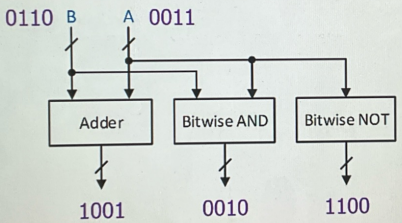
Basic Processor Model

The Arithmetic Logic Unit (ALU)

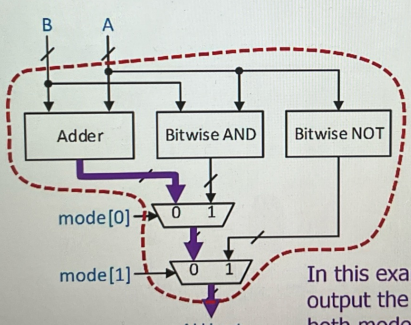
- A set of logic structures that perform a set of different possible computations on the inputs
  - One or more arithmetic operations (e.g., add, subtract)
  - One or more logic operations (e.g., AND, OR, NOT)

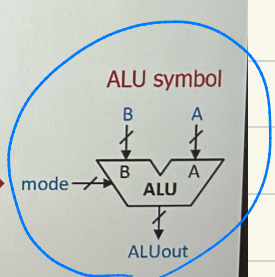


Bitwise: each result bit calculated by applying operation to corresponding position of the operand(s)

The Arithmetic Logic Unit (ALU)

- Includes a set of multiplexers that, based on a set of control signals, choose which result to send to the ALU output

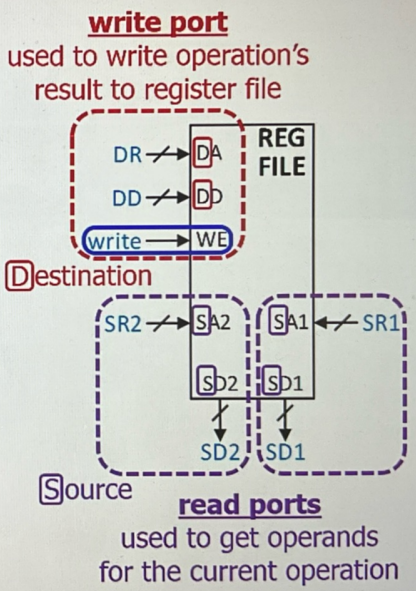




In this example, the ALU will output the sum of its inputs if both mode bits are equal to 0

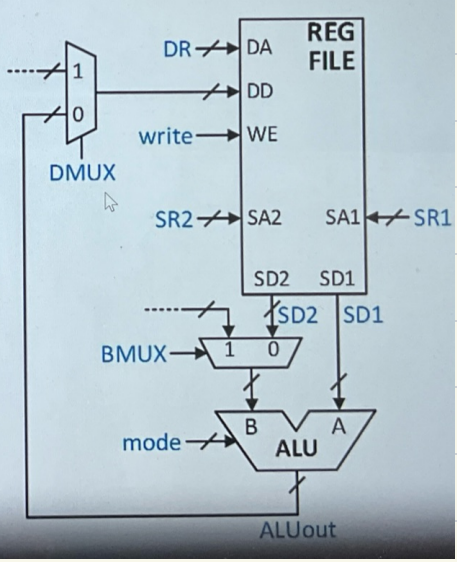
The Register File

- A **register file** is a small memory located near the ALU
  - Read from multiple locations and write to one location
  - Address inputs control which registers are accessed
  - Write enable controls whether the destination input is stored or not

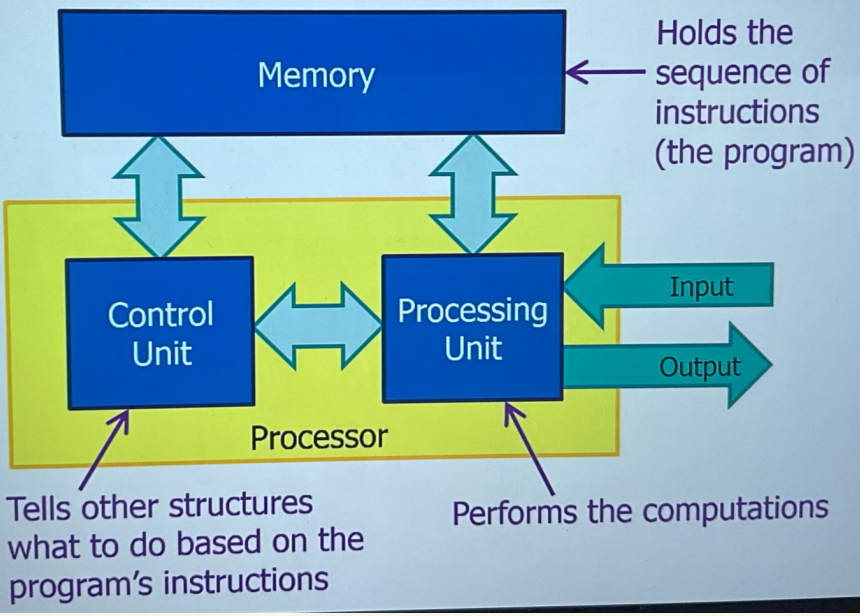


The Processing Unit

- Contains the **ALU** and the **register file**
- Control signals dictate which operation is performed on which data and where the result is stored



Conceptual Compute Model

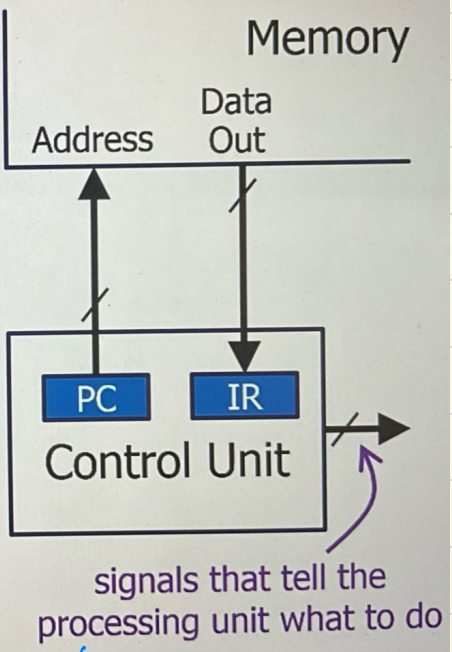


Stored Program Computer

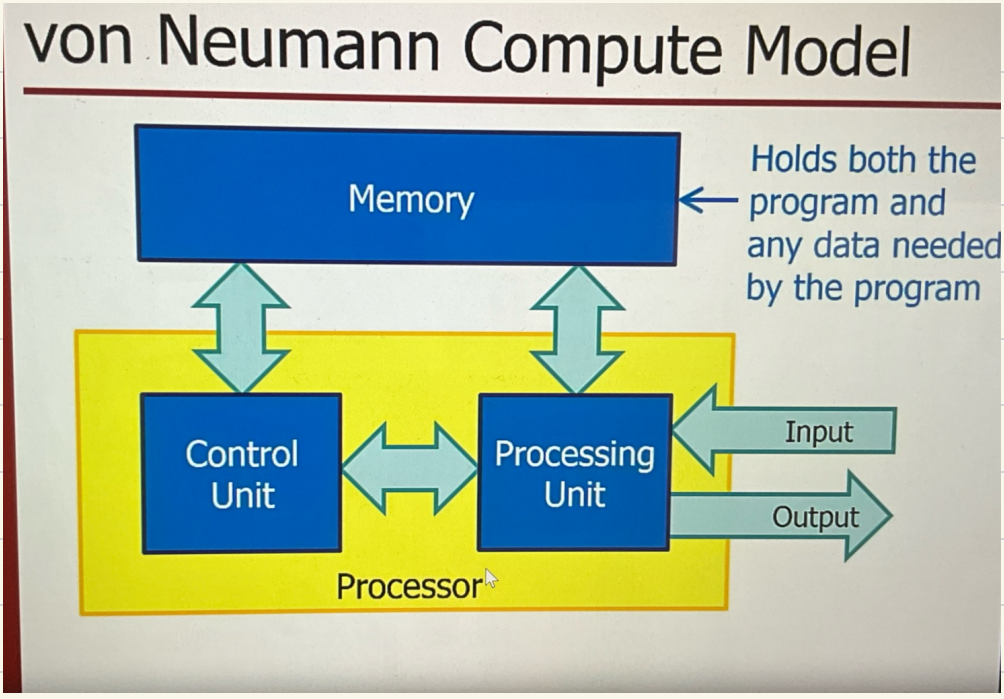
- A program is a sequence of instructions, each of which describes one "step" of the program
  - If we put a different sequence of instructions into memory, the computer performs a different task
- To execute a program, the computer repeatedly:
  - Reads the next instruction from memory
  - Does the operation described by that instruction
- We will assume a computer processes one instruction at a time…

Control Unit

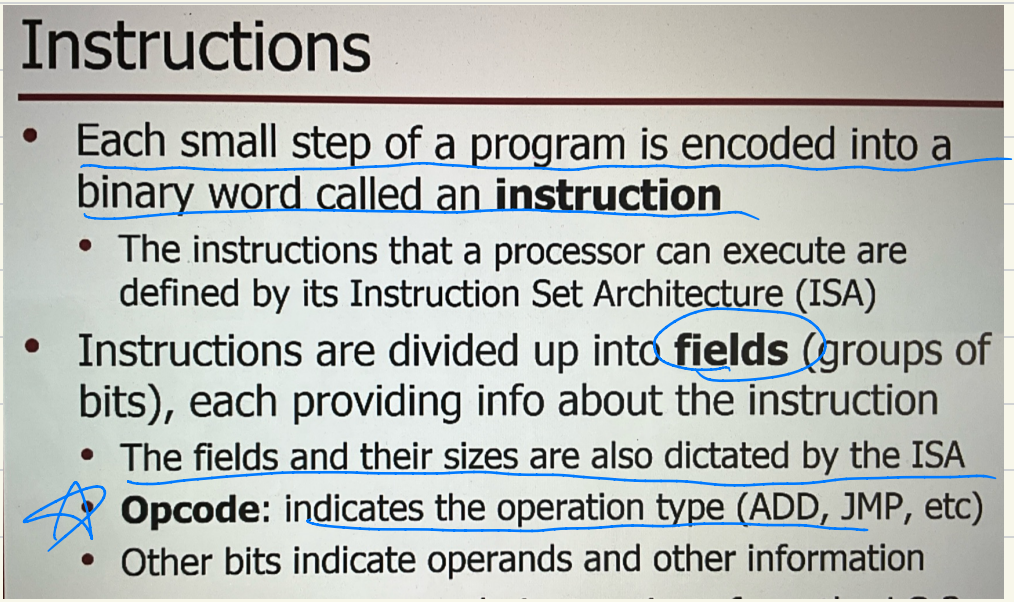
- Fetches instructions one by one from memory
- Tells each part of the computer what to do to execute each instruction
- Contains two special registers:
  - Program Counter (**PC**): contains the **address** of the **next** instruction
  - Instruction Register (**IR**): contains the current **instruction**



★ (Processing Unit contains register file and ALU)


von Neumann Compute Model
Memory
Holds both the program and any data needed by the program
Control Unit
Processing Unit
Input
Output
Processor

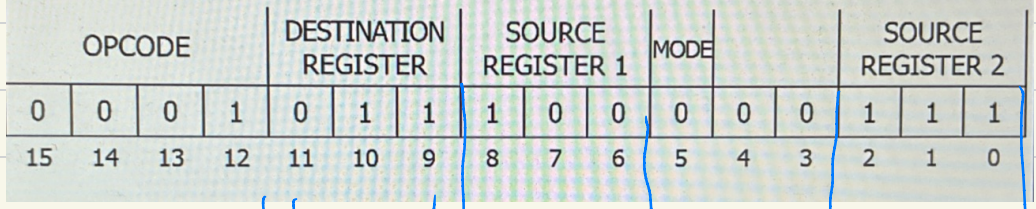
Instructions and Instruction Cycle



★ Instructions in computer are in binary codes, the codes themselves are divided into fields, where "Opcode" decides the operation type of the instruction.

Instructions

- **Example**: LC-3 instruction to compute R3 ← R4 + R7



OPCODE				DESTINATION REGISTER			SOURCE REGISTER 1			MODE			SOURCE REGISTER 2		
0	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Add Instruction

(value is 4)
∴ R4 register

(value is 7)
∴ R7 register

(Value is 3)
∴ Destination R3 register

- Five fields: opcode, destination register, source register 1, mode, source register 2
- Each field represents a separate piece of information about the instruction

Operate Instructions

- Use the ALU to perform a computation
- Operands include value(s) in register file and possibly a constant encoded in the instruction
- **Example**: R3 ← R4 + R7

OPCODE				DEST REG			SRC REG 1			MODE			SRC REG 2		
0	0	0	1	0	1	1	1	0	0	0	0	0	1	1	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

increment by 1

- **Example**: R0 ← R0 + 1

OPCODE				DEST REG			SRC REG 1			MODE	5-bit CONSTANT				
0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Indicates the addition "sub-type"

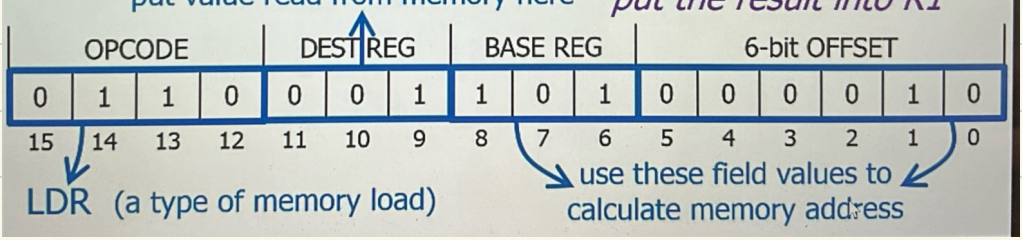
☆ (mode is the MUX's input to choose)

Data Movement Instructions

- Copy data between the memory and register file
  - Allows processing of more data than can fit in the register file (which is often relatively small)
- **Load** – copy from memory to the register file
- **Store** – copy from the register file to memory

- **Example:** R1 ← mem[R5 + 2]

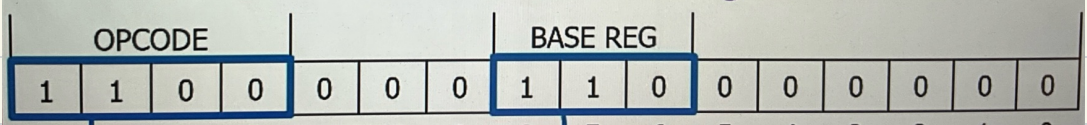
read from memory using an address 2 greater than contents of R5, and put the result into R1



Control Instructions

- Change the sequence of execution by changing the value in the Program Counter (PC)
  - Loops: need to execute instructions more than once
  - If/else conditions: need to skip certain instructions
  - Subroutines (similar to functions or methods in C/Java)

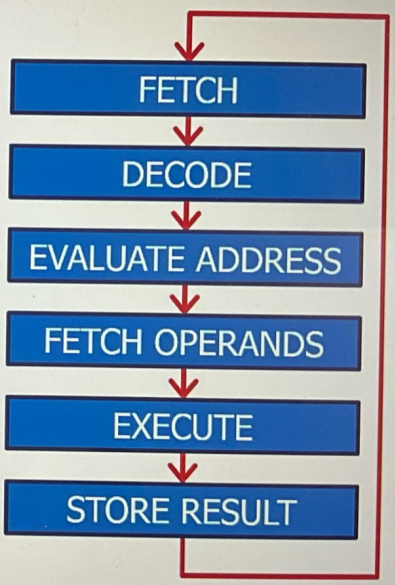
- **Example:** PC ← R6 *overwrite the PC with the value in register R6*



☆(It means now the PC will point to the R6's address for next execution)

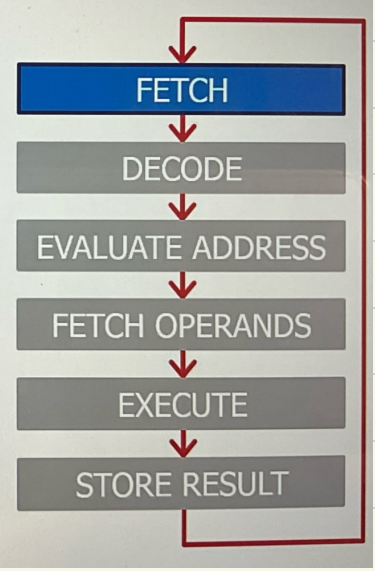
Instruction Cycle

- Describe the different **phases** of execution of each instruction
  - Not all phases required for every type of instruction
- Repeat forever
- The control unit performs different tasks in each phase
  - It is a **finite state machine**



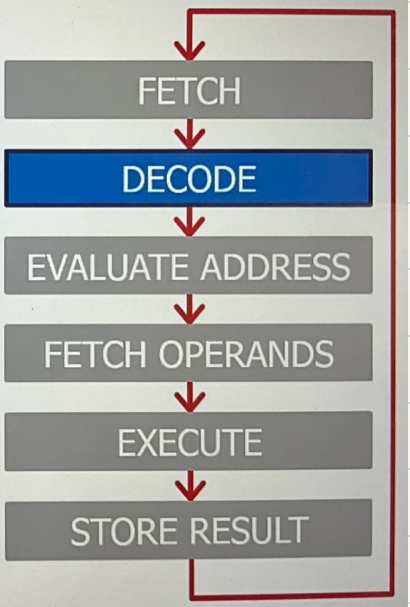
The FETCH Phase

- Read an instruction from memory
  - Address of the instruction contained in the **PC**
- Put value read from memory into the **IR**
  - The binary value that represents the instruction
- Increment the **PC** to "point" to the next instruction in memory (the next memory address)



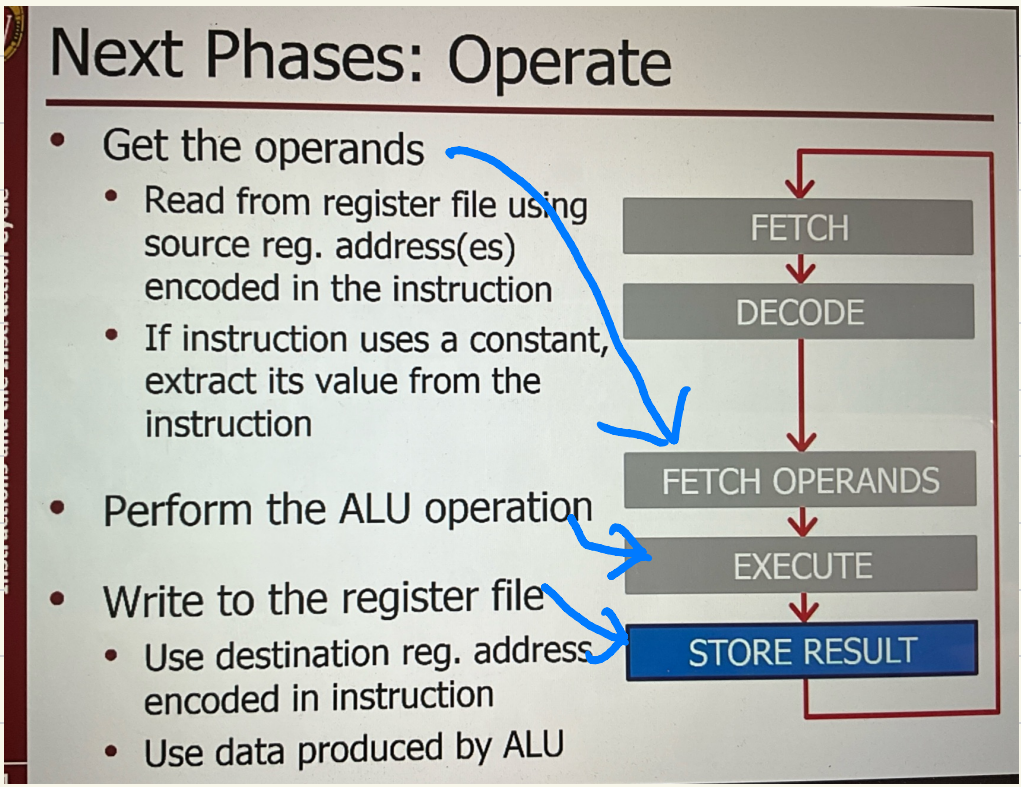
The DECODE Phase

- Determine what the instruction says the processor should do
  - The instruction type is indicated by the **opcode**
- Identify **operands** for the instruction based on the bit values in the instruction's **fields**



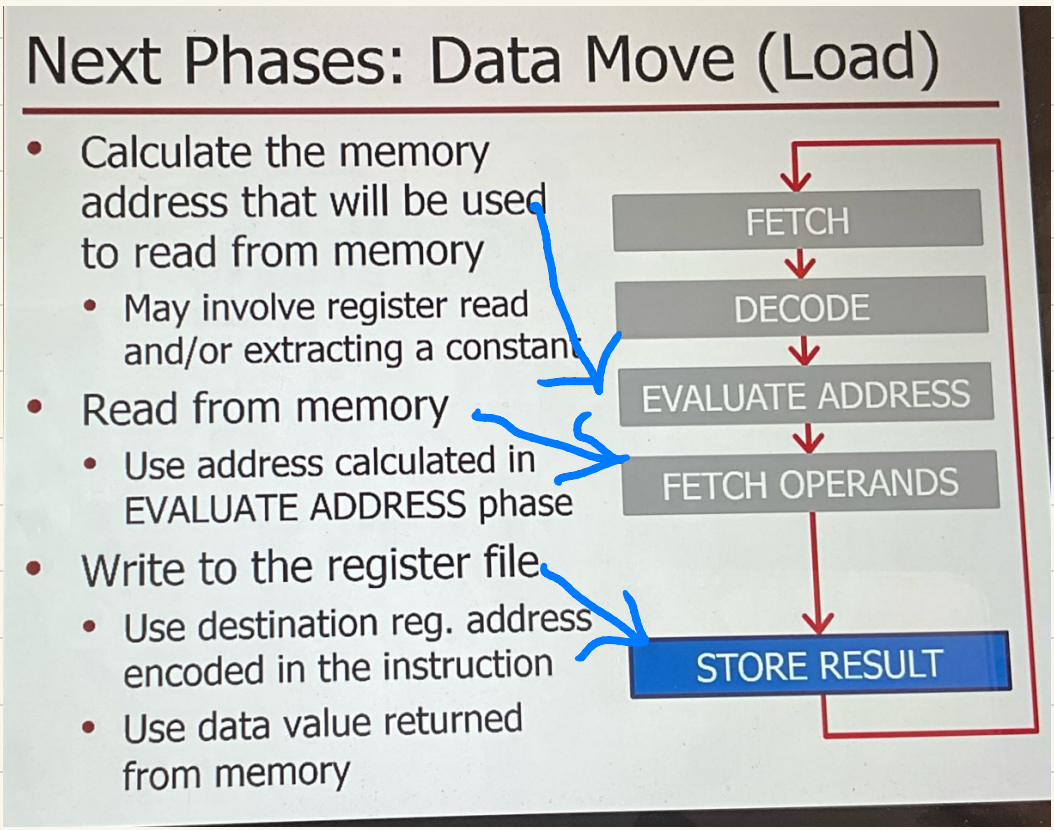
☆ Both Fetch / Decode are needed in any operation type

For Operate:

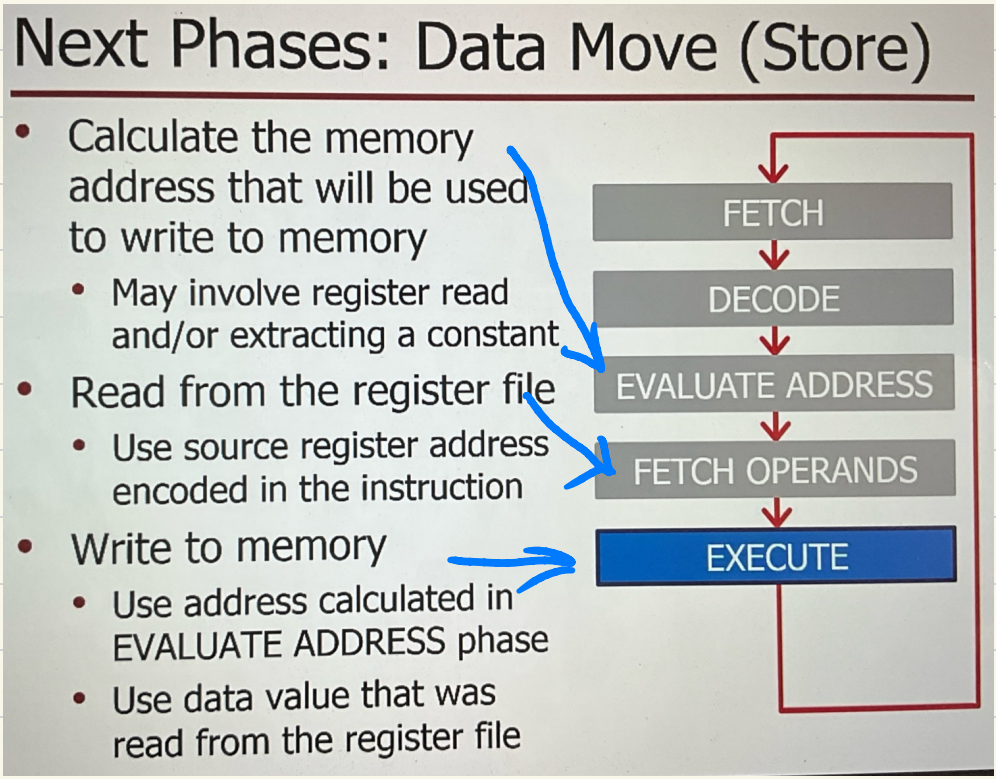


1. Get operands from source addresses
2. Perform the operation
3. Put / write the result back in register file

For Data movement : (Load)

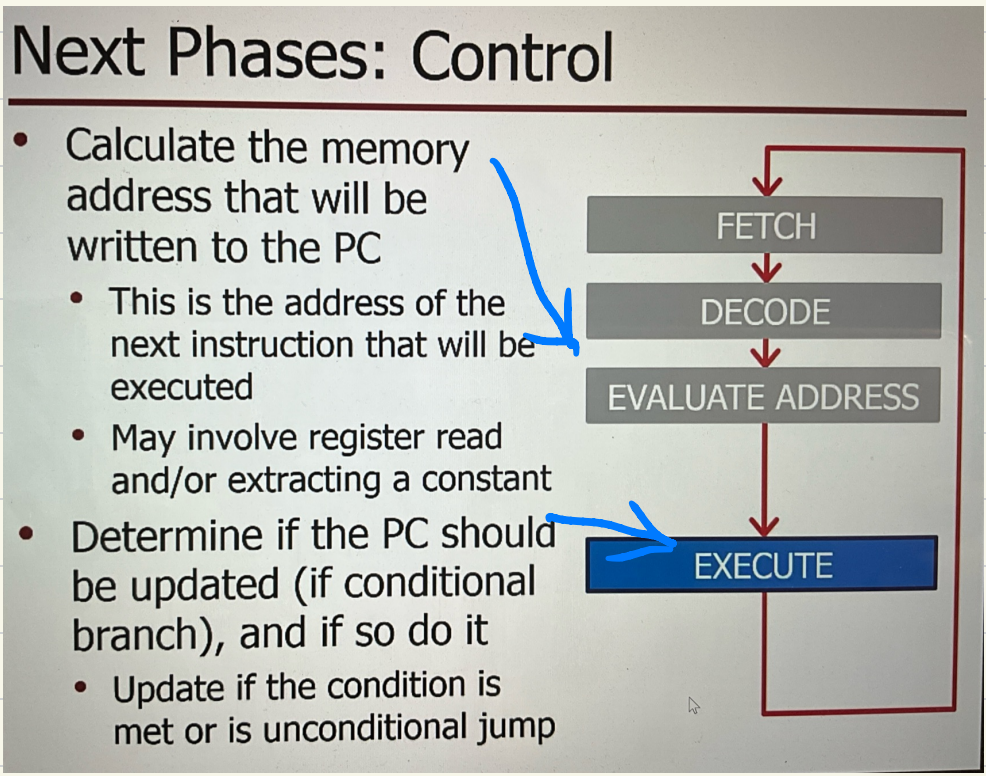


Data Movement (store):



* (storing result phase means to store in "register file", NOT memory)

Control :



☆ (change the pointing of the PC,

it can not follow the order

based on conditions)