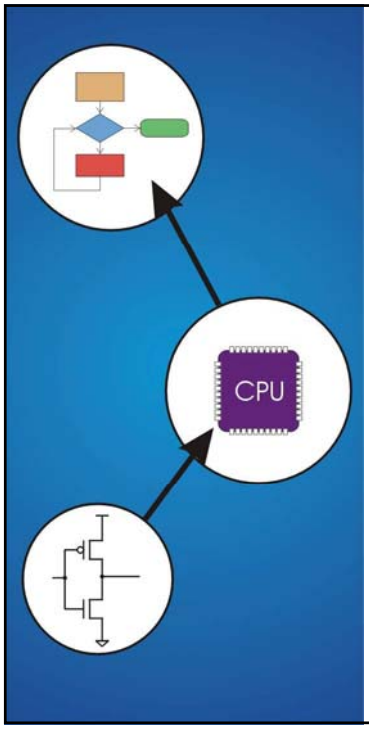


Chapter 4
The Von Neumann Model

The Stored Program Computer

1945: John von Neumann

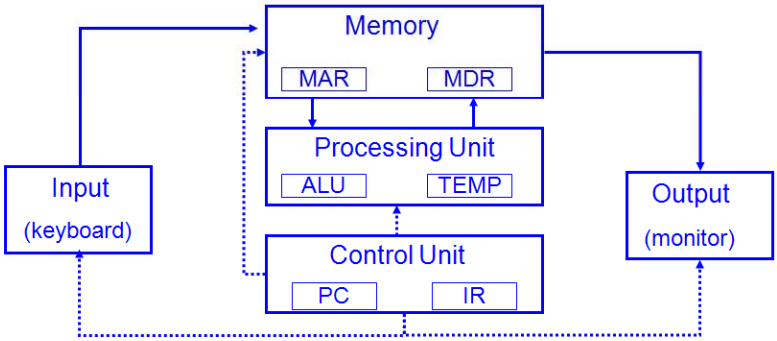
- **wrote a report on the stored program concept, known as the *First Draft of a Report on EDVAC (Electronic Discrete Variable Automatic Computer)***
- **The EDVAC was a binary serial computer with automatic addition, subtraction, multiplication with a memory of 5.5 kbytes.**
- **Addition time was 0.864 ms and its multiplication time was 2.9 ms**
- **6,000 vacuum tubes and 12,000 diodes, and consumed 56 kW of power. It covered 45.5 m² and weighted 7,850 kg.**
- **Operating personnel was thirty people for each eight-hour shift.**

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
http://en.wikipedia.org/wiki/EDVAC

The von Neumann Model - 1



- Memory: holds both data and instructions
- Processing Unit: carries out the instructions
- Control Unit: sequences and interprets instructions
- Input: external information into the memory
- Output: produces results for the user

4 - 3

Memory

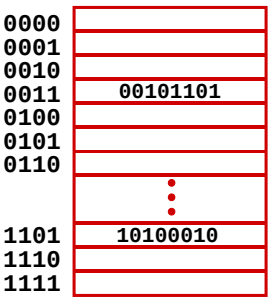
2^k x *m* array of stored bits

Address

- unique (*k*-bit) identifier of location

Contents

- *m*-bit value stored in location
- Each location has an *address* and *contents* stored at a given address.



Address Space:

- The total number of memory locations ("boxes") available.
  - eg. a 28 bit address provides an address space of 2^{28} locations.
- The LC-3 has an address space of 2^{16} locations - i.e. it uses a 16 bit address.

4-4

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):

- **read a value from a memory location**
  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):

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

4-5

Memory (relating to chap.3 logical circuits)

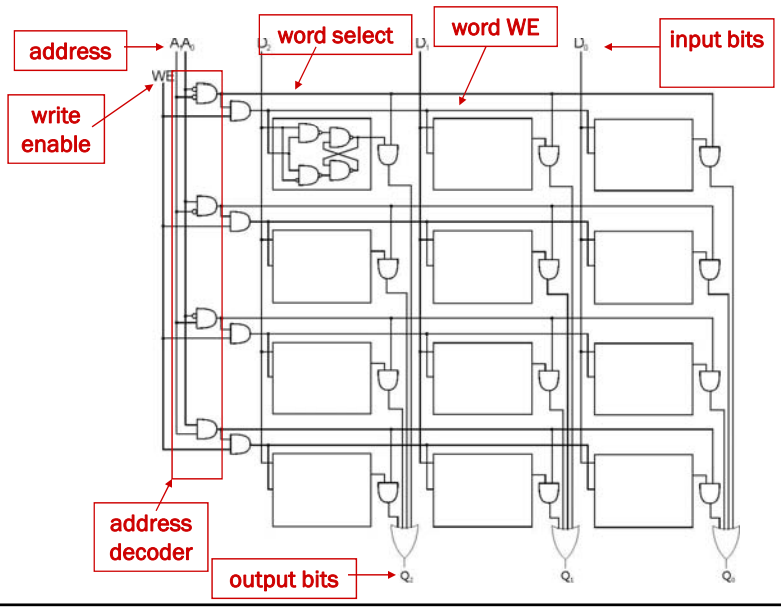
The Memory Address Register sets up the decoder circuitry in the memory.

Read: the contents of the specified address will be written to the Memory Data Register.

Write: the value to be stored is first written to the Memory Data Register, then the Write Enable is asserted, and the contents of the MDR are written to the specified address.

4 - 6

Remember Chap.3 and memory build-up

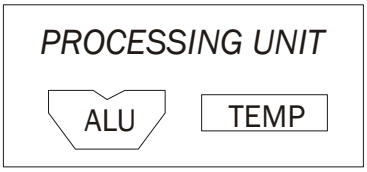


4-7

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

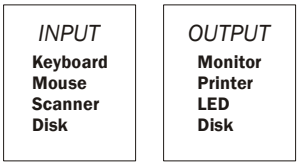
Word Size

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

4-8

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

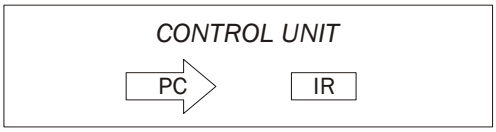
- **disk, network**

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

4-9

Control Unit

Orchestrates execution of the program



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

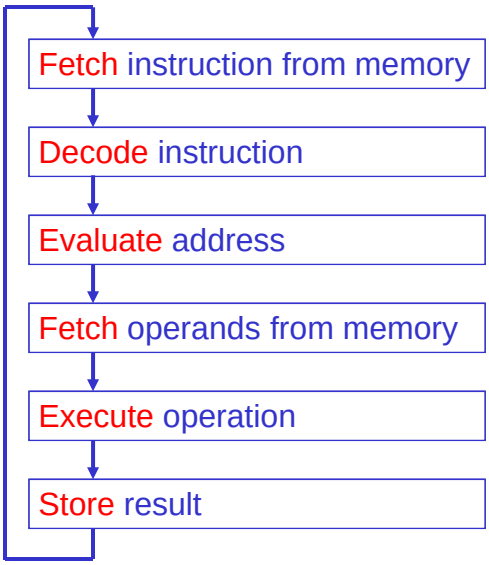
Program Counter (PC) contains the *address* of the next instruction to be executed.

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**

4-10

Instruction Processing



4-11

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)***.**

4-12

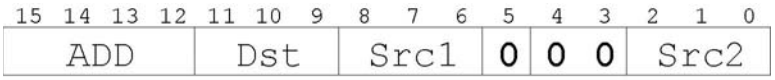
Example: LC-3 ADD Instruction

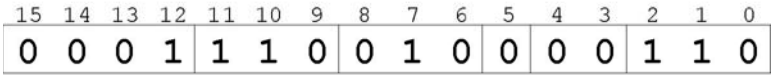
LC-3 has 16-bit instructions.

- **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 points to these registers.**
  - **Only need a unique 3-bits ID**





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

4-13

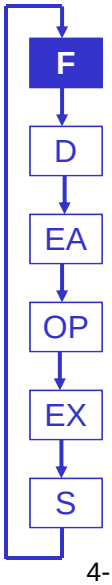
Instruction Processing: FETCH

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.**



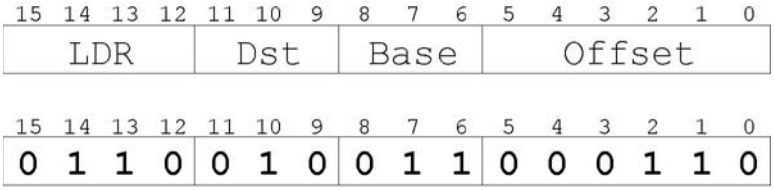
4-14

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 of that memory location to R2."

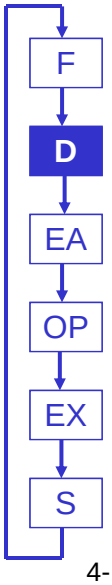
Instruction Processing: DECODE

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**



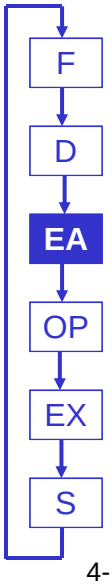
Instruction Processing: EVALUATE ADDRESS

For instructions that require memory access, compute address used for access.

- **e.g. the memory address from where we want to retrieve data (LDR)**
- **e.g. the memory address where we want to store data (STR)**

Examples:

- **add offset to base register (as in LDR)**
- **add offset to PC**
- **add offset to zero**



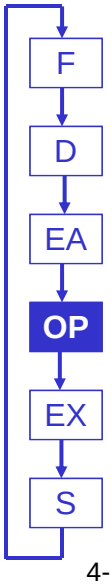
4-17

Instruction Processing: FETCH OPERANDS

Obtain source operands needed to perform operation.

Examples:

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



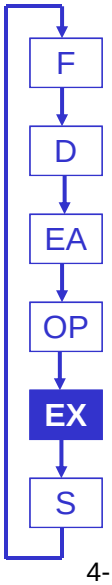
4-18

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)**



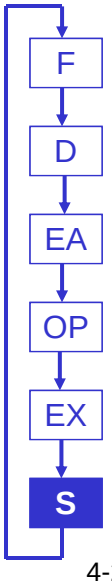
4-19

Instruction Processing: STORE RESULT

**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**



4-20