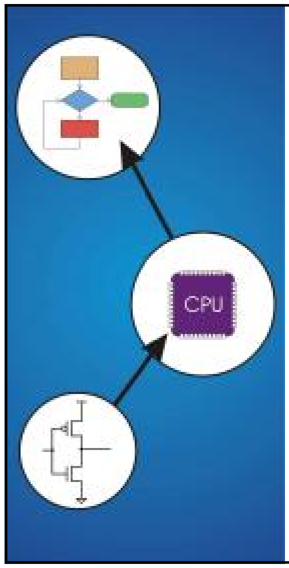


Chapter 4
The Von Neumann Model

The Stored Program Computer

1943: ENIAC

- **Presper Eckert and John Mauchly -- first general electronic computer.** (or was it John V. Atananasoff 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

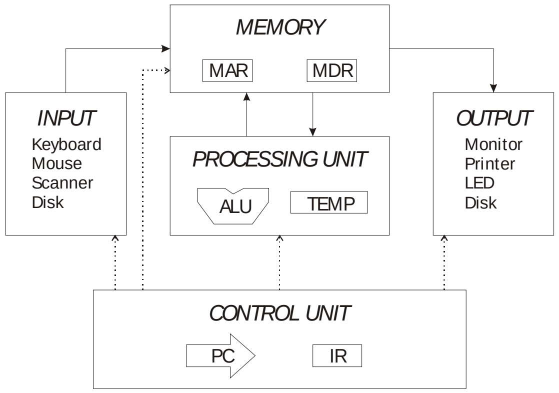
- **wrote a report on the stored program concept, known as the *First Draft of a Report on EDVAC***

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

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

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

Von Neumann Model



Memory

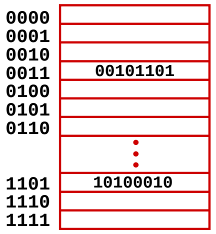
***k* x *m* array of stored bits (*k* is usually 2^n)**

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

MEMORY
MAR
MDR

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.

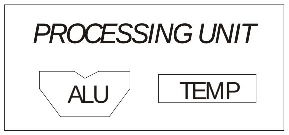
To write a value (X) to a location (A):

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-2 performs ADD, AND, NOT



Registers

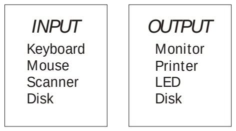
- Small, temporary storage
- Operands and results of functional units
- LC-2 has eight register (R0, …, R7)

Word Size

- number of bits normally processed by ALU in one instruction
- also width of registers
- LC-2 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-2 supports keyboard (input) and console (output)
- keyboard: data register (KBDR) and status register (KBSR)
- console: data register (CRTDR) and status register (CRTSR)

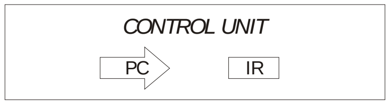
Some devices provide both input and output

- disk, network

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

Control Unit

Orchestrates execution of the program



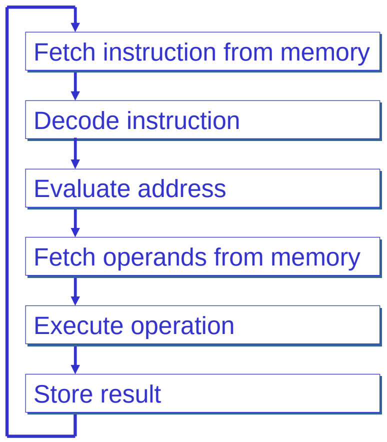
Instruction Register (IR) contains the ***current instruction***.

Program Counter (PC) contains the ***address*** of the next instruction to be executed.

Control unit:

- reads an instruction from memory
  - Ø the instruction's address is in the PC
- interprets the instruction, generating signals that tell the other components what to do
  - Ø an instruction may take many *machine cycles* to complete

Instruction Processing



4-٩

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)***.**

4-١٠

Example: LC-2 ADD Instruction

LC-2 has 16-bit instructions.

- **Each instruction has a four-bit opcode, bits [15:12].**

LC-2 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."

4-١١

Example: LC-2 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 stored in that address to R2."

4-١٢

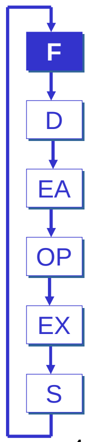
Instruction Processing: FETCH

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.



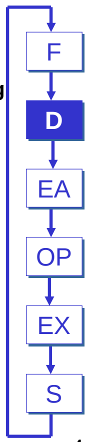
Instruction Processing: DECODE

First identify the opcode.

- In LC-2, 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:
  - Øfor LDR, last six bits is offset
  - Øfor ADD, last three bits is source operand #2

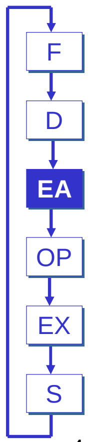


Instruction Processing: EVALUATE ADDRESS

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

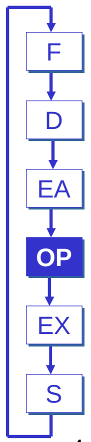


Instruction Processing: FETCH OPERANDS

Obtain source operands needed to perform operation.

Examples:

- load data from memory (LDR)
- read data from register file (ADD)

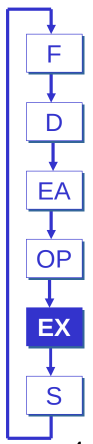


Instruction Processing: EXECUTE

Perform the operation,
using the source operands.

Examples:

- **send operands to ALU and assert ADD signal**
- **do nothing (e.g., for loads and stores)**

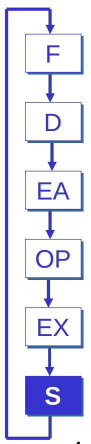


Instruction Processing: STORE

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**
  - **Øwrite address to MAR, data to MDR**
  - **Øassert WRITE signal to memory**



Microinstruction

Specifies the sequense of <u>microoperations</u> or <u>register transfer operations</u> needed to interpret and execute an instruction.

In the <u>Fetch</u> cycle for example the micro instructions executed are:

PC à MAR

Memory read

MDR à IR

PC+1 à PC

The micro instructions for the Fetch Operands cycle in the ADD instruction will be as follows:

- SR1 à ALU right side

- SR2 à ALU left side

For the LDR instruction, the Fetch operands cycle is as follows

- Evaluated address à MAR

- Read memory

- MDR à Data bus

Write down the micro instructions for the Store cycle for the two previous examples (ADD, LDR)

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, 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-2 JMPR Instruction

Set the PC to the value obtained by adding an offset to a register. This becomes the address of the next instruction to fetch.

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

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

"Add the value of 6 to the contents of R3, and load the result into the PC."

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:

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

- **not all phases are needed by every instruction**
- **phases may take variable number of machine cycles**

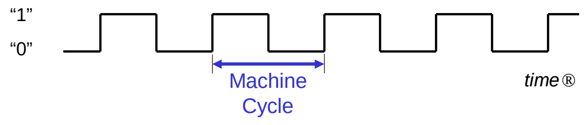
Driving Force: The Clock

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**



Instructions vs. Clock Cycles

MIPS vs. MHz

- **MIPS = millions of instructions per second**
- **MHz = millions of clock cycles per second**

These are not the same -- why?

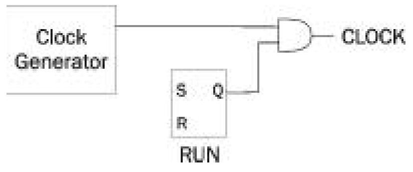
Stopping the Clock

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


Clock Generator
CLOCK
S
Q
R
RUN