Review

- Semi-conductors
  - Transistor
- Logic gates
  - NOT, AND, OR, XOR, NAND, NOR, XNOR
- Combining logic gates
  - Half/Full adder
  - RS-Latch
  - Memory

Von Neuman Model and Machine Codes

Lecture 22

Hyung-Sin Kim

SNU Graduate School of Data Science

Contents

- Von Neuman Model
  - Overview
  - Each component
- Instruction Processing (Machine codes)

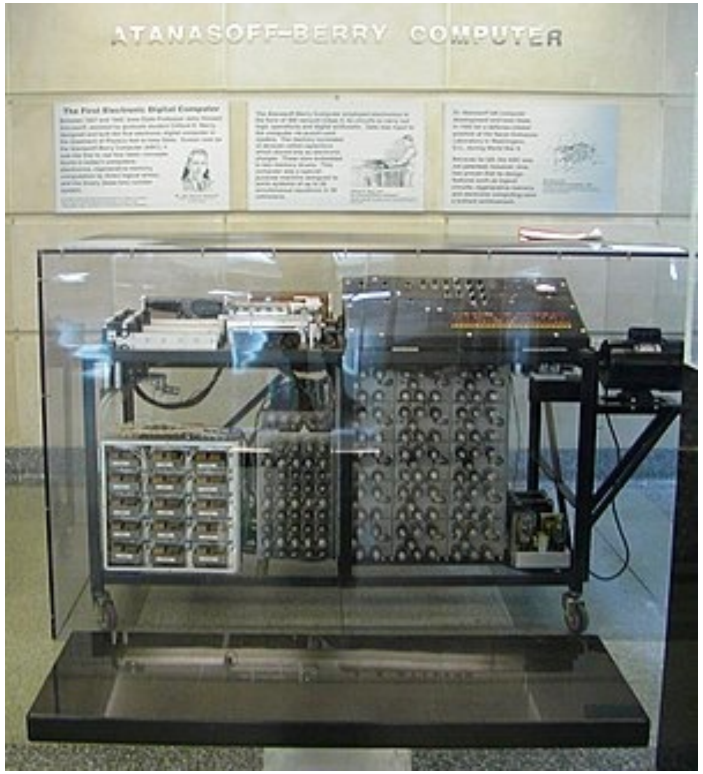
Von Neuman Model

- **Overview**
- **Each component**

Once upon a time, there was no software.
Of course, no concept of installing or removing software!

Those days, a computer was a machine that has electronic circuits designed for one specific program

So… A program was simply a hard-wired circuit…



It was ***super annoying*** *to build a different computer for every single program…*

People wanted to have a computer that can run multiple programs without changing its circuit!

The programs run in this way is called ***“soft”****ware*

*So… A **soft**ware program looked like this…* ☺

A software program was
***NOT electronically stored** in a computer*
*but **physically stored** on a set of **punchcards***

*The software program was **separate** from the computer hardware*

*The software program was given to the computer as an **input** when it needs to run*

Although more portable than a circuit program,
*it is still **NOT** a happy experience to make and manage so many groups of punchcards.* ☹

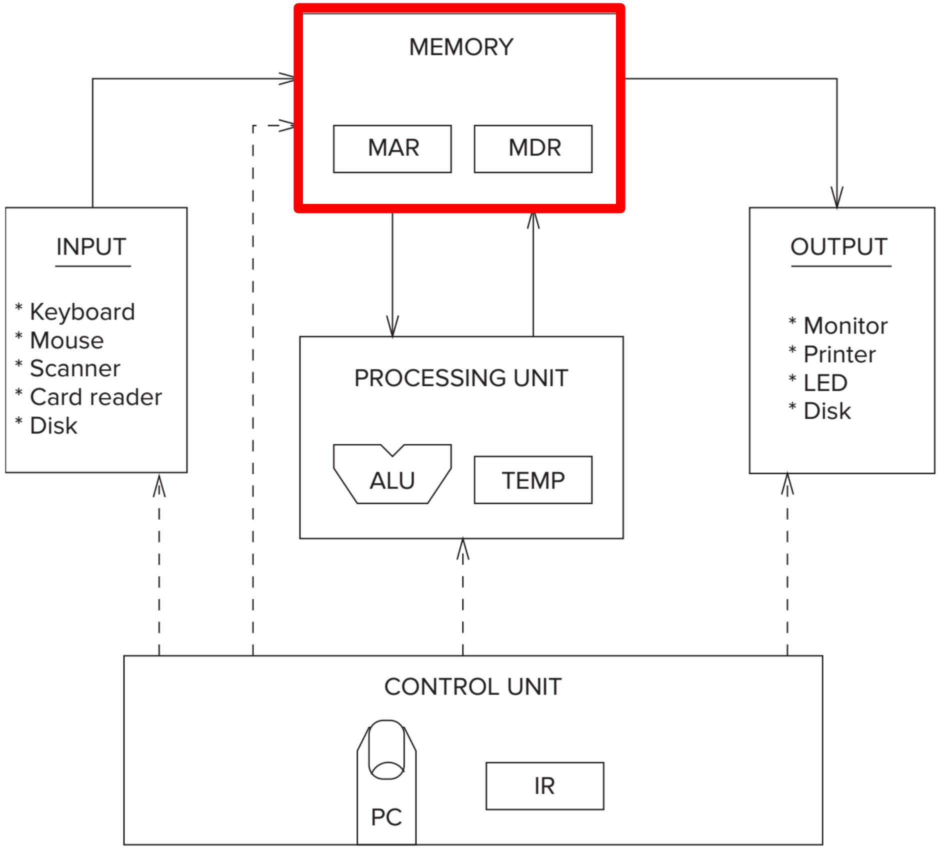
Can we **store** software programs ***electronically*** in a computer, instead of **punchcards**?

Stored program concept!

John von Neumann (1903 – 1957)

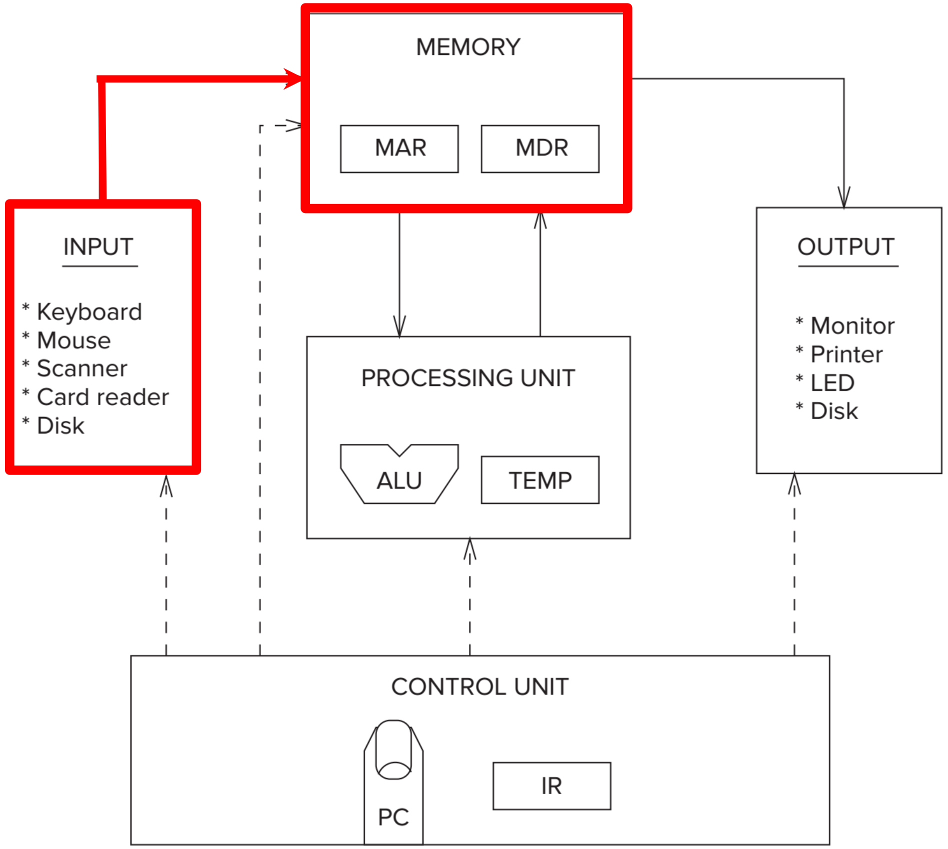
Von Neumann Model [1946]

- A program is contained in the **memory**



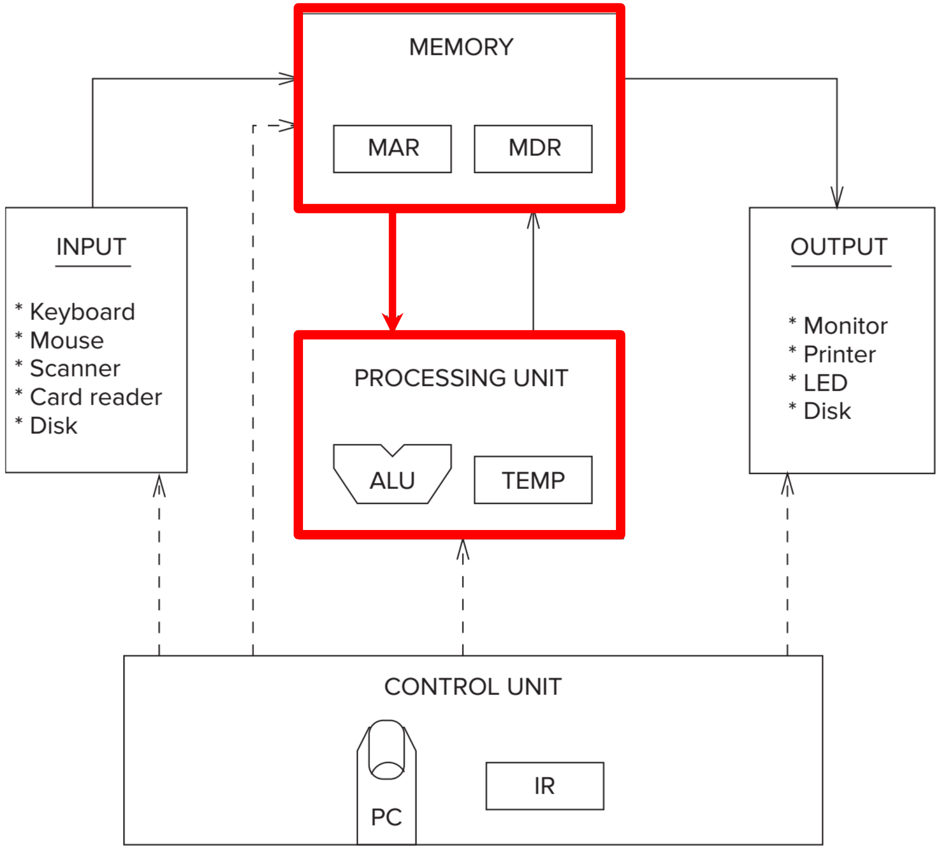
Von Neumann Model [1946]

- A program is contained in the **memory**
- Data for the program is either contained in the **memory** or obtained from the **input devices**



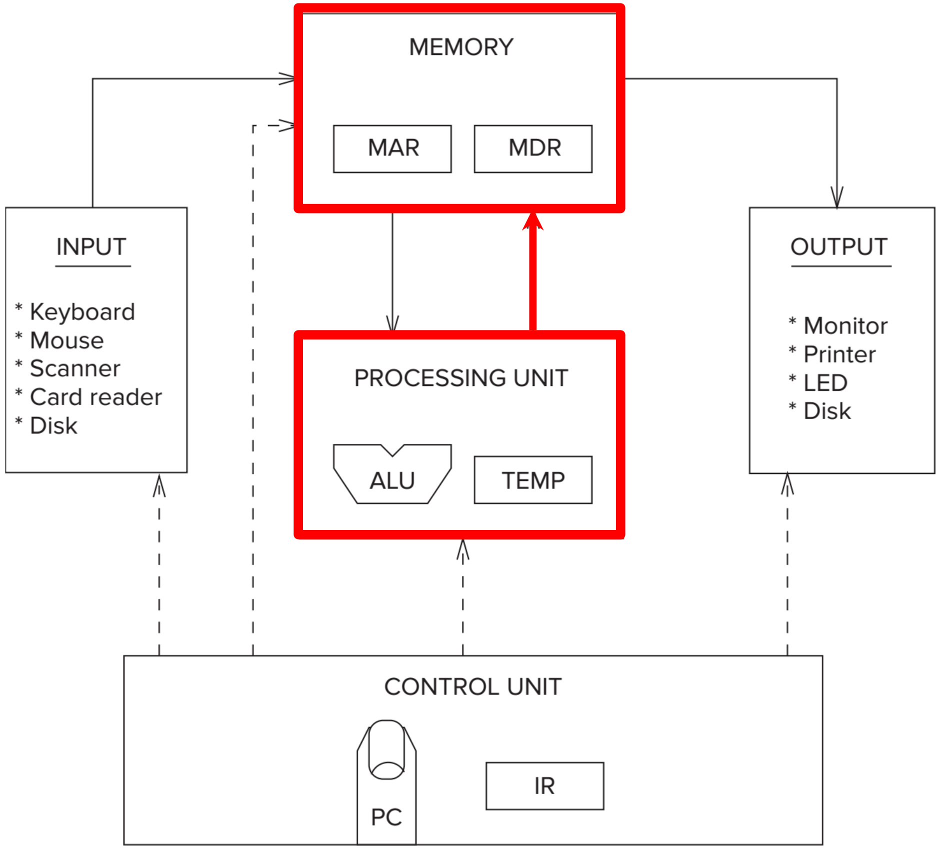
Von Neumann Model [1946]

- A program is contained in the **memory**
- Data for the program is either contained in the **memory** or obtained from the **input devices**
- A program instruction and data for its execution are sent from the **memory** to the **processing unit**



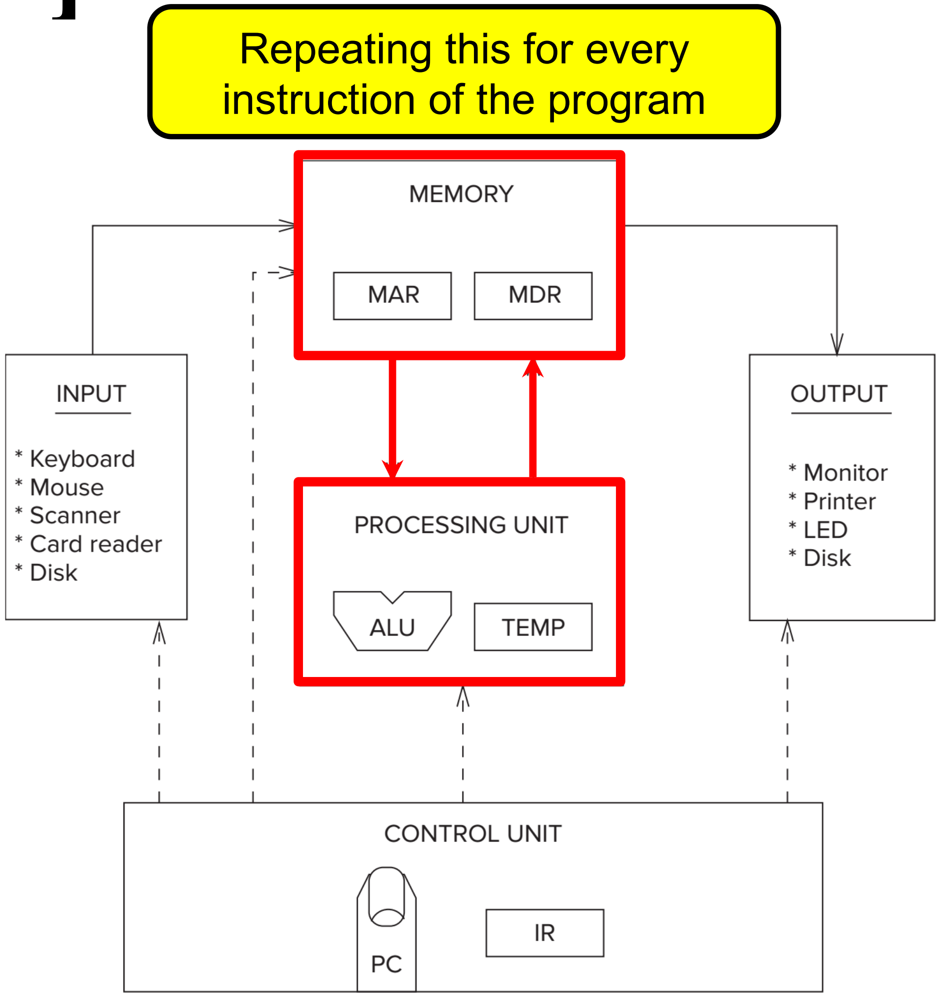
Von Neumann Model [1946]

- A program is contained in the **memory**
- Data for the program is either contained in the **memory** or obtained from the **input devices**
- A program instruction and data for its execution are sent from the **memory** to the **processing unit**
- The program instruction is executed in the **processing unit** and the result is sent to the **memory**



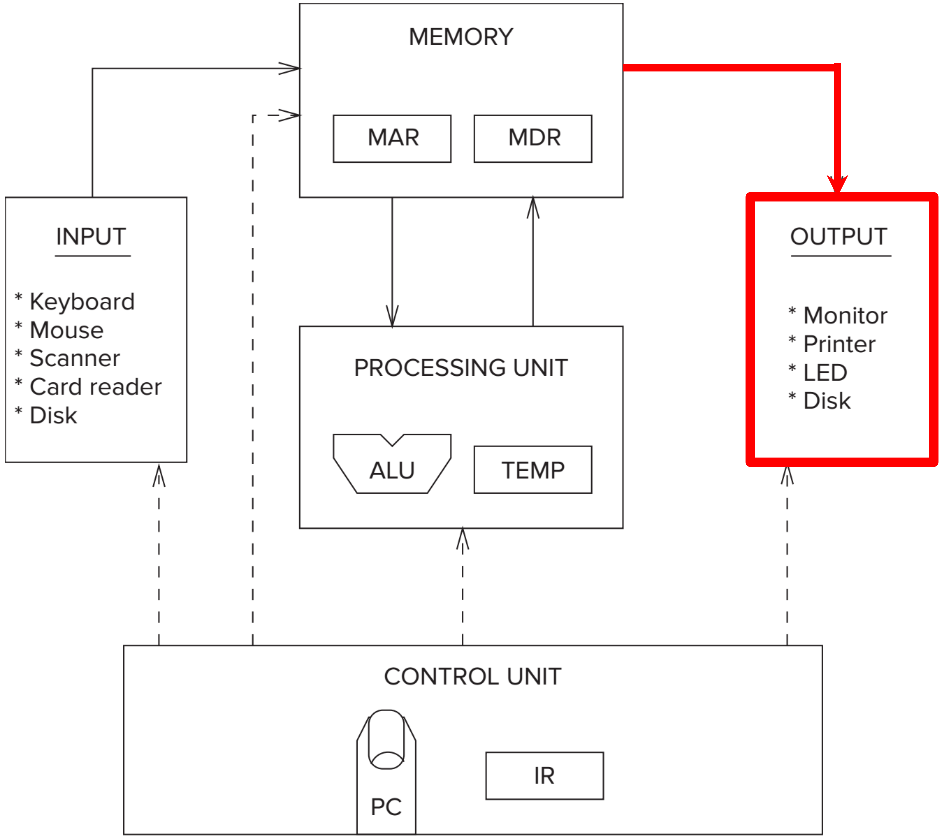
Von Neumann Model [1946]

- A program is contained in the **memory**
- Data for the program is either contained in the **memory** or obtained from the **input devices**
- A program instruction and data for its execution are sent from the **memory** to the **processing unit**
- The program instruction is executed in the **processing unit** and the result is sent to the **memory**



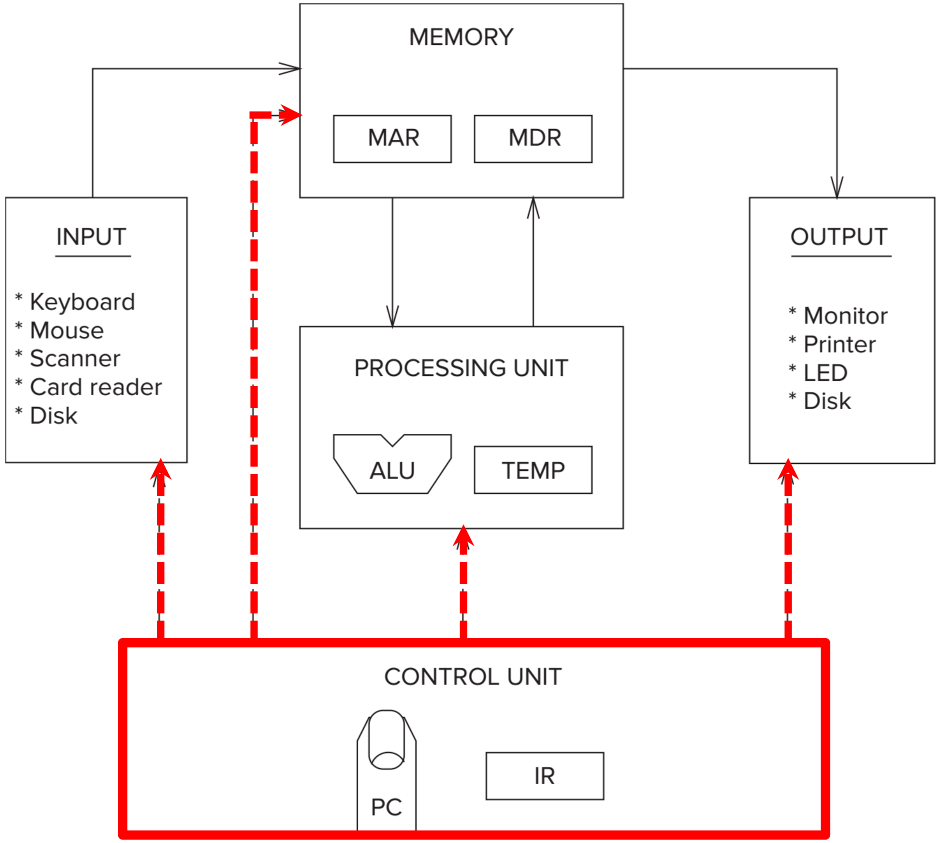
Von Neumann Model [1946]

- A program is contained in the **memory**
- Data for the program is either contained in the **memory** or obtained from the **input devices**
- A program instruction and data for its execution are sent from the **memory** to the **processing unit**
- The program instruction is executed in the **processing unit** and the result is sent to the **memory**
- The program results are provided by the **output devices**



Von Neumann Model [1946]

- A program is contained in the **memory**
- Data for the program is either contained in the **memory** or obtained from the **input devices**
- A program instruction and data for its execution are sent from the **memory** to the **processing unit**
- The program instruction is executed in the **processing unit** and the result is sent to the **memory**
- The program results are provided by the **output devices**
- The entire procedure is managed by the **control unit**, which should make sure that everything is OK

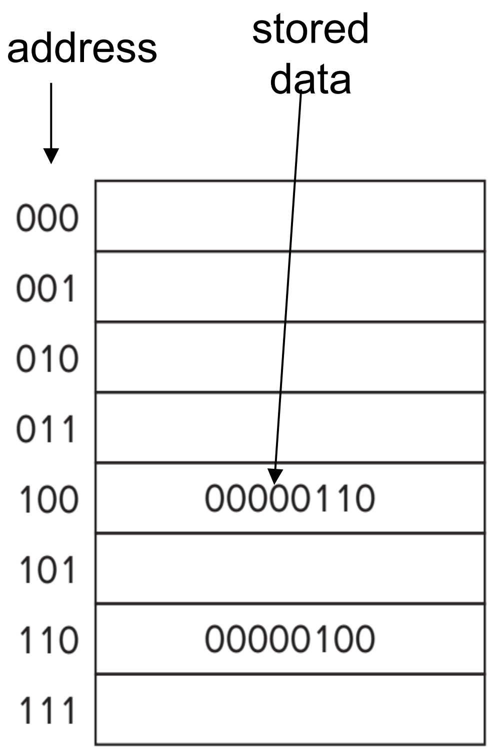


Von Neuman Model

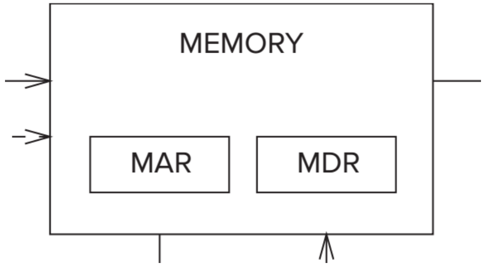
- Overview
- **Each component**

Von Neumann Model – Memory

- Memory has many memory **locations**, each of which has its **address** and scan/store **data**
- Memory size depends on how many locations it has and how many bits each location stores
  - One location usually stores 8 bits (1 byte)
- For example, if memory has 2^{34} distinct locations, each of which stores 8 bits (1 byte) of information…
  - 16 GB memory (16 giga locations x 1 byte)
  - It needs 34 bits to represent memory addresses



Von Neumann Model – Memory



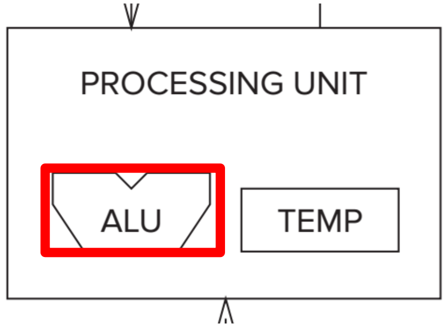
- Two registers for memory operation (writing and reading)
  - MAR: memory's address register
  - MDR: memory's data register
  - **Register**: a small amount of fast storage that the processor can quickly access

- Example 1: Storing a value A to a memory location B
  - Write location B to MAR, write value A to MDR, and execute storing operation
  - Information contained in the MDR will be written into the memory location B (indicated by the MAR)
- Example 2: Loading a value A from a memory location B
  - Write location B to MAR and execute loading operation
  - Information contained in the memory location B (indicated by the MAR) will be written to MDR

Von Neumann Model – Processing Unit

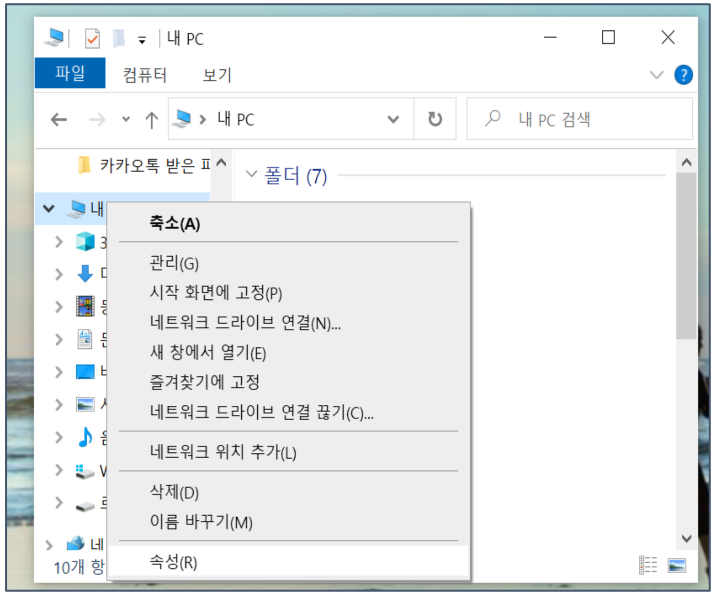
- A modern computer's processing unit consists of many complex functional units (division, square root …)

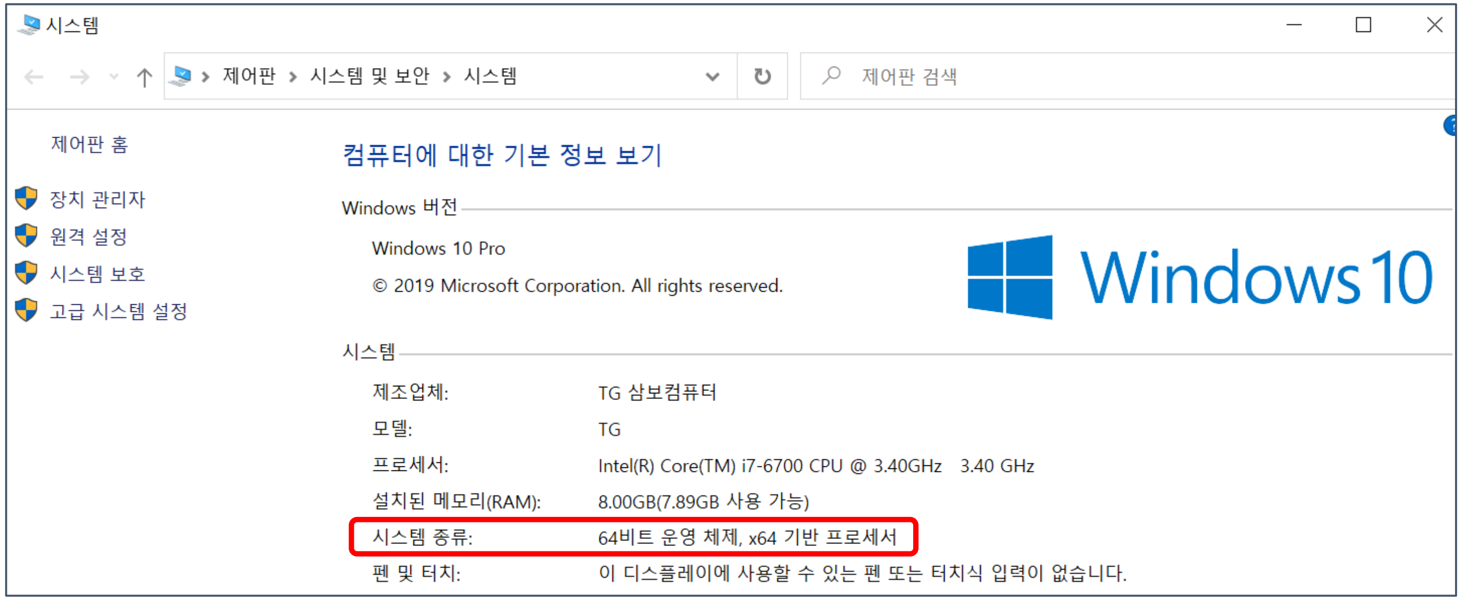
- The simplest processing unit is the **ALU** (arithmetic and logic unit)
  - Basic arithmetic functions (e.g., ADD and SUBTRACT)
  - Basic logic operations (e.g., bit-wise AND, OR, and NOT)



- ALU normally processes data elements of a **fixed size**
  - The data elements are called **words**
  - For example, to perform ADD operation, ALU receives two words as inputs and produces a single word (the sum) as output
  - Most microprocessors in modern computers have a word length of **64 bits** or **32 bits**

Von Neumann Model – Processing Unit



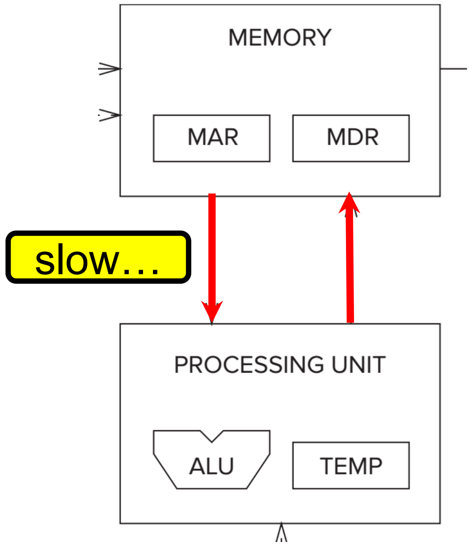


Von Neumann Model – Processing Unit

- Word size is like the number of lanes on a road

Von Neumann Model – Processing Unit

- When ALU performs multiple operations…
  - **(A+B)xC**
  - (1) Get inputs A and B from memory
  - (2) Process A+B and produce the result T
  - (3) Store the result T in a memory location
  - (4) Get inputs T and C from memory
  - (5) Process TxC and produce the result R
  - (6) Store the result R in a memory location



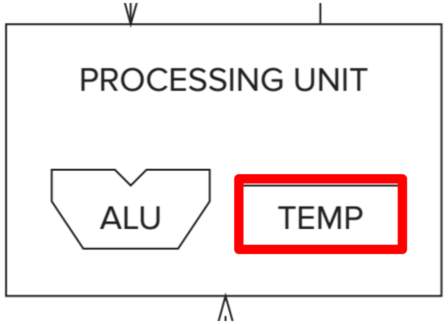
- It takes longer to access memory than perform ADD or MULTIPLY

- We want to **minimize memory access** in the middle of operations!

Von Neumann Model – Processing Unit

- Almost all computers have small amount of storage very close to the ALU to allow results to be **temporarily stored** if they will be needed to produce additional results in the near future

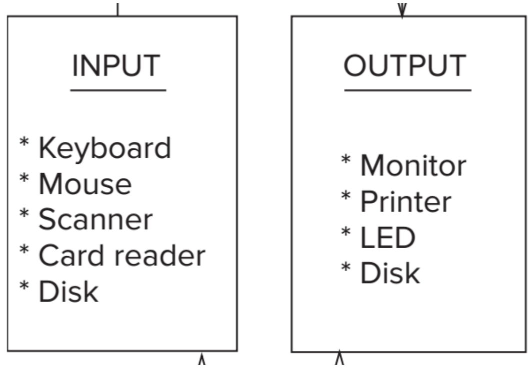
- The temporary storage comprises a set of **registers**
  - Again, a register is a small amount of fast storage



- The size of information stored in each register is **identical** to that of information processed in the ALU
  - A 64-bit processor's register stores 64-bit of information

Von Neumann Model – 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) holding ASCII code of a key struck
        - Status register (KBSR) holding status information about the keys struck
    - Monitor
        - Data register (DDR)
        - Status register (DSR)



Von Neumann Model – Control Unit

- The control unit is like the conductor of an orchestra
  - In charge of making all the other parts play together harmonically
- To this end, it should keep track of which instruction is being executed
- Instruction register (IR)
  - A register that contains the **instruction** currently being executed
- Program counter (PC)
  - A register that contains the memory **address** where the next instruction is stored

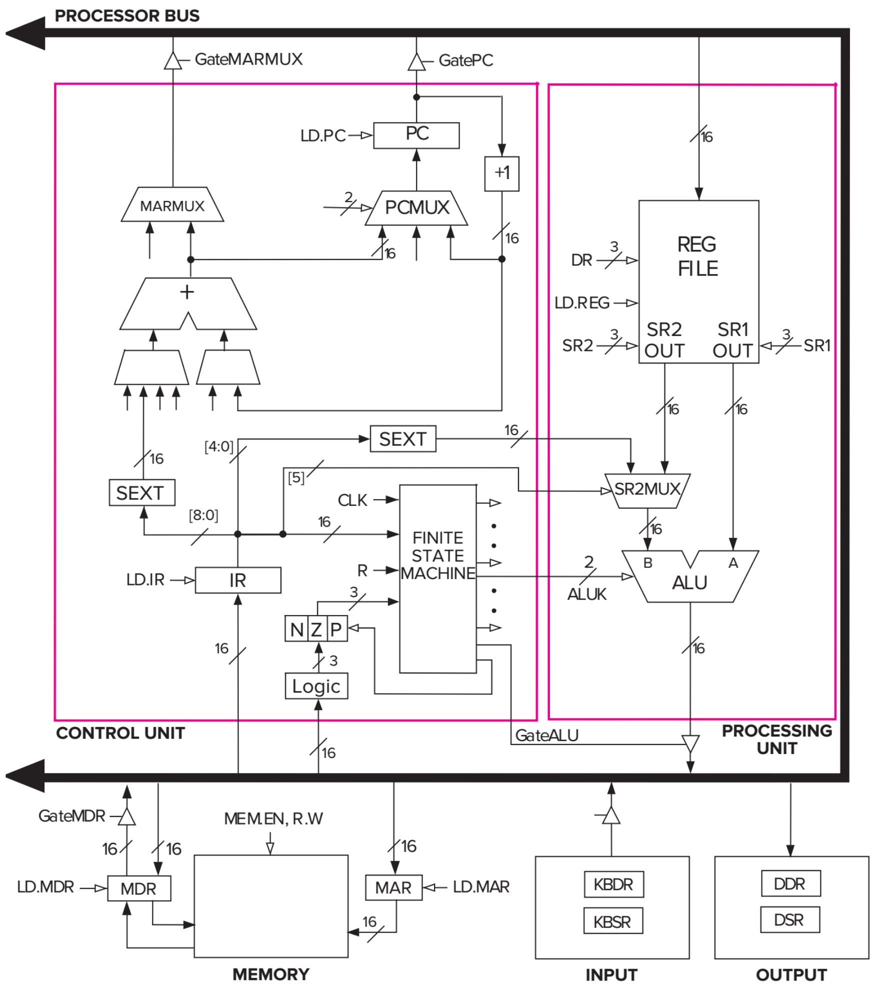
Von Neumann model

The program and data are both electronically stored in the computer's memory, as <u>sequences of bits</u>

The program is executed in the processing unit,
<u>one instruction at a time,</u>
under the direction of the control unit

Von Neumann Model – LC-3

- LC-3 (Little Computer 3)
  - An example von Neumann computer
  - Used for teaching
  - 16-bit word size
  - 8 registers in the processing unit



How does it represent an instruction as a bit sequence?

How does it process an instruction?

Instruction processing (Machine codes)

Computing Bootcamp

Instruction

- **Instruction** is the most basic unit of computer processing
  - Opcode: operation to be performed
  - Operands: data/locations to be used for operation

- Instruction Set Architecture (ISA)
  - A computer's instructions and their formats (bit sequences) / semantics

- **LC-3** (little computer 3): An example von Neumann machine
  - Its processing unit has temporary storage comprising **8 registers (R0~R7)**, each of which contains **16 bits**
  - An instruction of LC-3 consists of **16 bits** (i.e., one word)
    - Bits [15:12]: opcode
    - Bits [0:11]: operands

Instruction – ADD

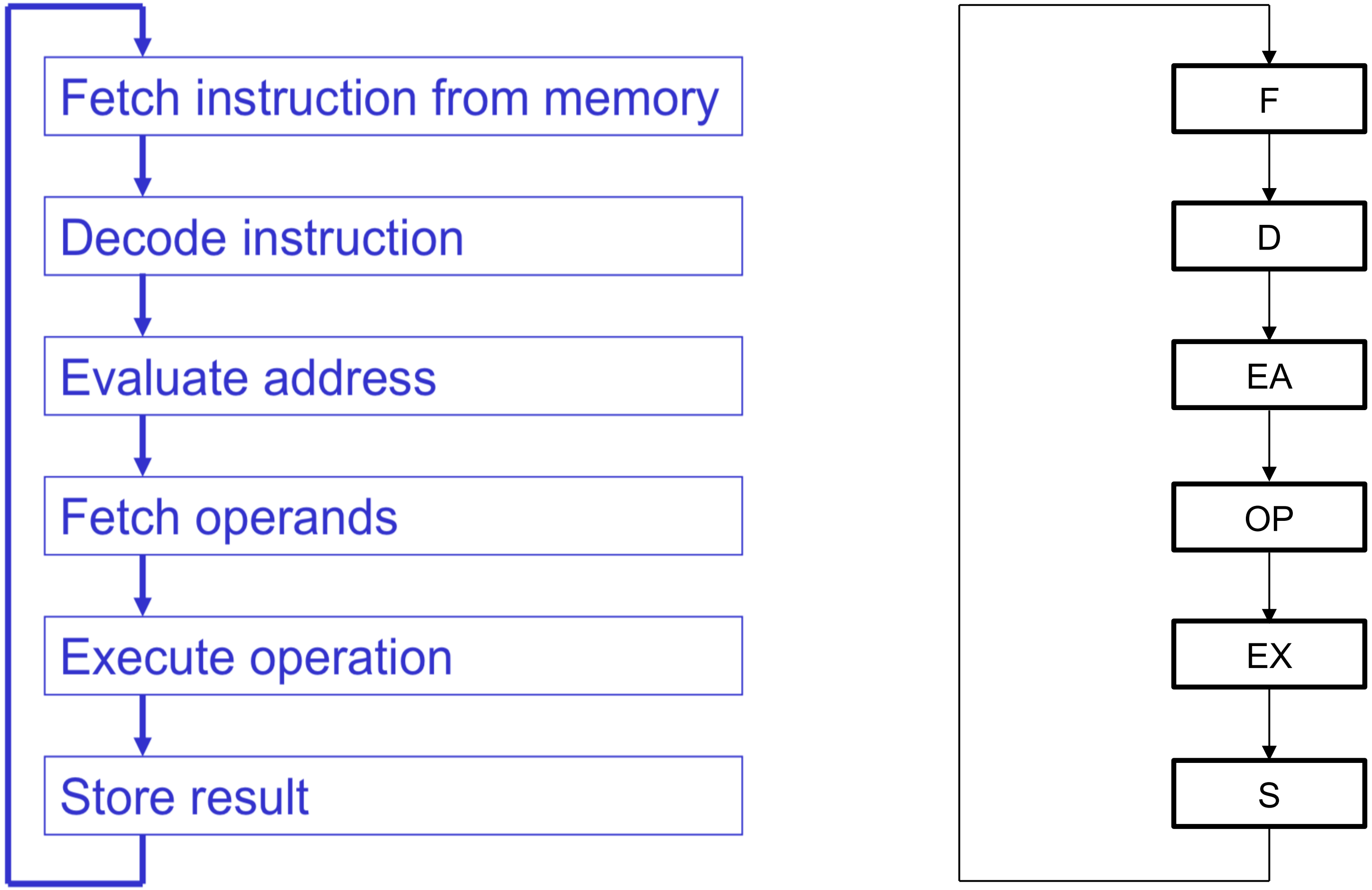
- Bit [15:12]: Opcode
- Bit [9:11]: Register where the result will be stored
- Bit [6:8]: Register where the input 1 (source 1) is stored
- Bit [0:2]: Register where the input 2 (source 2) is stored

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="4">ADD</td><td colspan="3">Dst</td><td colspan="3">Src1</td><td>0</td><td>0</td><td>0</td><td colspan="3">Src2</td></tr>
</table>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0

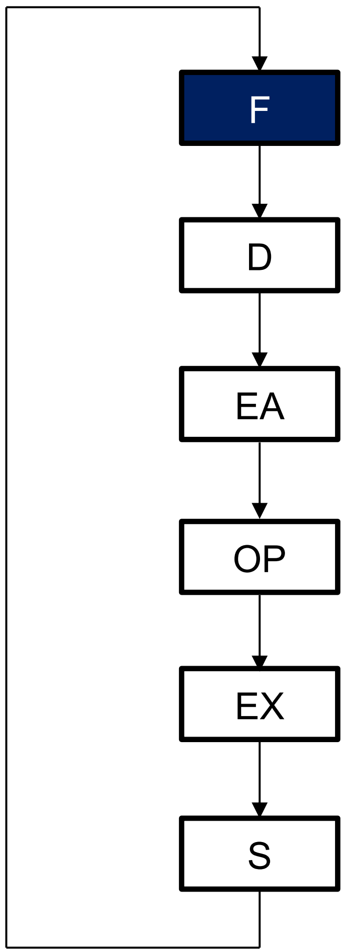
"Add the value in R2 to the value of R6, and store the result in R6

Instruction Processing



Instruction Processing – Fetch

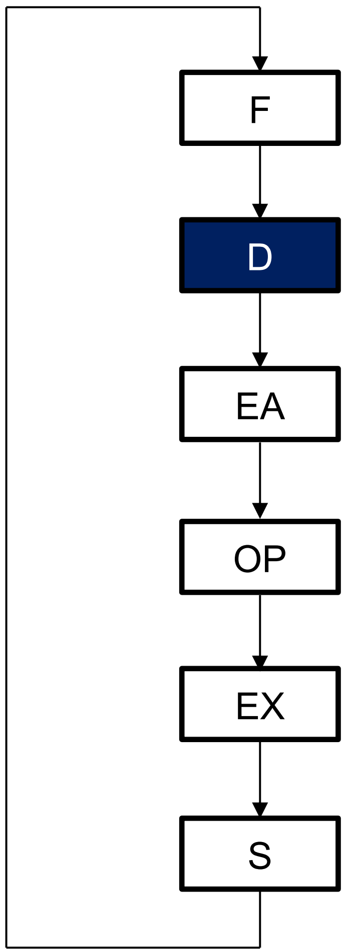
- Load the instruction in the memory location (indicated by **PC**) into **IR**
  - Copy the value in **PC** (control unit) into **MAR** (memory)
  - Send “read” signal to memory
  - Now the instruction is stored in **MDR** (memory)
  - Copy the value in **MDR** into **IR** (control unit)
- Now **IR** stores the instruction represented as a sequence of 16 bits
- Increment **PC**, so that it points to the memory location where the next instruction is stored



Instruction Processing – Decode

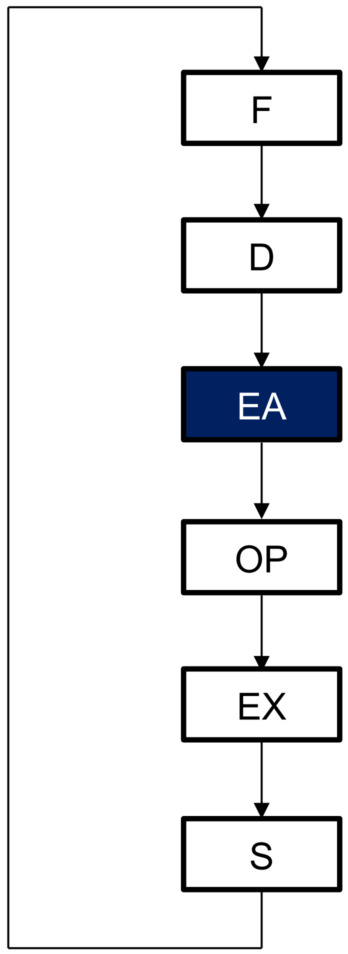
- Identify the opcode (first 4 bits) – operation ID (ADD, SUBTRACT…)

- Depending on opcode, identify other operands from the remaining 12 bits
  - For ADD,
    - Bit [9:11]: Register where the result will be stored
    - Bit [6:8]: Register where the input 1 (source 1) is stored
    - Bit [0:2]: Register where the input 2 (source 2) is stored



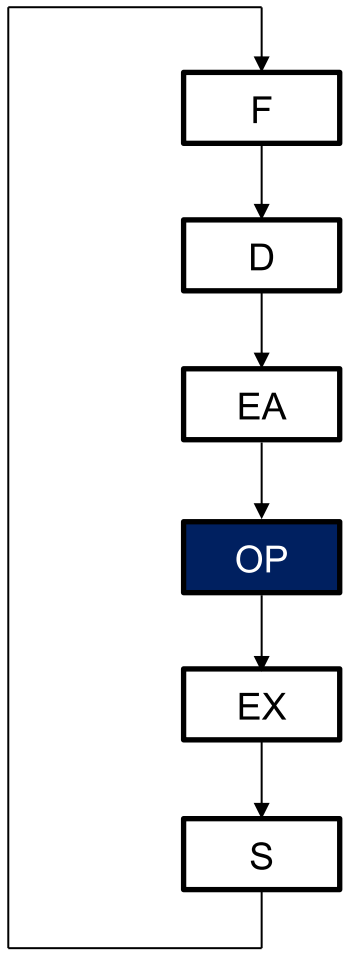
Instruction Processing – Evaluation Address

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



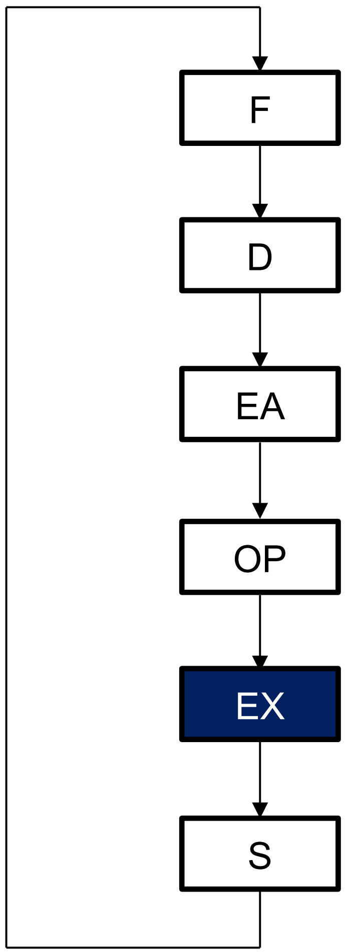
Instruction Processing – Fetch Operands

- For instructions that require memory access, compute address used for access
- Obtain source operands needed to perform operation
  - From registers in ALU



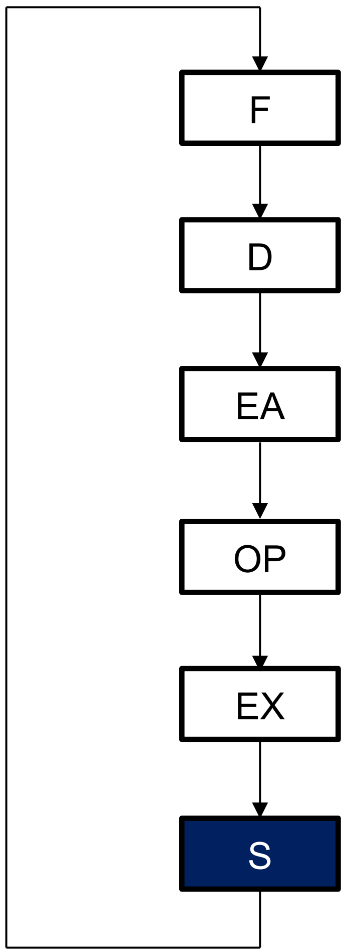
Instruction Processing – Execute

- For instructions that require memory access, compute address used for access
- Obtain source operands needed to perform operation
  - From registers in ALU
- Perform the operation (indicated by opcode) using the source operands



Instruction Processing – Store Result

- For instructions that require memory access, compute address used for access
- Obtain source operands needed to perform operation
  - From registers in ALU
- Perform the operation (indicated by opcode) using the source operands
- Write results to destination register (or memory)



Summary

Summary

- Von Neumann model
  - Main concept
  - Memory
  - Processing unit
  - Input & Output
  - Control unit

- Instruction processing
  - Fetch instruction from memory – Decode instruction – Evaluate address – Fetch operands – Execute operation – Store result

Q&A

Any questions?

Thanks!