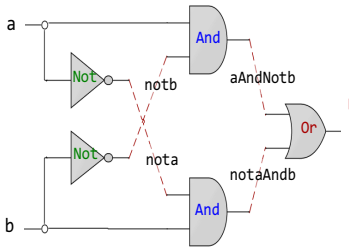
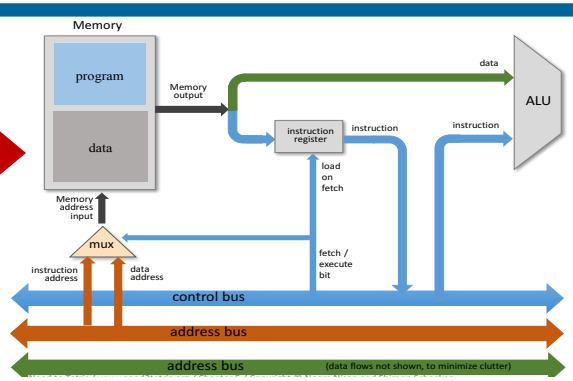
Computer Organization & Assembly Language Programming




```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```




```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 14

Instruction Set Architecture - I

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

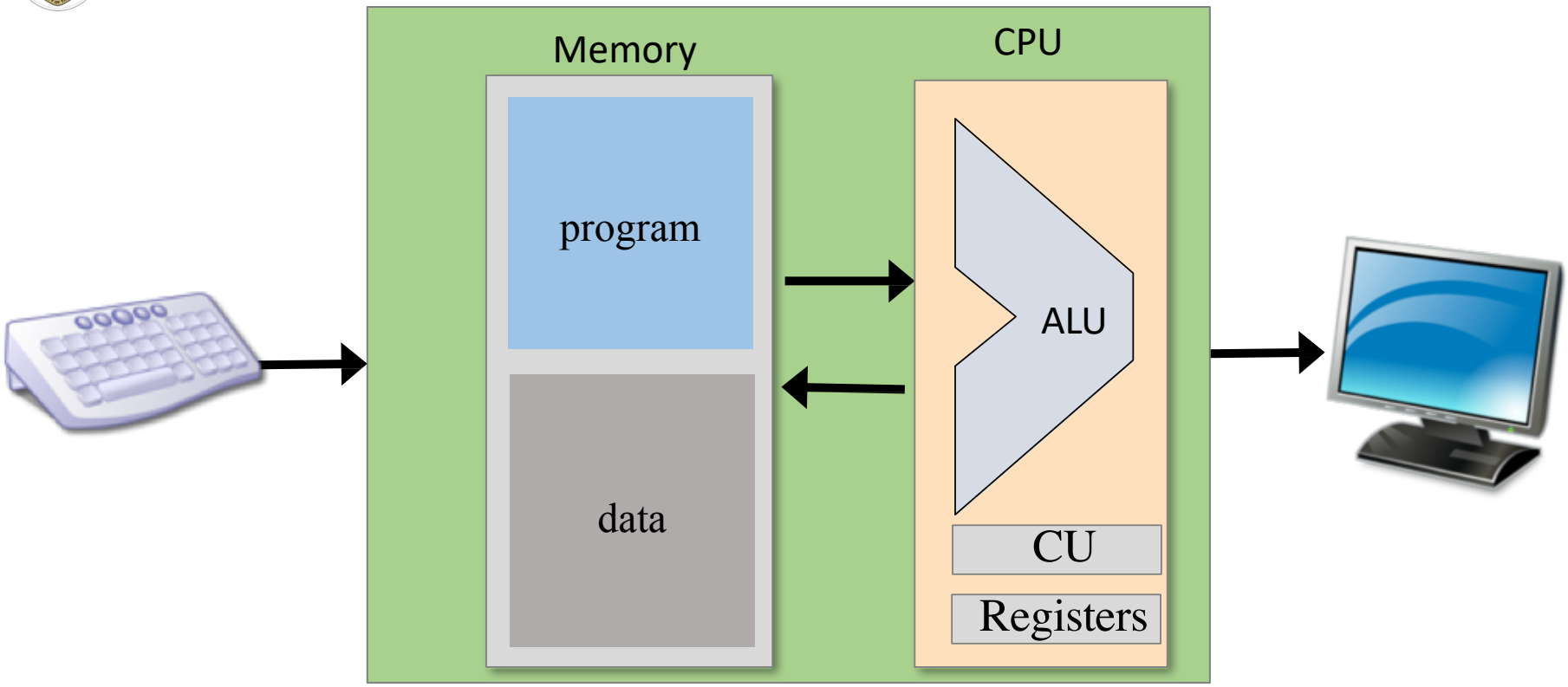
Slides of first half of the course are adapted from:
https://www.nand2tetris.org
Download s/w tools required for first half of the course from the following link:
https://drive.google.com/file/d/0B9c0BdDJz6XpZUh3X2dPR1o0MUE/view

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- Overview of Computer System
- Universality of Computer System
- Turing Machine
- Von Neumann Architecture
- Instruction Set Architecture (ISA)

Overview of Computer System



- A **machine language** is an agreed-upon formalism, designed to code low-level programs as a series of machine instructions
- Using these instructions residing inside the ***Memory***, the programmer can command the ***CPU*** to fetch an instruction/data from memory/input device, perform arithmetic/logic operations on that data, and finally store result inside the memory/output device. Moreover, data may need to be moved from one ***Register*** to another and may need to test different Boolean conditions

Universality of Computer Systems

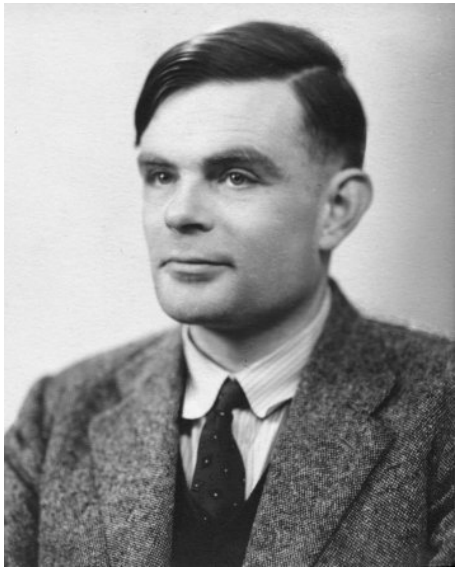
Computers Are Flexible

- Most machines in the world do one thing, e.g., washing machine washes clothes, air conditioners are used to control temperature of a room
- On the other hand a computer e.g., a smart phone can do lots and lots of things, voice communication, word processing, playing games, using internet, watch videos, and so on

Universality

Same **hardware** can run many different **software** programs

Theory



Alan Turing: (1912 – 1954)

Universal Turing Machine (1936)

Practice

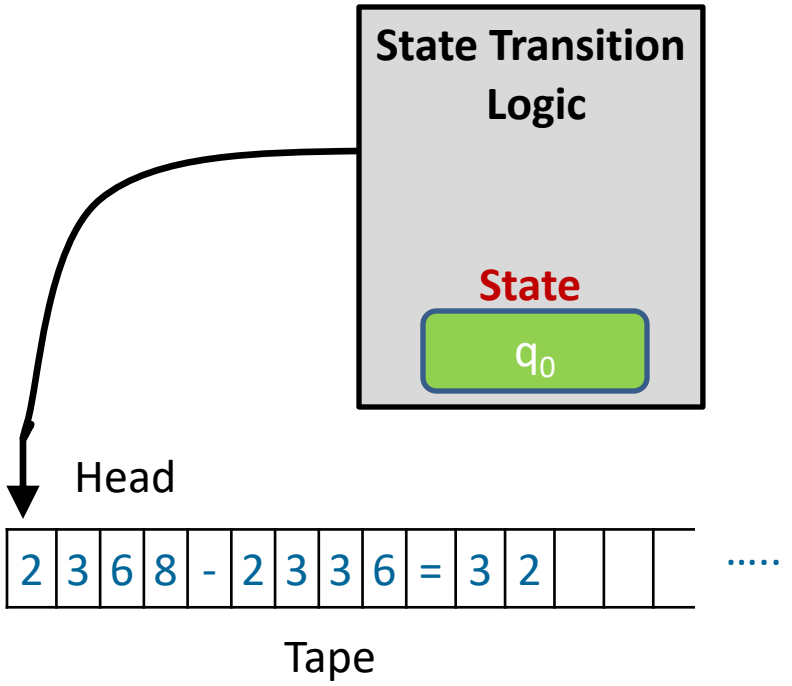
John Von Nuemann: (1903 – 1957)

Stored Program Computer

Turing Machine

- The **Turing Machine** (TM) was invented in 1936 by **Alan Turing**, which is a simple device that can capture the notion of computability (Computability means what can or cannot be solved/computed on a computer?)
- A Turing Machine (TM) is a mathematical model which consists of state transition logic, a state register, a head and an infinite length tape divided into cells on which input is given. It reads the input symbol pointed to by the Head, according to the transition function replaces it with another symbol, change the register state, and moves the Head to either left or right. On reaching the end of the string, if the register state is accept state, the input string is accepted, otherwise rejected
- A Turing Machine (TM) consists of:
  - i. Finite set of states (Q)
  - ii. Input alphabets (allowed symbols)
  - iii. Tape alphabets (input symbols)
  - iv. Transition Logic/Function
  - v. Start state q_0
  - vi. Accept state q_{accept}
  - vii. Reject state q_{reject}

$$\begin{array}{r} 2368 \\ -\ 2336 \\ \hline 32 \end{array}$$



Turing Machine (cont…)

Example: A TM that checks an odd binary number

i. $Q = \{q_0, q_1, q_a, q_r\}$

ii. Σ = Input alphabets $\{0, 1\}$

iii. X=Tape alphabets{0, 1, space}

iv. Transition Table/Function

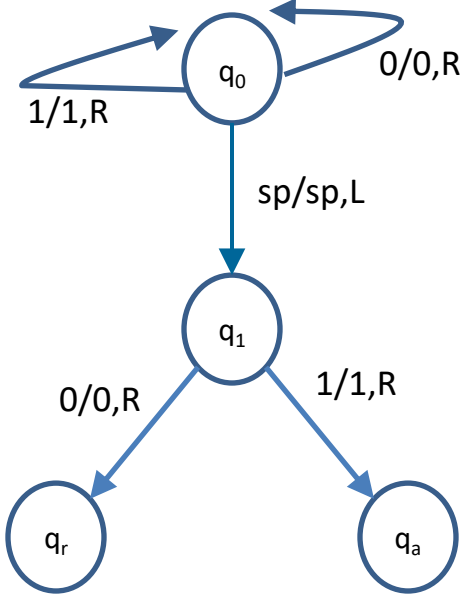
$$\delta : Q,X \rightarrow Q,X ,L/R$$

v. Start state = q_0

vi. Accept state = q_a

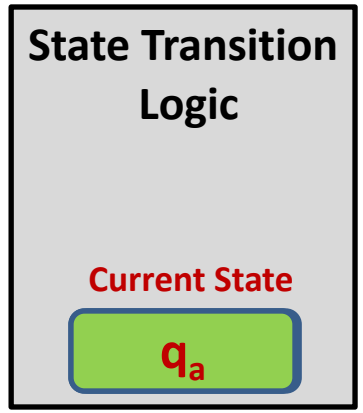
vii. Reject state = q_r

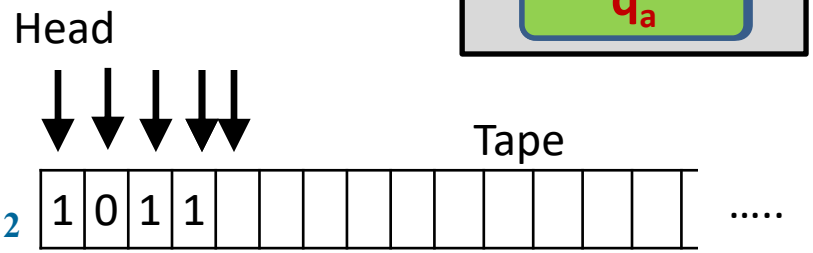
State Diagram



Transition Table

q_0 , 0	q_0 , 0, R
q_0 , 1	q_0 , 1, R
q_0 ,sp	q_1 , sp, L
q_1 , 0	q_r , 0, R
q_1 , 1	q_a , 1, R





Turing Machine (cont…)

Example: A TM that checks an odd binary number

i. $Q = \{q_0, q_1, q_a, q_r\}$

ii. Σ = Input alphabets $\{0, 1\}$

iii. X=Tape alphabets{0, 1, space}

iv. Transition Table/Function

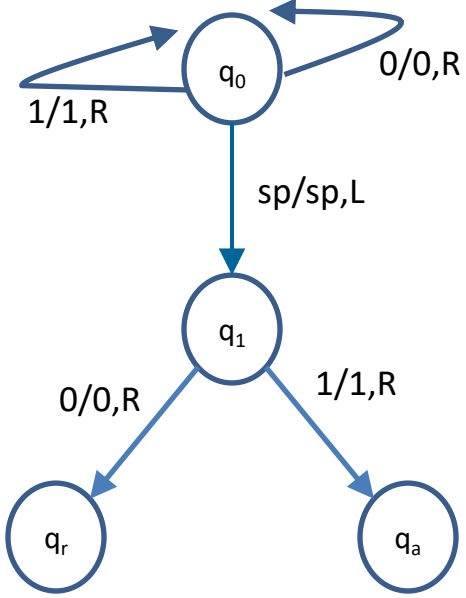
$$\delta : Q,X \rightarrow Q,X ,L/R$$

v. Start state = q_0

vi. Accept state = q_a

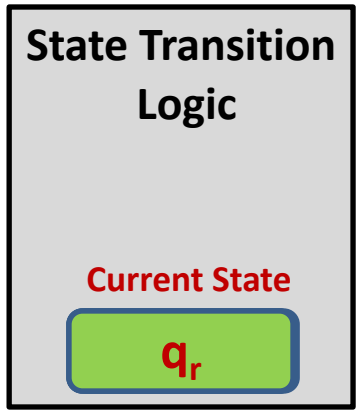
vii. Reject state = q_r

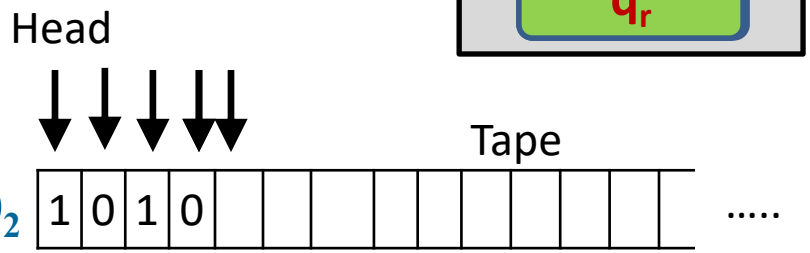
State Diagram



Transition Table

q_0 , 0	q_0 , 0, R
q_0 , 1	q_0 , 1, R
q_0 ,sp	q_1 , sp, L
q_1 , 0	q_r , 0, R
q_1 , 1	q_a , 1, R

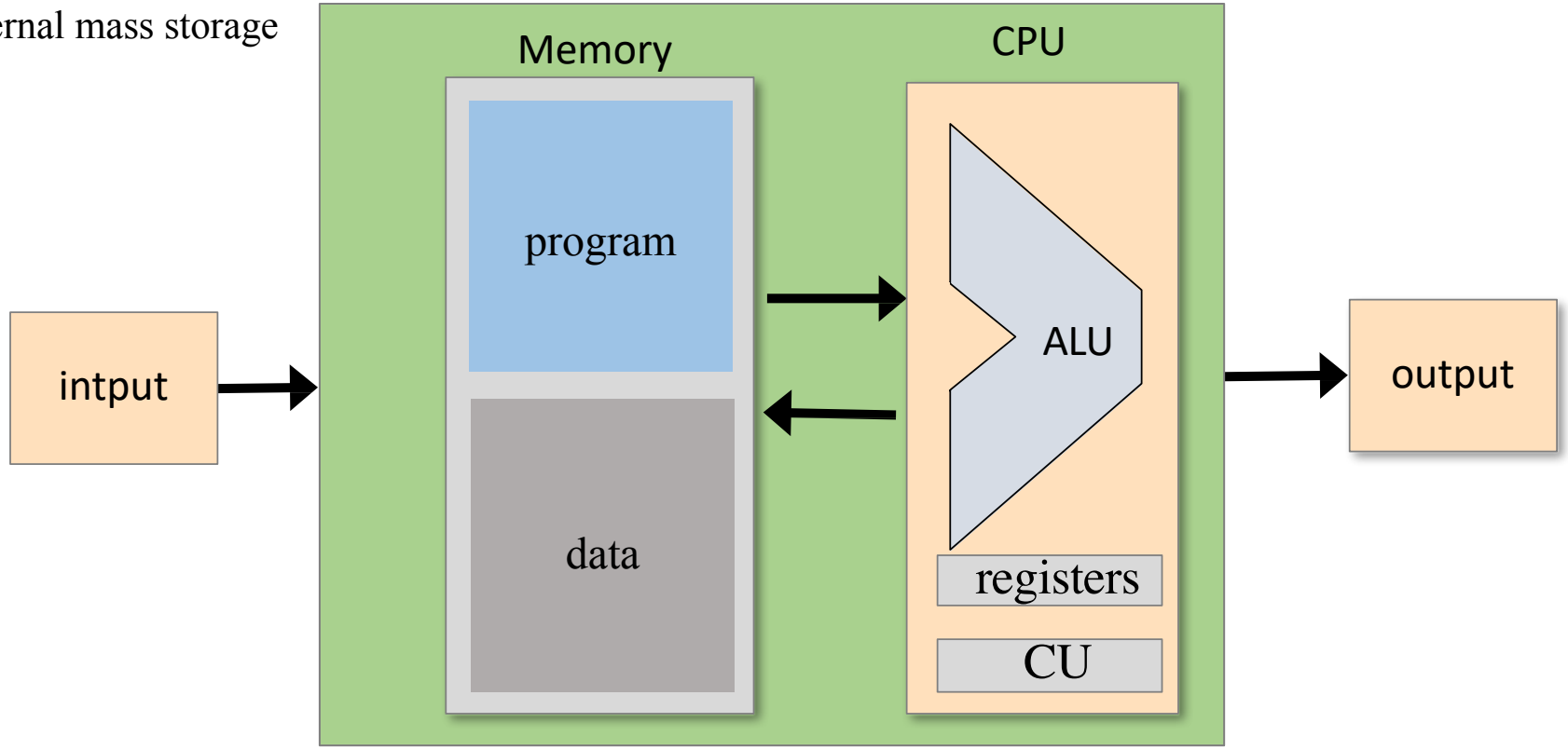




Von Neumann Architecture

The Von Neumann architecture is a computer architecture given by a mathematician and physicist John von Neumann describes the design architecture for an electronic digital computer with these components:

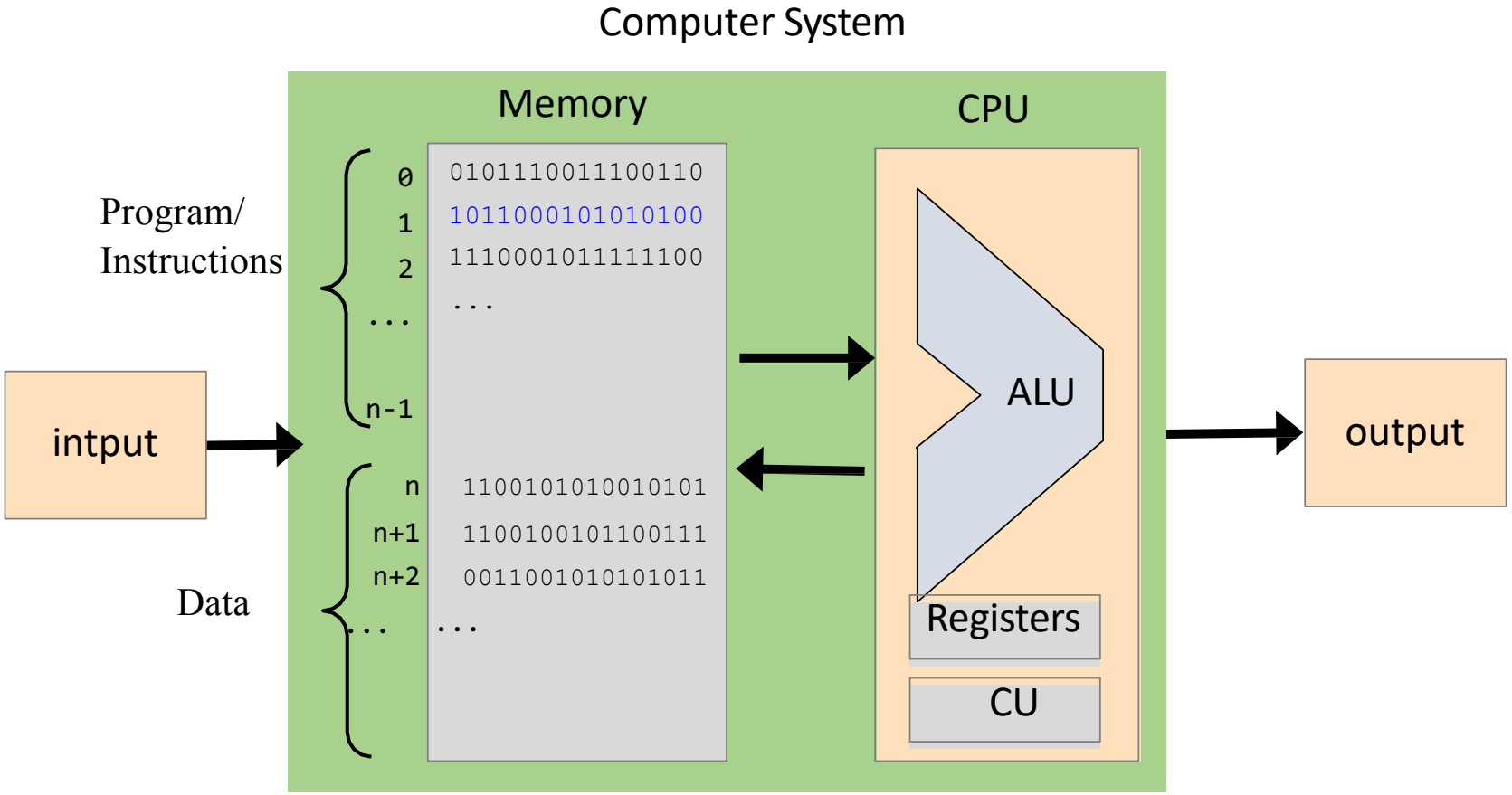
- A Processing Unit that contains an ALU and registers
- A Control Unit that contains an instruction register and program counter
- A Memory unit that stores data and instructions
- An Input and Output mechanism
- An external mass storage



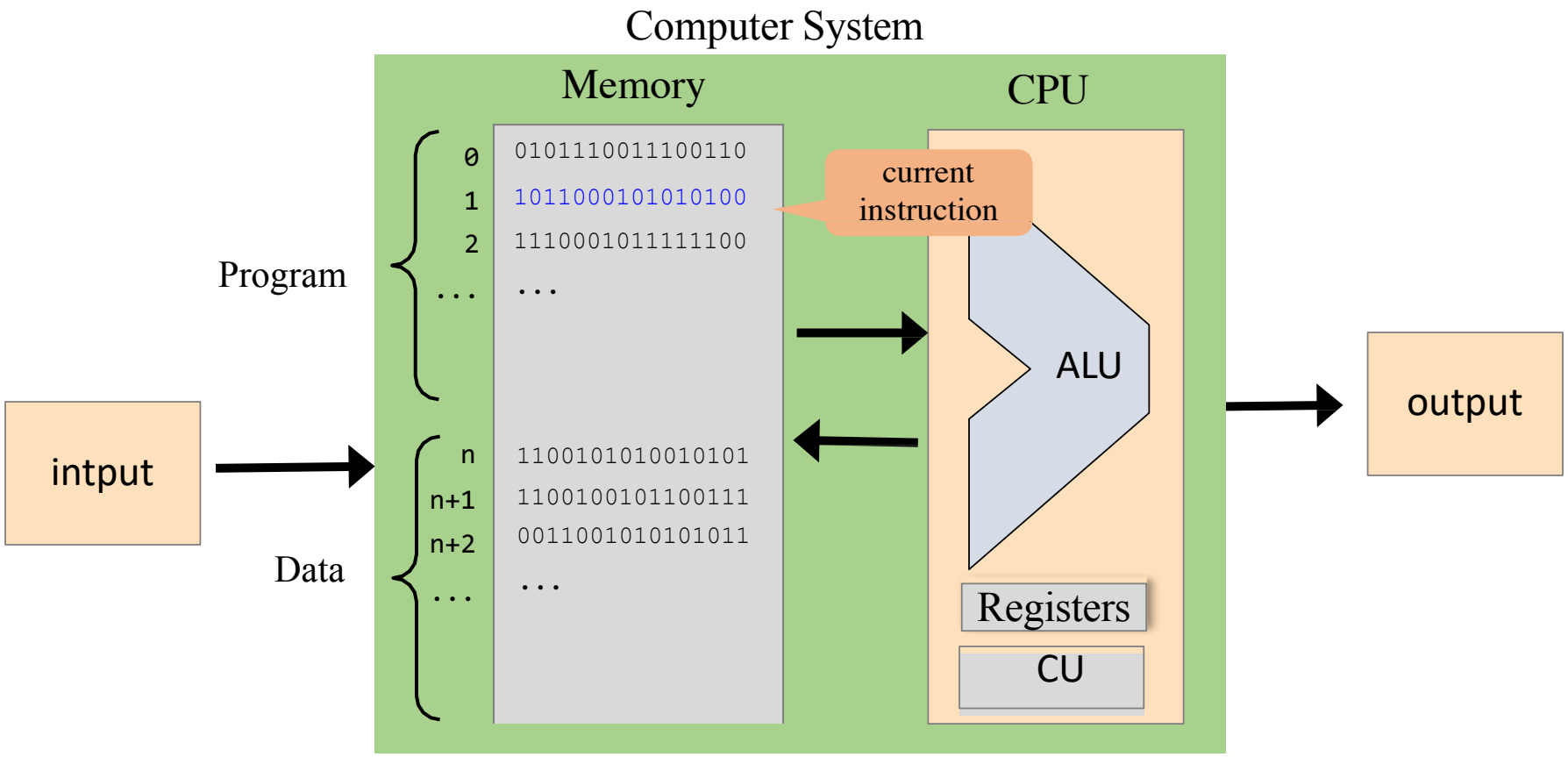
Stored Program Concept

The main idea in the Von Neumann architecture is the stored program concept. We can put the program inside the memory along with the data on which this program is going to operate. This is how

Same hardware can run many different software programs



Machine Language



Handling instructions:

- 1011 means "addition" — operation
- 000101010100 means "operate on memory address 340" — addressing
- Next we have to execute the instruction at address 2 — control

Assembler and Mnemonics

Instruction: `0100010 0011 0010`

add R3 R2

Sample instruction

Option 1:

- Use machine language, in which case a programmer should exactly know the bit positions and their meanings
- Really difficult to write programs in machine language. No one do so these days

Option 2:

- Use symbolic machine language instructions, using assembly language of the specific hardware, e.g., **`add R3 R2`**
- A bit easy to write programs in assembly language, but later someone has to transform it to machine language
- A program called the assembler is used to translate the symbolic code into machine code

Assembler and Symbols

- An instruction that instructs to add 1 to the contents of memory location 129, may be encoded in the machine and assembly language like:

Machine Language:

```
1010 0001 10000001
```

add 1 Mem[129]

Assembly Language:

```
add 1, Mem[129]
```

- For a programmer, it is a bit difficult to specify and memorize the memory addresses for different purposes
- A more friendlier syntax can be used if we assume that the symbol **`index`** stands for **`Mem[129]`**
- The assembler will resolve the symbol **`index`** into the specific memory address, i.e., "**`index`**" ⟹ **`Mem[129]`**

Assembly Language:

```
add 1, index
```

Instruction Set Architecture (ISA)

- Every computer has an ***Instruction Set Architecture (ISA)***, which is the set of instructions, registers, memory space and other features visible to the assembly language programmer
- It is an Interface between hardware and low-level software and sometimes referred to as a machine language, although it is not entirely accurate
- Example ISAs: x86, ARM, MIPS, PowerPC, SPARC, RISC-V
- **Six dimensions of ISA:**
  - Class of ISA
  - Types and Sizes of Operands
  - Operations
  - Memory Addressing Models and Addressing Modes
  - Control Flow Instructions
  - Encoding an ISA

Things To Do

Coming to office hours does NOT mean you are academically week!