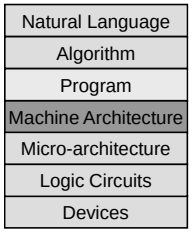
These notes are based on Dr Delmas' and many other people's notes

Chapter 4

The Von Neumann Model



Agenda & Reading

- The architecture of modern computer systems
- Instructions and instruction execution
- Read Chapter 4 of the textbook

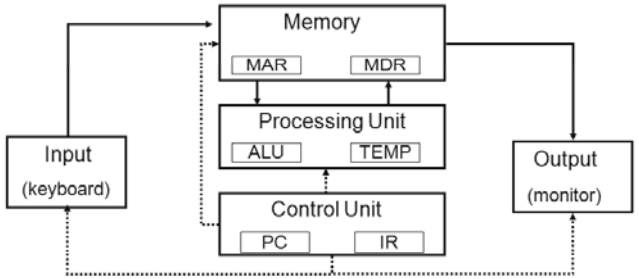
Agenda

- **The architecture of modern computer systems**
  - **The von Neumann architecture**
  - **Memory unit, processing unit, control unit, input/output**
- Instructions and instruction execution

von Neumann model (1)

- Modern computer architecture is first described by John von Neumann in 1940s
  - von Neumann model (architecture)
- A computer consists of
  - a *memory*, containing instructions and data
  - a *processing unit*, for performing arithmetic and logical operations
  - a *control unit*, for interpreting instructions
  - a *input/output unit*, for interacting with the users

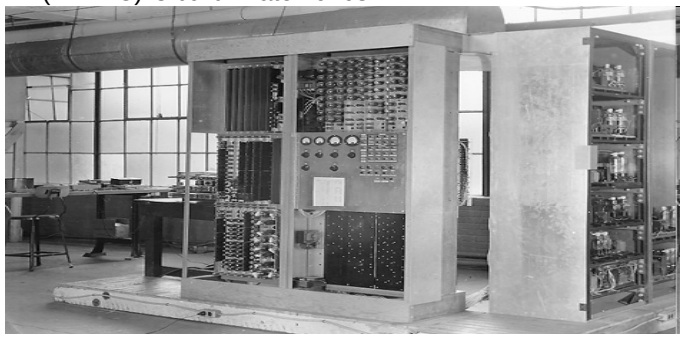
von Neumann model (2)



EDVAC image from

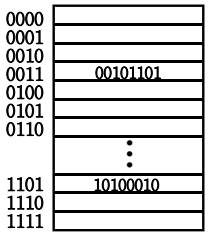
http://www.library.upenn.edu/exhibits/rbm/mauchly/img/9mach48.jpg

- The first machine based on von Neumann architecture (EDVAC) is built in late 1940s



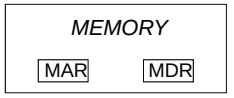
Memory

- The memory unit consists of many locations
- Each location consists of many bits
- Address
  - unique (*k*-bit) identifier of location
- Contents
  - *m*-bit value stored at a location
  - Each location has an *address*
- Address Space
  - The total number of memory locations ("boxes") available.
    - eg. a 28 bit address provides an address space of 2^{28} locations.



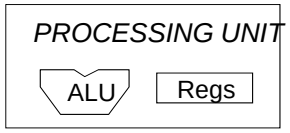
Interface to Memory

- How to 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
  - Write the address (A) into the MAR.
  - Send a "read" signal to the memory.
  - Read the data from MDR.
- To STORE a value (X) to a location (A):
  - write a value to a memory location
  - Write the data (X) to the MDR.
  - Write the address (A) into the MAR.
  - 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, …)



Registers
- Small, temporary storage
- store operands and results of functional units

Word Size
- The number of bits normally processed by ALU in one instruction
- also width of registers
- 32 bits, 64 bits etc

Input and Output

Devices for getting data into and out of computer

INPUT
Keyboard
Mouse
Scanner
Disk

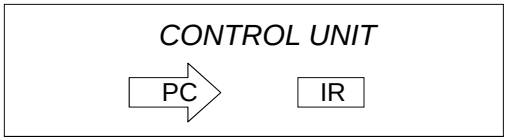
Each device has its own interface, usually a set of registers like the memory's MAR and MDR
- LC-3 supports keyboard (input) and monitor (output)
- keyboard: data register (KBDR) and status register (KBSR)
- monitor: data register (DDR) and status register (DSR)

Some devices provide both input and output
- disk, network

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

Agenda

- The architecture of modern computer systems
- **Instructions and instruction execution**
  - **What makes up an instruction**
  - **The steps involved in executing an instruction**

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.

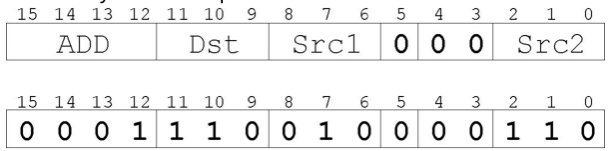
Example: LC-3 ADD Instruction

LC-3 has 16-bit instructions.

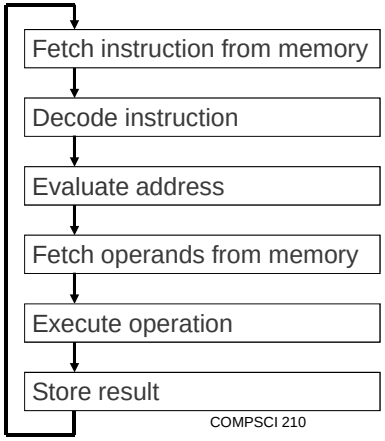
- Each instruction has a four-bit opcode, bits [15:12].

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



Instruction Processing



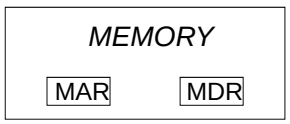
Instruction Processing: FETCH

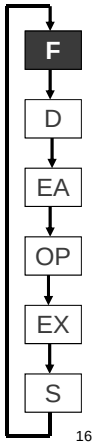
Load next instruction (at address stored in PC) from memory into Instruction Register (IR).

- 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.

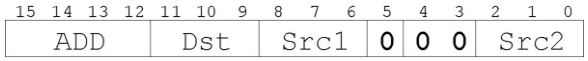


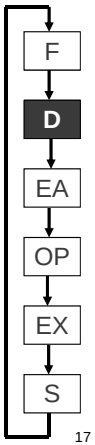


Instruction Processing: DECODE

First identify the opcode.
- In LC-3, this is always the first four bits of instruction.

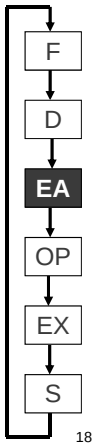
Depending on opcode, identify other operands from the remaining bits.
- Example:
  - for ADD, bits [11..9] is ID of the register that holds the result





Instruction Processing: EVALUATE ADDRESS

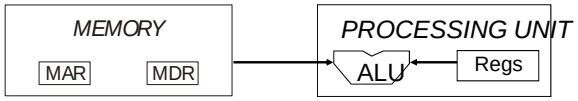
For instructions that require memory access, compute address used for access.
- –e.g. the memory address from where we want to retrieve data (LDR)
- –e.g. the memory address where we want to store data (STR)

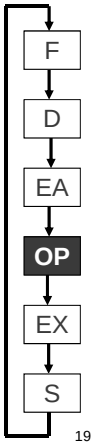


Instruction Processing: FETCH OPERANDS

Obtain source operands needed to perform operation.

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

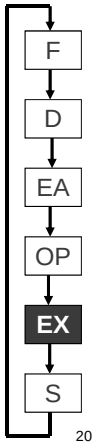




Instruction Processing: EXECUTE

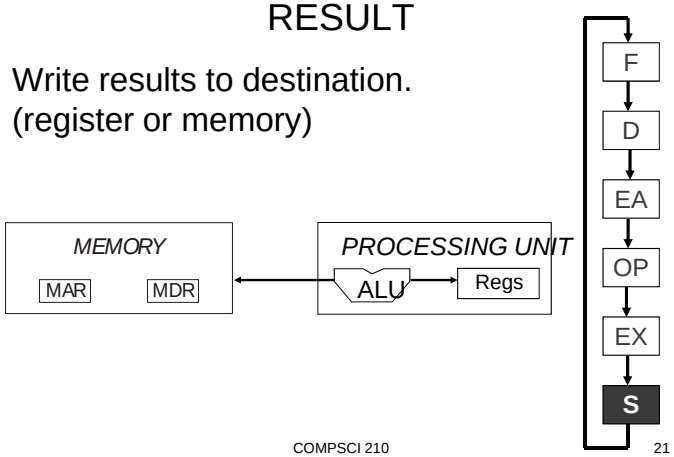
Perform the operation, using the source operands.

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



Instruction Processing: STORE RESULT

Write results to destination.
(register or memory)



Instruction Cycle – start over

- Start over …
  - The control unit just keeps repeating this whole process: so it now Fetches a new instruction from the address currently stored in the PC.
    - Recall that the PC was incremented in the first step (FETCH), so the instruction retrieved will be the next in the program as stored in memory - unless the instruction just executed changed the contents of the PC.

Changing the Sequence of Instructions

- In the FETCH phase, we increment the Program Counter by 1.
  - What if we don't want to always execute the instruction that follows this one?
  - examples: loop, if-then
- Control instructions can be used to change the contents of the PC.
  - jumps are unconditional -- they always change the PC
  - branches are conditional -- they change the PC only if some condition is true (e.g., the result of an ADD is zero)

Types of Instruction

- *Operate* Instructions
  - process data (addition, logical operations, etc.)
- *Data Movement* Instructions …
  - move data between memory locations and registers.
- *Control* Instructions …
  - change the sequence of execution of instructions in the stored program.
    - The default is sequential execution: the PC is incremented by 1 at the Fetch step, in preparation for the next one.
    - Control instructions set the PC to a new value during the Execute phase, so the next instruction comes from a different place in the program.
    - This allows us to build control structures such as loops and branches.

review

- The von Neumann architecture
- How each of the components in the von Neumann architecture works
- The functionality of different types of instructions
- The steps involved in executing an instruction