The von Neumann Model

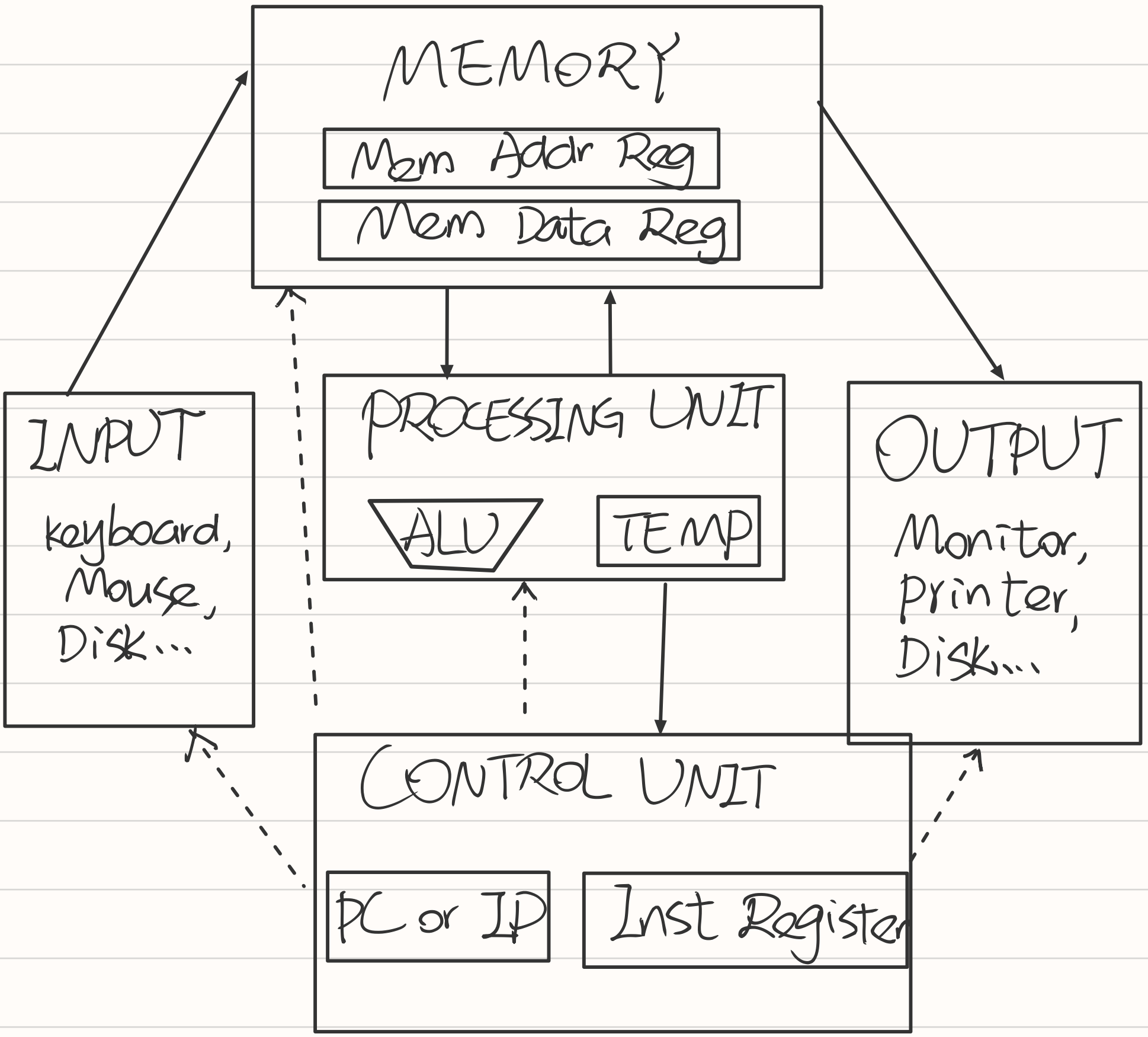
We need an execution model to process computer programs!

John von Neumann proposed a fundamental model in 1946

this model consists of 5 components:

① Memory (store the program and data)
② Processing unit
③ Input
④ output
⑤ Control Unit (control the order in which instructions are carried out)

Two examples: LC-3, MIPS



The von Neumann Model

Memory:

Memory contains bits (二进制数)
- Bits are logically grouped into Byte(字节) (8 bits)
and words(8,16,32 bits)

Address space: Total number of uniquely identifiable locations in memory

In LC-3 → 2^{16}, 16-bit address
In MIPS → 2^{32}, 32-bit address
In x86-64 → 2^{48}, 48-bit address

Word-Addressable Memory:

Each data word has a unique address

In MIPS → 32-bit data word
In LC-3 → 16-bit data word

Byte-Addressable Memory:

Each byte has a unique address ☆

MIPS and LC-3 are actually Byte-Addressable

Big-Endian vs Little-Endian: Convention

Accessing Memory: MAR vs MDR.

Two ways:

1. Reading or loading data from a memory location
2. Writing or storing data to a memory location

Two registers:

1- MAR: Memory Address Register
2. MDR: Memory Data Register

To read:

Step1: Load the MAR with address we want to read
Step2: Data in corresponding location gets placed in MDR

To write:

Step1: Load the MAR with the address and the MDR with the data we want to write
Step2: Activate Write Enable signal → value in MDR is written to address specified by MAR

Processing Unit:

- performs the actual computation(s).
- Consist of many functional units
- Start with a simple ALU: Arithmetic and Logic Unit

Fast Temporary Storage

A computer provides a small amount of storage very close to ALU to store temporary values and quickly access them later

A memory access is much slower than an addition multiplication or division.

This is usually a set of registers called Register File

LC-3 has 8 general purpose regsiters (GPRs)

R0 - R7 : 3-bit register number

Regsiter size = Word length = 16 bits

MIPS has 32 general purpose regsiters

R0 - R31 : 5-bit register number

Register size = Word Length = 32 bits

MIPS Register File Convention

Name	Register Number	Usage
`$0`	0	the constant value 0
`$at`	1	assembler temporary
`$v0-$v1`	2-3	function return value
`$a0-$a3`	4-7	function arguments
`$t0-$t7`	8-15	temporary variables
`$s0-$s7`	16-23	saved variables
`$t8-$t9`	24-25	temporary variables
`$k0-$k1`	26-27	OS temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	function return address

Control Unit (the core)

Conduct a step-by-step process of excuting (every instruction in) a program (in sequence)

- It keeps track of which instruction being processed, via Instruction Register (IR), which contains the Instruction.

- It keeps track of which instruction to process next, via Program counter (PC) or Instruction Pointer (IP), which contains the address of the (next) instruction to process.

Two key Properties:

1. Stored Program

- Instructions stored in a linear memory array
- Memory is unified between instructions and data

The interpretation of a stored value depends on the control signals.

2. Sequential Instruction Processing

- One instruction processed (fetched, executed, completed) at a time.
- PC (or IP) identifies the current instruction
- PC is advanced sequentially except for control transfer instructions.

For word-addressable memory, the processor increments the PC by 1 (LC-3)

For Byte-addressable memory, the processor increments the PC by the Instruction length in Bytes (4 In MIPS)

The Instruction:

An Instruction is the most basic unit of computer processing.

Instructions are words in the language of a computer.

Instruction set Architecture (ISA) is the vocabulary

The language of computer can be written as

Machine Language: Computer-readable representation

Assembly language: Human-readable representation

Is made up of 2 parts:

opcode: 操作码 specifies what the Instr does

operands: 操作数 specify who the Instr is to do it to

Both are specified in instruction format (or instr. encoding)

Instruction Types:

1. Operate instructions: Execute operations in the ALU
2. Data movement instructions: read from or write to memory.
3. Control instructions: Change the sequence of execution.

From Assembly to Machine Code in MIPS

- Addition

MIPS assembly

```
add $s0, $s1, $s2
```

Field Values

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32

rd ← rs + rt

Machine Code (Instruction Encoding)

op	rs	rt	rd	shamt	funct
000000	10001	10010	10000	00000	100000
31–26	25–21	20–16	15–11	10–6	5–0
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Destination Register (DR)

Source Registers (SR)

MIPS R-type Instruction Format

3 register operands

(only shift operations) shift amount

opcode

operation in R-type instructions

High-level code: a = A[2];

- MIPS Load Data

MIPS assembly

```
lw  $s3, 8($s0)
```

base + offset

$s3 ← Memory[$s0+8]

Field Values

I-Type