Introduction to Computer and Programming

Lecture 12

Yue Zhang
Westlake University

August 1, 2023

WestlakeNLP

Chapter 12.

Building a Computer

WestlakeNLP

The Design

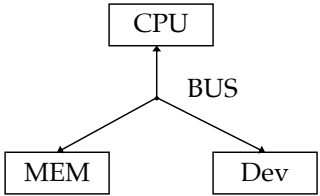
- We have now both the **theory** (computability issues) and the **materials** (transisters/chips) ready!
- Let us build a computer!
- A computer is a programmable finite state machine.
- We need to specify the program (i.e., instruction set) and the architecture (i.e., hardware FSM)

WestlakeNLP

The Design

The Von Neumann Architecture

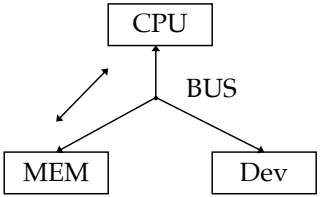
John von Neumann
(December 28, 1903 – February 8, 1957)



- CPU – Central processing unit; the computer FSM.
- MEM – The memory; where **data** and **code** are stored.
- Dev – Devices; which digitize input signals and translates output signals.
- BUS – Bus for communication between components.

The Design

The Von Neumann Architecture

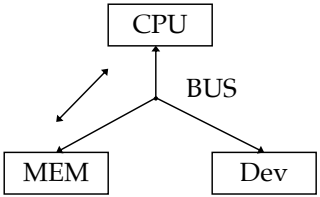


- Load data and code to the memory.
- Machine starts.
- The CPU reads code from the beginning of the memory and executes code until a HALT command is executed.

WestlakeNLP

The Design

The Von Neumann Architecture



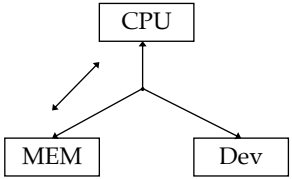
- Load data and code to the memory.
- Machine starts.
- The CPU reads code from the beginning of the memory and executes code until a HALT command is executed.

Design a specific machine according to this architecture.

WestlakeNLP

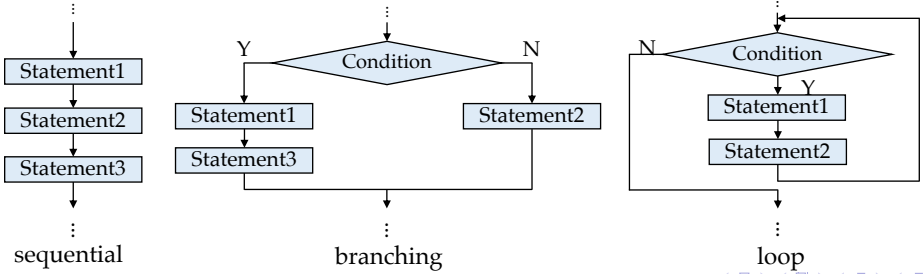
The Design

The Von Neumann Architecture



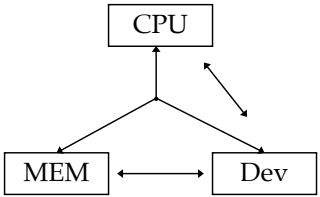
- Code execution should allow branching and looping.

Modes of code execution



The Design

The Von Neumann Architecture



- The (input) device can **interrupt** the CPU code execution process, and put data to the memory. (e.g., key press)
- The CPU can call the (output) device to send data to it. (e.g., play sound)

WestlakeNLP

The Design

Correspondingly, the following basic CPU functions are needed.

1. Loading data and tracking execution.
2. Arithmetic and logical functions.
3. Reading and writing memory.
4. Talking to devices.

WestlakeNLP

The Design

How do we implement the above using circuit?

- Specify a computer word.

  32 bits = 4 bytes

  communication, command, index

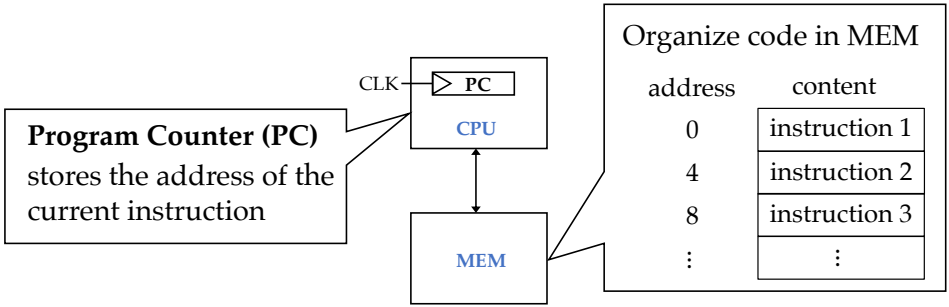
- Define the instruction set while designing the circuits.

1. Loading data and tracking execution.
2. Arithmetic and logical functions.
3. Reading and writing memory.
4. Talking to devices.

commands ⟷ implementations

WestlakeNLP

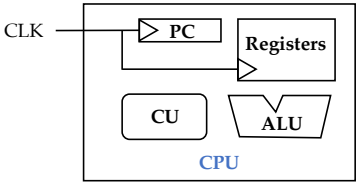
1. Loading and Executing Instructions



Why not 0, 1, 2 ?

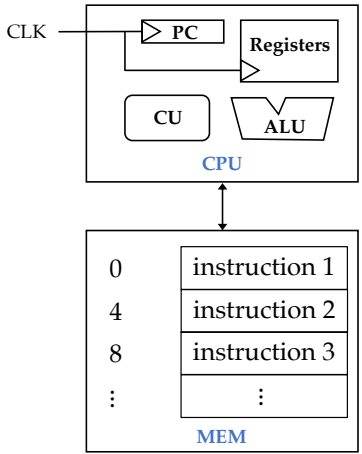
byte addressing 1 word = 4 bytes

1. Loading and Executing Instructions



- **Program Counter (PC)**
  stores the address of current instructions.
- **Control Unit (CU)**
  interpret the current instructions.
- **Registers**
  a small number of words for storing operands to compute.
- **Arithmetic Logical Unit (ALU)**
  $+ \ - \ \times \ \div$ and or not $\cdots$

1. Loading and Executing Instructions



- Machine starts.
- PC starts from 0.

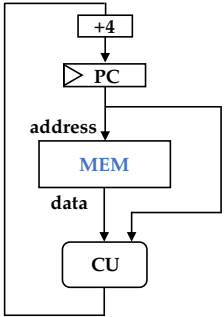
Repeat:

- Load a word (instruction) from PC address.
- CU interprets the instruction and sends signals to CPU components.
- PC increases by 4 or jumps according to CU instruction.

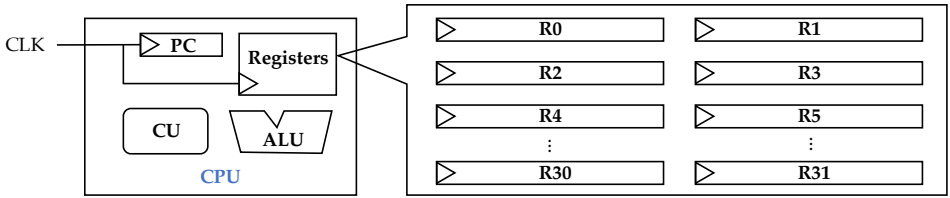
Until (HALT is the instruction):

- Machine stops.

1. Loading and Executing Instructions

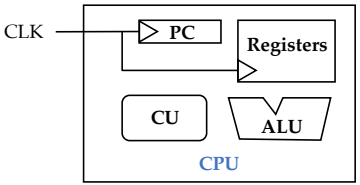


2. Arithmetic and Logical Functions



- There are 32 registers, each being a computer word, to store data for computation hardware wiring.
- CPU operations are executed with registers, which contains operands and results for ALU, address for PC, etc.
- Why start with 0 (not 1) ? Binary encoding.

2. Arithmetic and Logical Functions



The operations

$+ \quad - \quad \times \quad \div \quad \%$

$AND \quad OR \quad NOT$

$\gg \quad \ll \quad \cdots$

% —— module

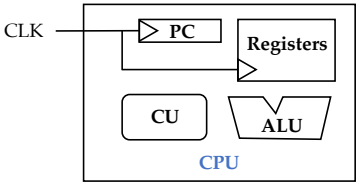
$\gg$, $\ll$ —— shift

$5 \,\%\, 2 = 1, \; 7 \,\%\, 4 = 3$

$00110010 \gg 2 = 00001100$

$00110010 \ll 3 = 10010000$

2. Arithmetic and Logical Functions



The operations

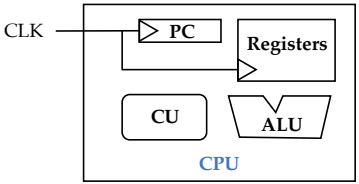
$+ \quad - \quad \times \quad \div \quad \%$

$AND \quad OR \quad NOT$

$\gg \quad \ll \quad \cdots$

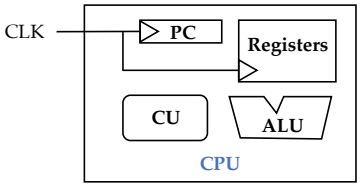
- Are these enough for a computer ?
- Yes ! We learned M $\rightarrow$ N circuit and computation theory.

2. Arithmetic and Logical Functions



- The implementation
  - How to pack ALU instructions into a computer word?
  - How to wire the circuit?

2. Arithmetic and Logical Functions



- How to pack ALU instructions into a computer word?

32 bits

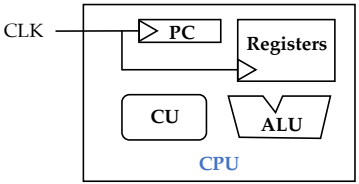
COMMAND	RESULT	OPERAND1	OPERAND2

e.g.,

+	R26	R3	R4

meaning adding the content of registers R3 and R4, and stores the result into register R26.

2. Arithmetic and Logical Functions



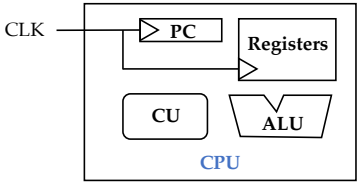
- How to pack ALU instructions into a computer word?

32 bits

COMMAND	RESULT	OPERAND1	OPERAND2

- Allocate 6 bits to encode COMMAND.
  - 000001 for +
  - 000010 for −
  - …
  - $2^6 = 64$ commands can be accommodated.

2. Arithmetic and Logical Functions



- How to pack ALU instructions into a computer word?

32 bits

COMMAND	RESULT	OPERAND1	OPERAND2

➢ Allocate 5 bits to encode the registers for result and operand.

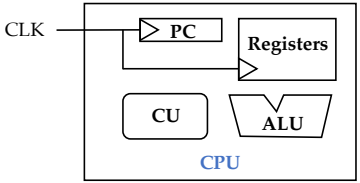
- 00000 for R0
- 00001 for R1
- 00010 for R2
- 00011 for R3

…

- 11111 for R31

✓ $2^5 = 32$ exactly
✓ binary index starts from 0

2. Arithmetic and Logical Functions



- How to pack ALU instructions into a computer word?

32 bits

COMMAND	RESULT	OPERAND1	OPERAND2

e.g.,

+	R26	R3	R4	0
6 bits	5 bits	5 bits	5 bits	11 bits

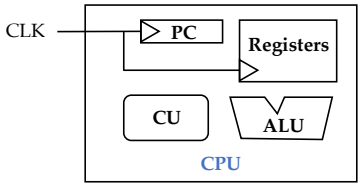
000001	11010	00011	00100	00000000000

$26 = 16 + 8 + 2 = binary\ 11010$

$\therefore R3 + R4 \Rightarrow R26$ is encoded in [00000110100001100100000000000000]

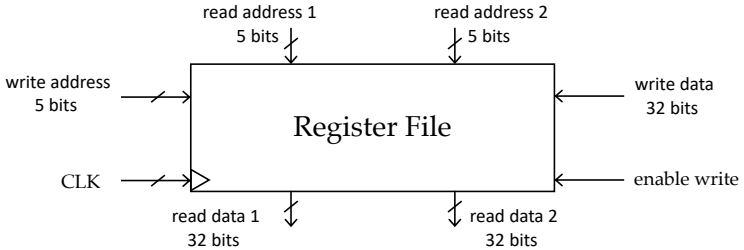
This is what each instruction looks like!

2. Arithmetic and Logical Functions



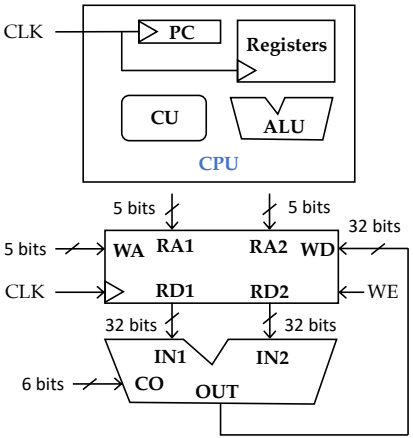
- How to wire the circuits ?

Organize the registers into a component.



Knowledge in Chapters 3 and 4 allows us to build it.

2. Arithmetic and Logical Functions

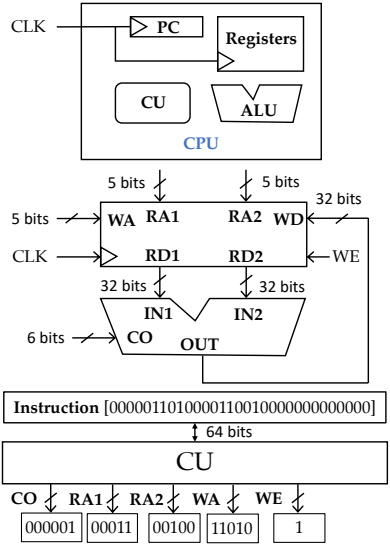


- How to wire the circuits ?

➢ Connect the registers with the ALU.

WA — write address	WD — write data
RA1 — read address 1	RA2 — read address 2
RD1 — read data 1	RD2 — read data 2
IN1 — input 1	IN2 — input 2
OUT — output	CO — operation

2. Arithmetic and Logical Functions



- How to wire the circuits ?

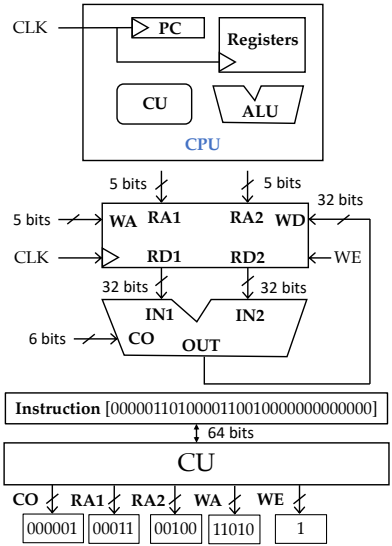
➢ Connect the registers with the ALU.

WA — write address	WD — write data
RA1 — read address 1	RA2 — read address 2
RD1 — read data 1	RD2 — read data 2
IN1 — input 1	IN2 — input 2
OUT — output	CO — operation

➢ Feed the CU with the input instruction, and let it output signals for CO, RA1, RA2, WA and WE.

- $R3 + R4 \Rightarrow R26$ after CLK!

2. Arithmetic and Logical Functions



- How to wire the circuits ?

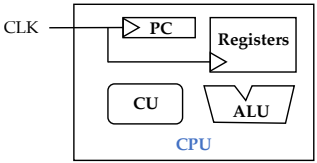
- The other instructions can be handled similarly.

  CU —— combinational logic

  ALU —— combinational logic

- Knowledge in Chapter 5 allows us to build them.

2. Arithmetic and Logical Functions



- Enriching arithmetic functions by adding constants to instructions.

6 bits	5 bits	5 bits	16 bits
COMMAND	RESULT	OPERAND1	CONSTANT

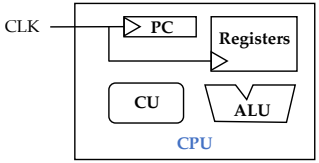
e.g., add R1 and 127, storing the results to R2.

ADDC	R2	R1	127

⇩

010001	00010	00001	0000000011111111

2. Arithmetic and Logical Functions



- Enriching arithmetic functions by adding constants to instructions.

6 bits	5 bits	5 bits	16 bits
COMMAND	RESULT	OPERAND1	CONSTANT

e.g., add R1 and 127, storing the results to R2.

ADDC	R2	R1	127

⇩

010001	00010	00001	0000000011111111

- Useful for loading constants to registers.

$65539 = \underbrace{0000000000000001}_{16\text{ bits}}\ \underbrace{0000000000000011}_{16\text{ bits}}$

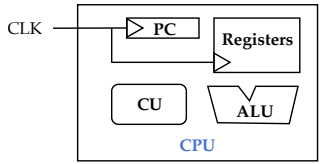
step1. $R3 - R3 \Rightarrow R3$ (set R3 to 0)

step2. $R3 + 1 \Rightarrow R3$ (set R3 to 1)

step3. $R3 \ll 16 \Rightarrow R3$ (set R3 to 65536)

step4. $R3 + 3 \Rightarrow R1$ (set R3 to 65539)

3. Reading and Writing Memory



- The instructions

  Loading

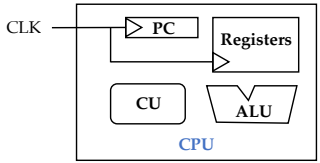
  $MEM[ADDR] \Rightarrow REG$

  Storing

  $REG \Rightarrow MEM[ADDR]$

ADDR is stored in a register.

3. Reading and Writing Memory



- The instructions

  Loading

  $MEM[ADDR] \Rightarrow REG$

  Storing

  $REG \Rightarrow MEM[ADDR]$

ADDR is stored in a register.

Let $LOAD = 31, STORE = 32$ for COMMAND

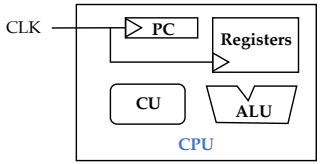
6 bits	5 bits	5 bits	16 bits
COMMAND	REG	ADDR	0

e.g., loading memory address stored in register R2 into register R1.

LOAD	R1	R2	0

100001	00001	00010	0000000000000000

3. Reading and Writing Memory



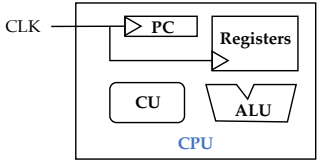
Example

Loading memory address 256 to R1

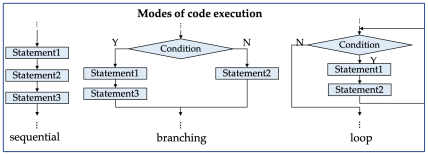
$$R3 - R3 \Rightarrow R3$$
$$R3 + 256 \Rightarrow R3$$
$$MEM[R3] \Rightarrow R1$$

- Need to extend the CPU hardwire to accommodate these.

Changing PC



- By default, the PC changes incrementally.
  $PC + 4 \Rightarrow PC$



- To supporting branching and looping, we need jumping functions.

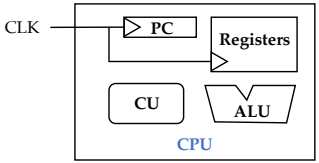
Let $JMP = 33$ if $REG = 0$, then
$PC + 4 \Rightarrow MEM[R2]$
$MEM[R1] \Rightarrow PC$

6 bits	5 bits	5 bits	5 bits	11 bits
JMP	R1	R2	REG	0

e.g., if $R3 = 0$, then jump to R1 and store old $PC + 4$ to R2.

100011	00001	00010	00011	00000000000

4. Talking to Devices

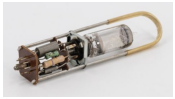


The principle

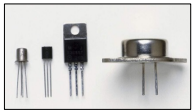
- Devices can send signals to the CPU to interrupt its current execution, in order to send signals (e.g., key press, mouse click).
- CPU can call devices to send signals (e.g., play sound).
- CPU talks to devices through agreed memory.
- More to discuss in OS lecture.

The Implementation

Vacuum Tube

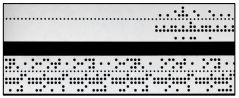


Transistor



Integrated Circuit

Paper Tape



Floppy Disk

Keyboard and Mouse

Tablet

Cloud Storage

The Implementation

- Computer Word
  byte $\rightarrow$ 16 bits machines
  $\rightarrow$ 32 bits machines
  $\rightarrow$ 64 bits machines
- CPU Speed (In the 1950s)
  from 1 kHz to a few hundred kHz
  IBM 701 $\rightarrow$ around 17 kHz
- Memory (In the 1950s)
  IBM 701 $\rightarrow$ 2048 words of 36 bits each

In the marketing of hard disk drives,
$1K = 1000(kilo)$
$1M = 1000K \approx 1million(mega)$
$1G = 1000M$
$1T = 1000G$
$1P = 1000T$
$b = bit$, $B = byte$