SFU

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Lecture 2
Technological Systems – Key Concepts
Computer Systems – Hardware

IAT 267 Introduction to Technological Systems

Instructor: Helmine Serban

Organizational items

- Assignment 1 (computer systems) – will be posted this week (Friday); due in two weeks

- We will have online quizzes
  - Questions from the workshop and lecture content
  - Will be posted to WebCt
  - We will let you know when the first quiz becomes available

Topics

- The big picture – from discrete electronic components to the computer system
- Computer organization: hardware
- Software

Summary of key concepts

- Technological Systems:
  - Iterative design
  - Trade-offs
  - Managing complexity
  - Real-world constraints
  - Feedback
- Computer Systems
  - Universal computing device
  - Transformations between layers

Computing Machines

- Ubiquitous (= everywhere)
  - General purpose: servers, desktops, laptops, PDAs, etc.
  - Special purpose: cash registers, ATMs, games, telephone switches, etc.
  - Embedded: cars, hotel doors, printers, VCRs, industrial machinery, medical equipment, etc.
- Distinguishing Characteristics
  - Speed
  - Cost
  - Ease of use, software support & interface
  - Scalability

Computer Systems

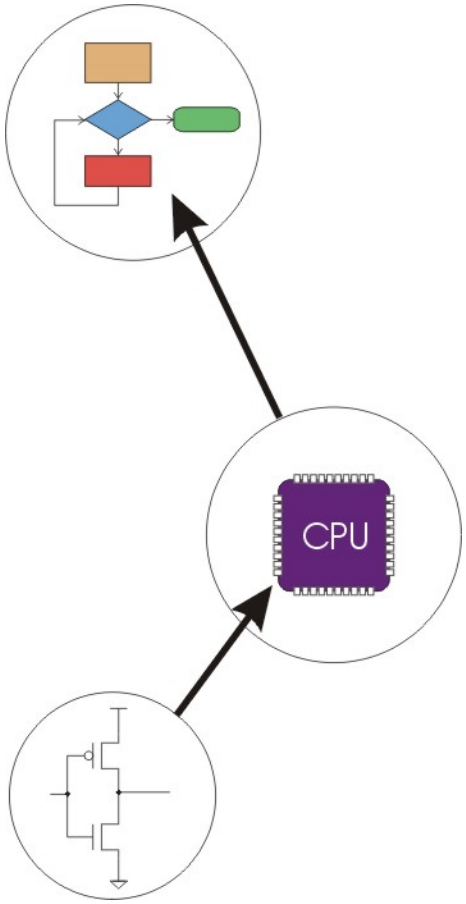
- **Purpose of a computer**
  - Turn data into information
  - Data: the raw facts and figures
  - Information: data that has been summarized and manipulated for use in decision making

- **Hardware vs. Software**
  - Hardware is the machinery and equipment in the computer
  - Software is the electronic instructions that tell the computer how to perform a task

Computer Systems

- **The basic operations**
  - **Input**: What goes in to the computer system
  - **Processing**: The manipulation a computer does to transform data into information
  - **Storage**:
    - Temporary storage: Memory is ***primary storage***
    - Permanent storage: Disks and media such as DVDs and CDs are ***secondary storage***
  - **Output**: What comes out
    - Numbers or pictures on the screen, printouts, sounds
  - **Communications**: Sending and receiving data

Big Idea #2:
Transformations Between Layers



Algorithms

Language

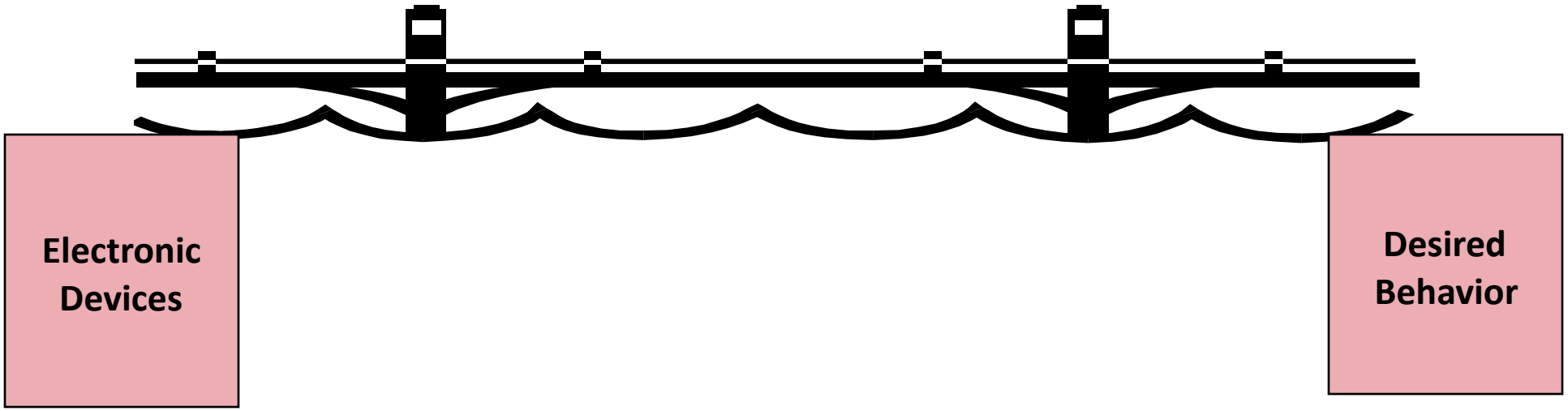
Instruction Set Architecture

Microarchitecture

Circuits

Devices

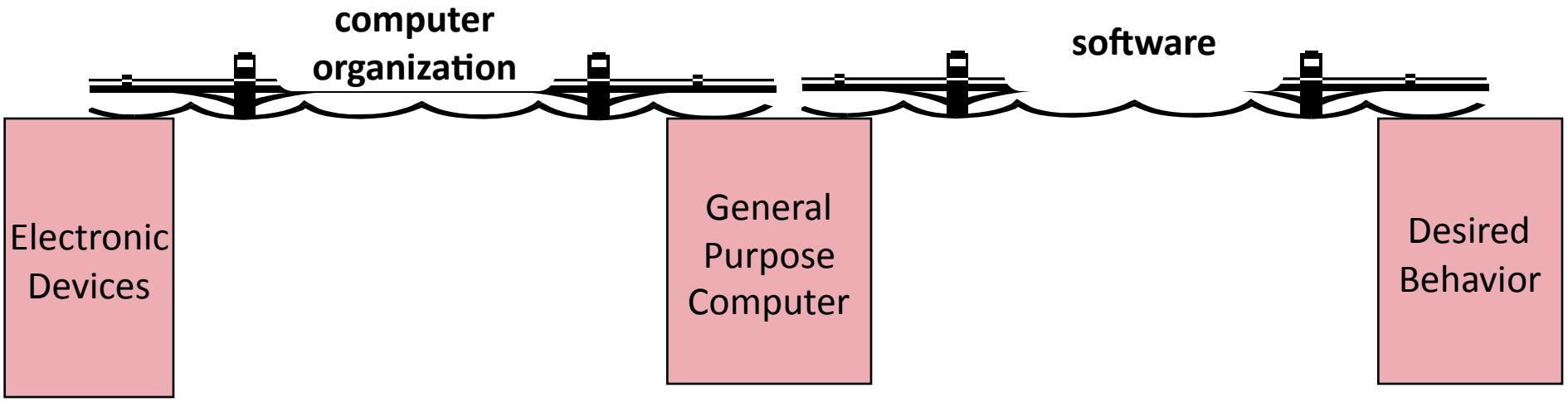
What is Computer Organization?



… a very wide gap between the intended behavior and the workings of the underlying electronic devices that will actually do all the work.

The forerunners to modern computers attempted to assemble the raw devices (mechanical, electrical, or electronic) into a separate purpose-built machine for each desired behavior.

Role of General Purpose Computers



A general purpose computer is like an island that helps span the gap between the desired behavior (application) and the basic building blocks (electronic devices).

How do we represent data in a computer?

- At the lowest level, a computer is an electronic machine.
  - works by controlling the flow of electrons

- Easy to recognize two conditions:
  1. presence of a voltage – we'll call this state "1"
  2. absence of a voltage – we'll call this state "0"

- Could base state on *value* of voltage, but control and detection circuits more complex.
  - compare turning on a light switch to measuring or regulating voltage

Computer is a **binary** digital system

Digital system:

- finite number of symbols

Binary (base two) system:

- has two states: 0 and 1

- Basic unit of information is the *binary digit*, or *bit*.
- Values with more than two states require multiple bits.
  - A collection of two bits has four possible states:
    00, 01, 10, 11
  - A collection of three bits has eight possible states:
    000, 001, 010, 011, 100, 101, 110, 111
  - *A collection of n bits has 2^n possible states.*

What kinds of data do we need to represent?

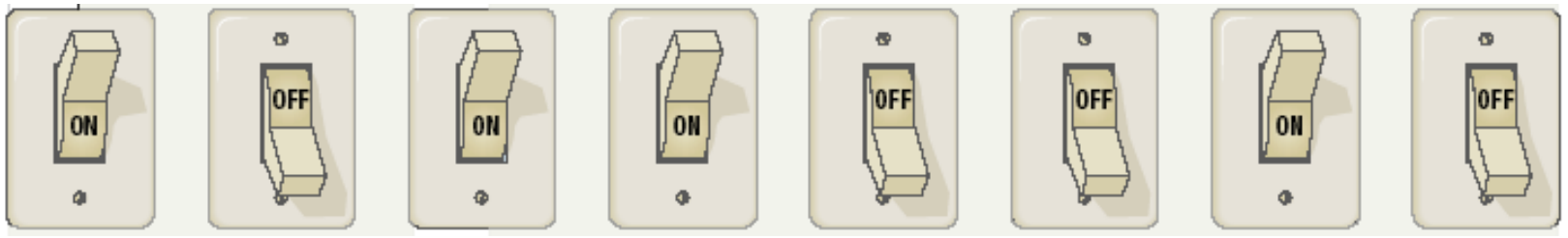
- Numbers – signed, unsigned, integers, floating point, complex, rational, irrational, …
- Text – characters, strings, …
- Images – pixels, colors, shapes, …
- Sound
- Logical – true, false
- Instructions
- …

- Data type:
  - *representation* and *operations* within the computer

Bits and bytes

Binary digits---bits

A **byte** comprises of **8 bits** and represents **1 character**



Binary and Haxadecimal Representation

- Binary digits: 0 and 1
  - Number of digits: 2 (base 2)
- Hexadecimal representation: 0, 1, 2, 3, 4, 5, 6 , 7, 8, 9 , A, B, C, D, E, F
  - Number of digits: 16 (base 16)

- Haxadecimal notation is used as a covenience notation – eg, 0011 1101 0110 1110 = 3D6E

Hardware of a computer system

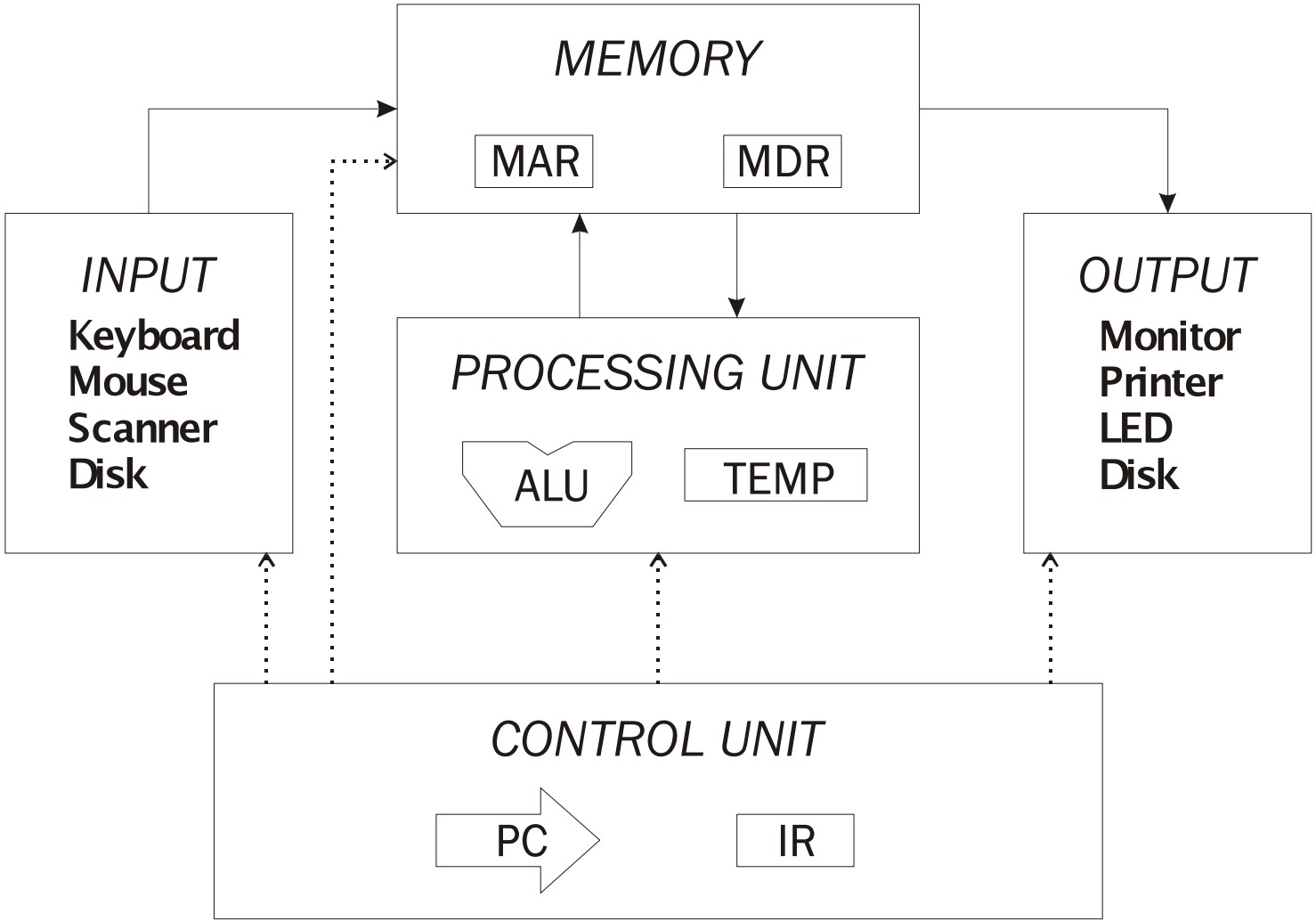
Computer Architecture

- Von Neumann Systems: Stored-Program Computer Systems
  - The original idea is from the inventor of ENIAC: Mauchly & Eckert
  - Firstly published by Von Neumann
  - Most modern general computers adopt Von Neumann Architecture

Three Elements of the Von Neumann Architecture

- Consists of three hardware elements
  - **Central Processing Unit (CPU)**
    - Control unit: Program Counter (PC), Instruction Register (IR)
    - Arithmetic Logic Unit (ALU)
    - Registers (small storage area)
  - **Main Memory**: holds programs and data
  - **I/O systems**: take external information into the memory/produces results for the user

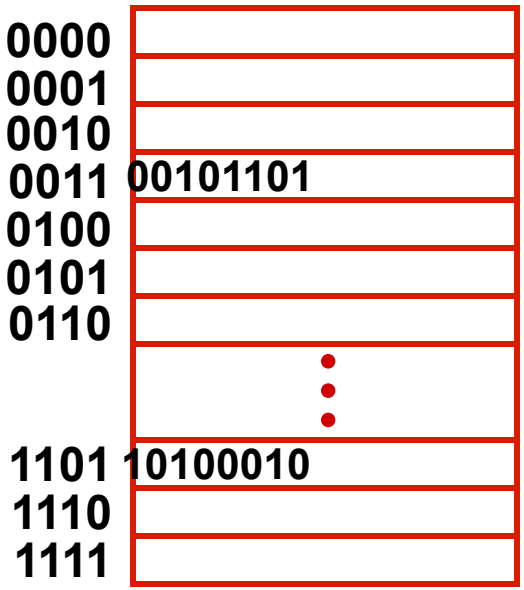
Von Neumann Model



- **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

Memory

- 2^k x 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



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): (Load = read from)
  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): (Store = write to)
  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, …)
- Registers
  - Small, temporary storage
  - Operands and results of functional units
- Word Size
  - number of bits normally processed by ALU in one instruction
  - also width of registers

Registers

- Special, high-speed storage areas
- Temporarily store date before processing
- All data must be represented in a register before it can be processed
- Numbers of registers / size of each – determine the power and speed of the CPU

Storage in registers

- Temporary storage

- For example:
  - Program instruction – while it is decoded
  - Data – while it is being processed by ALU
  - The result of a calculation

Word Size

- Word size is the number of bits that the processor may process at one time
  - The more bits in a word, the faster the computer
  - E.g.: a 32-bit computer (with a 32-word processor) will transfer data within each microprocessor chip in 32-bit chunks, or 4 bytes at a time (1 byte = 8 bits).

ALU – Arithmetic Logic Unit

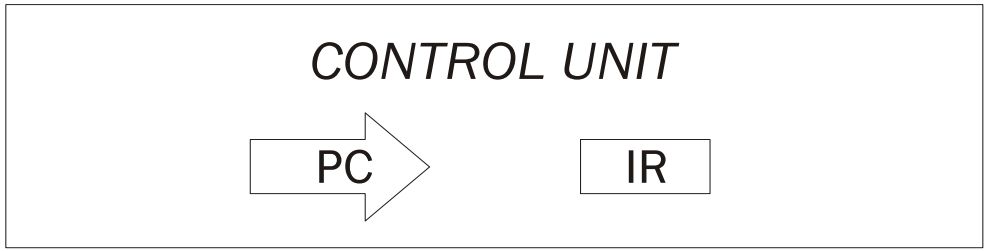
- The **arithmetic logic unit** (**ALU**) is a digital circuit that performs arithmetic and logical operations.
- The ALU is a fundamental building block of the central processing unit (CPU) of a computer.
- Even the simplest microprocessors contain an ALU for purposes such as maintaining timers.

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
  - 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

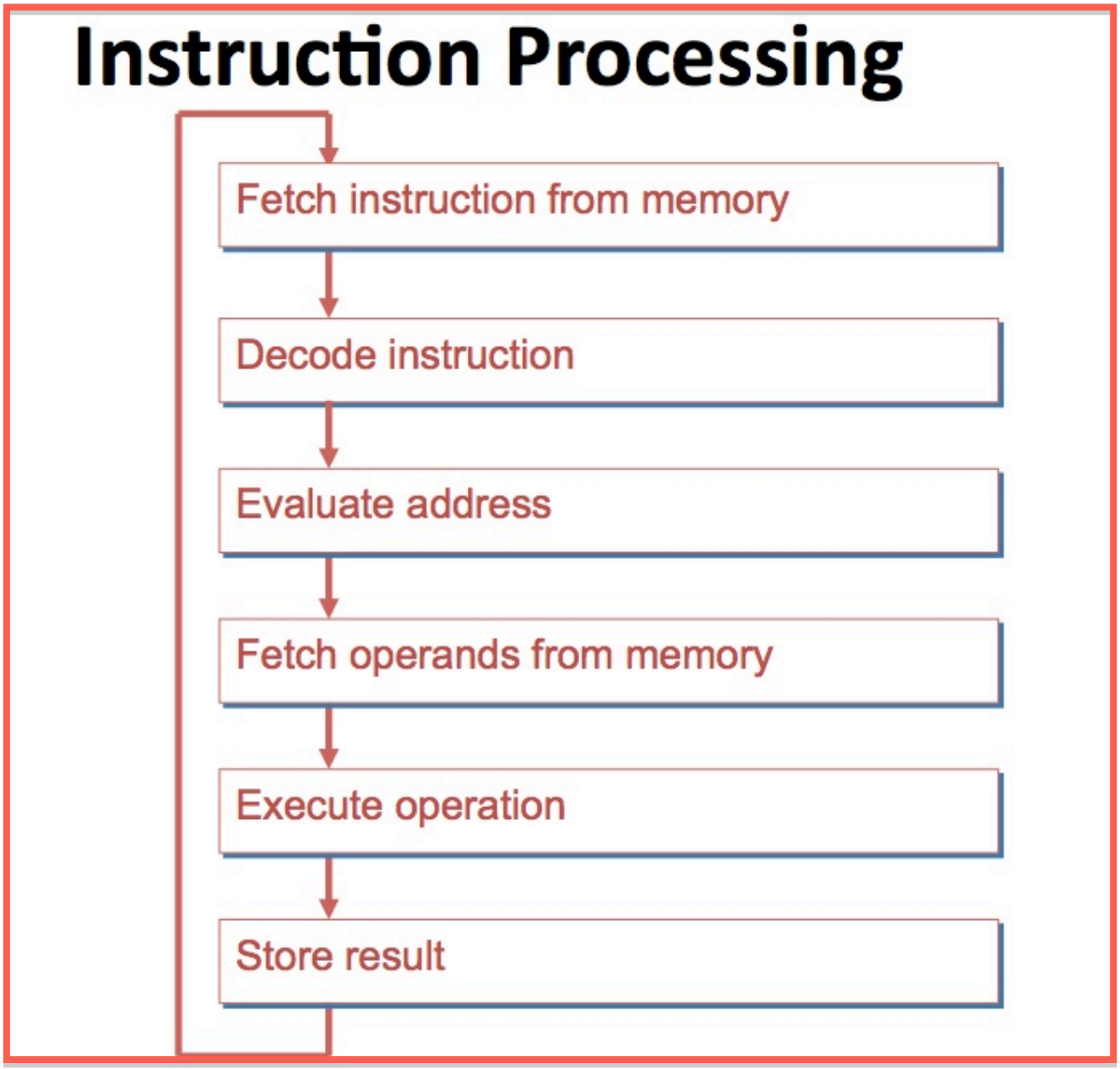
Machine Cycle

- The shortest interval in which an elementary operation can take place within the processor. It is made up of some number of clock cycles.

- The computer can only do one thing at a time. Each action must be broken down into the most basic steps. One round of steps from getting an instruction back to getting the next instruction is called the **Machine Cycle**.

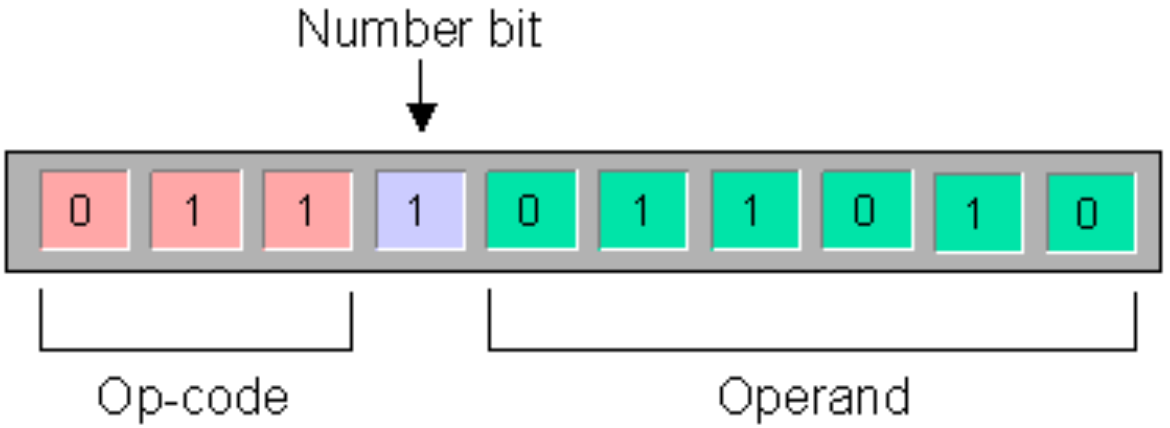
Instruction Processing

- Central ideas in the von Neumann model:
  - Program and data are both stored as sequences of bits in the computer's memory
  - The program is executed one instruction at a time under the direction of the control unit


Instruction Processing
Fetch instruction from memory
Decode instruction
Evaluate address
Fetch operands from memory
Execute operation
Store result

Instruction Structure

- Each machine instruction is composed of two parts: the op-code and the operand



A Simple Machine Language

Op-code	*Mnemonic*	*Function*	*Example*
001	LOAD	Load the value of the operand into the Accumulator	LOAD 10
010	STORE	Store the value of the Accumulator at the address specified by the operand	STORE 8
011	ADD	Add the value of the operand to the Accumulator	ADD #5
100	SUB	Subtract the value of the operand from the Accumulator	SUB #1
101	EQUAL	If the value of the operand equals the value of the Accumulator, skip the next instruction	EQUAL #20
110	JUMP	Jump to a specified instruction by setting the Program Counter to the value of the operand	JUMP 6
111	HALT	Stop execution	HALT

A simple machine language

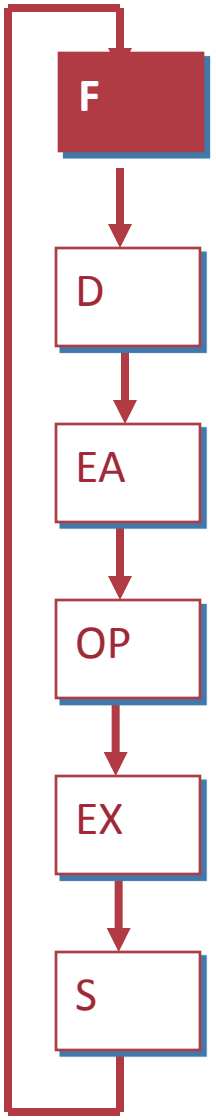
Instruction

- 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)*.

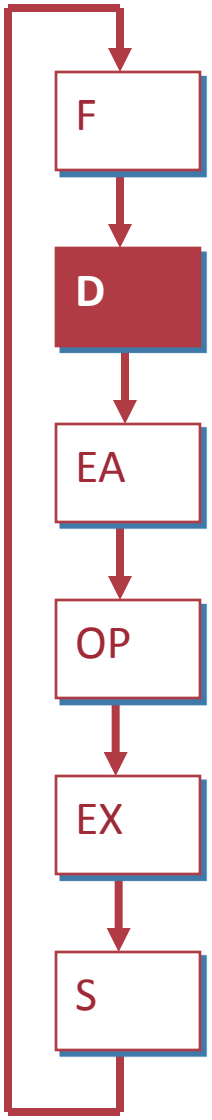
Instruction Processing: FETCH

- 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

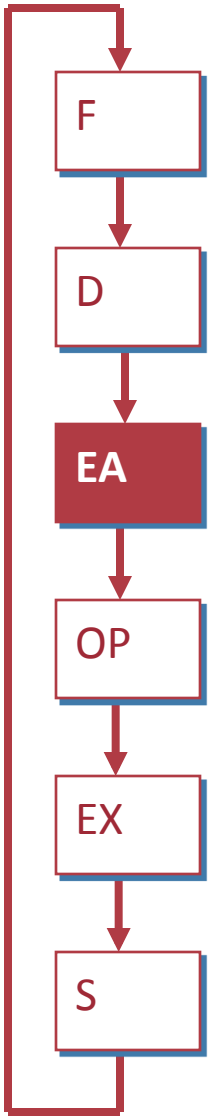
- First identify the opcode.
  - A 4-to-16 decoder asserts a control line corresponding to the desired opcode.
- Depending on opcode, identify other operands from the remaining bits.



Instruction Processing: EVALUATE ADDRESS

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

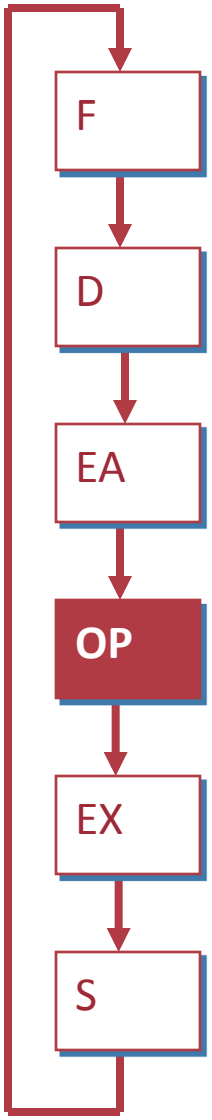
- Examples:
  - add offset to base register (as in LDR)
  - add offset to PC
  - add offset to zero



Instruction Processing: FETCH OPERANDS

- Obtain source operands needed to perform operation.

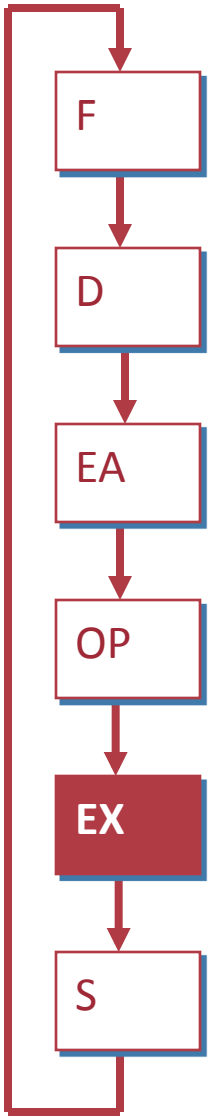
- Examples:
  - load data from memory (LDR)
  - read data from register file (ADD)



Instruction Processing: EXECUTE

- Perform the operation, using the source operands.

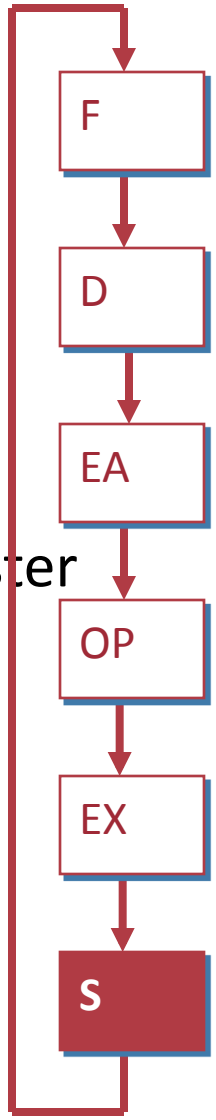
- Examples:
  - send operands to ALU and assert ADD signal
  - do nothing (e.g., for loads and stores)



Instruction Processing: STORE RESULT

- Write results to destination.
  (register or memory)

- 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



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, function call

- Need special instructions that change the contents of the PC.

- These are called *control instructions*.
  - 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)

Instruction Processing Summary

- Instructions look just like data -- it's all interpretation.

- Three basic kinds of instructions:
  - computational instructions
  - data movement instructions
  - control instructions

- Six basic phases of instruction processing:
- F → D → EA → OP → EX → S
  - not all phases are needed by every instruction
  - phases may take variable number of machine cycles

Software

- System Software: The Power behind the Power
- The Operating System: it's role
- Other System Software: Device Drivers & Utilities

System Software: The Power behind the Power

- **Application Software**
  - Software developed to solve a particular problem for users
    - Either performs useful work on a specific task
    - Or provides entertainment
  - We interact mainly with this software

- **System Software**
  - Enables application software to interact with the computer
  - Helps the computer to manage its own internal and external resources

System Software: The Power behind the Power

- System Software has 3 basic components:

1. Operating System (OS)

- The principal component of system software
- Low-level, master system of programs to manage basic computer operations

2. Device Drivers

- Help the computer control peripheral devices

3. Utility Programs

- Used to support, enhance, or expand existing programs in the compute

Thank you

Questions?