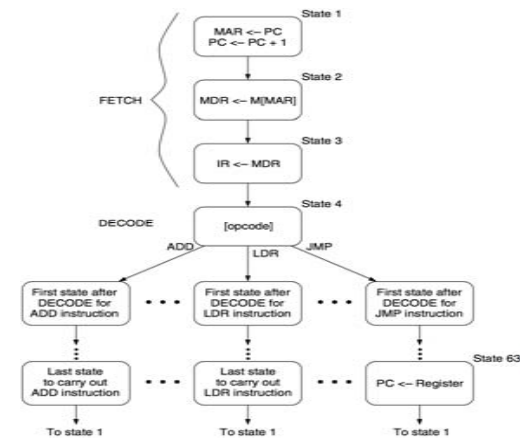
Instructions that change PC other than +1 in assembly:

- **Branches** are conditional
 - Change the PC only if some condition is true e.g., the contents of a register is zero
- **Jumps** are unconditional -> Always change the PC

CIT 593

25/29

Control Unit State Diagram



CIT 593

26/29

Stopping the Computer

How does the computer know its done after x number instructions?

- e.g. $A*B+C - 3 \text{ loads} + 1 \text{ MULT} + 1 \text{ ADD} = 5 \text{ total instructions}$

Older Machines

- Done by a **HALT** instruction

Modern machines

- User programs execute under the control of O.S.
 - Once the program terminates, the control instruction changes PC to again start O.S.

CIT 593

27/29

Instruction Processing Summary

Instructions look just like data

- Interpreted by machine

Three basic kinds of instructions

- Computational instructions (ADD, AND, ...)
- Data movement instructions (LD(load), ST(store), ...)
- Control instructions (JMP, BRnz, ...)

Six basic phases of instruction processing

$F \rightarrow D \rightarrow EA \rightarrow OP \rightarrow EX \rightarrow S$

- Not all phases are needed by every instruction
- Phases may take variable number of machine cycles
- Multiple phases per cycle possible

CIT 593

28/29