

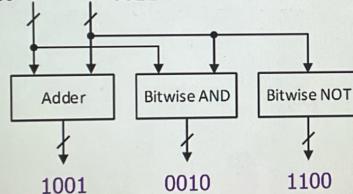


Basic Processor Model

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)

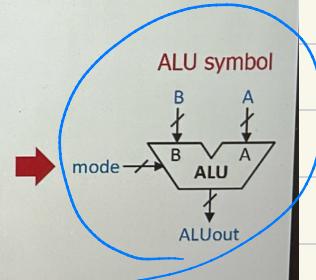
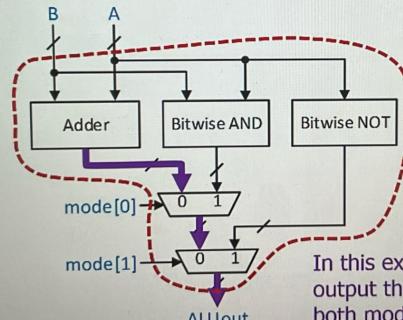
0110 B A 0011



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

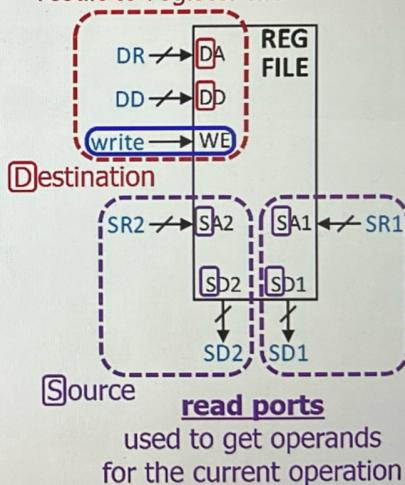


In this example, the ALU will output the sum of its inputs if both mode bits are equal to 0

The Register File

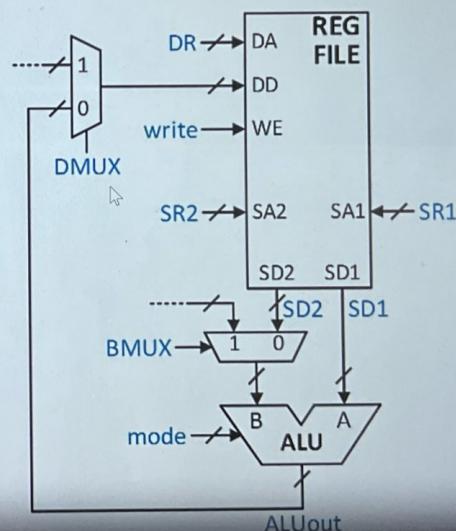
- A **register file** is a small memory located near the ALU
 - Read from multiple locations and write to one location
 - Address inputs control which registers are accessed
 - Write enable controls whether the destination input is stored or not

write port
used to write operation's result to register file

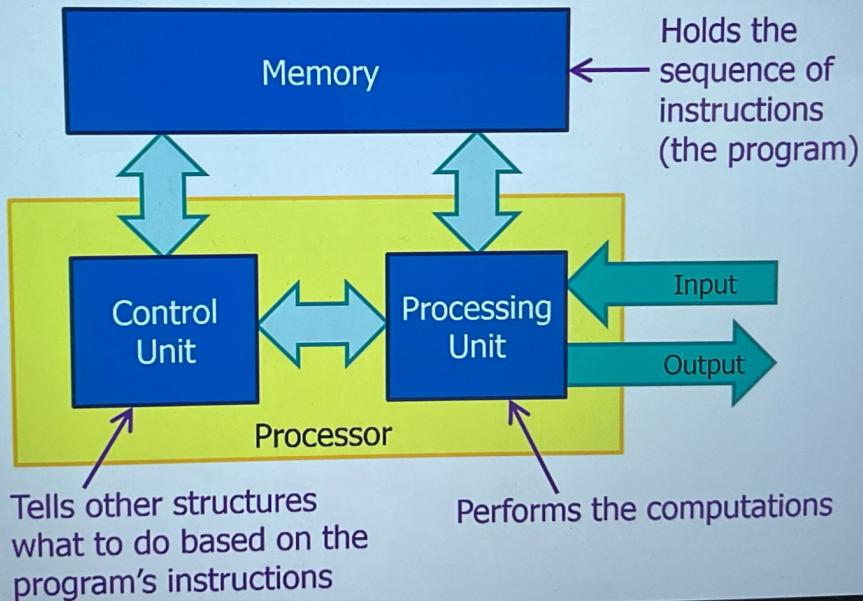


The Processing Unit

- 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

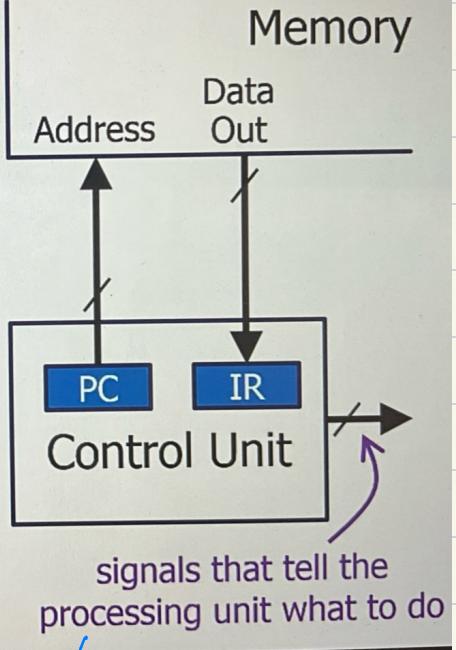


Stored Program Computer

- 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...

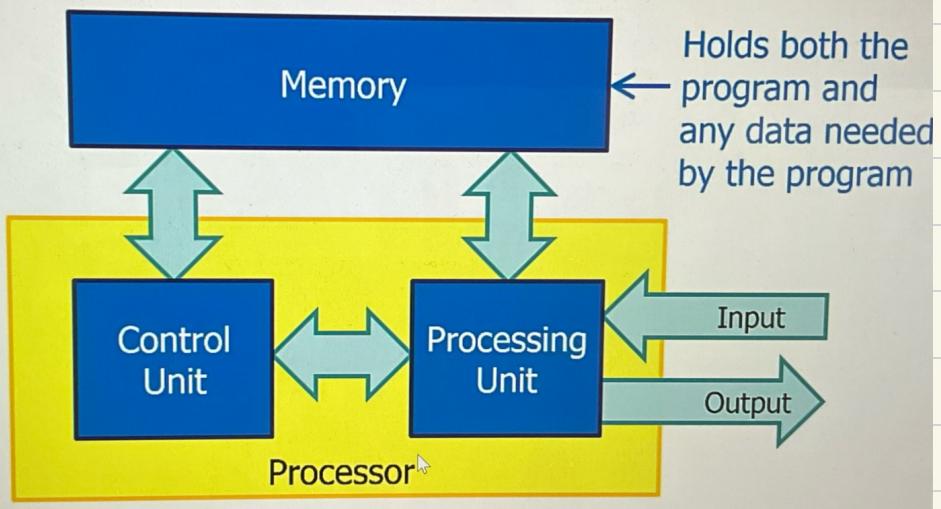
Control Unit

- 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 Registers
file and ALU)

von Neumann Compute Model



Instructions and Instruction Cycle

Instructions

- Each small step of a program is encoded into a binary word called an instruction
 - The instructions that a processor can execute are defined by its Instruction Set Architecture (ISA)
- Instructions are divided up into fields (groups of bits), each providing info about the instruction
 - The fields and their sizes are also dictated by the ISA
 - Opcode: indicates the operation type (ADD, JMP, etc)
 - Other bits indicate operands and other information

* 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 $R3 \leftarrow R4 + R7$

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

Annotations:

- A blue bracket under the OPCODE field is labeled "Add Instruction".
- Blue arrows point from the DESTINATION REGISTER field to the value "3", with the note "Value is 3" and "∴ Destination R3 register".
- Blue arrows point from the SOURCE REGISTER 1 field to the values "4" and "7", with the notes "(value is 4)" and "(value is 7)", respectively, and "∴ R4 register" and "∴ R7 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: $R3 \leftarrow R4 + R7$

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: $R0 \leftarrow R0 + 1$

OPCODE				DEST REG			SRC REG 1				MODE		5-bit CONSTANT				
0	0	0	1	0	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 ALU'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

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

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

put value read from memory here

LDR (a type of memory load)

use these field values to calculate memory address

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:** $PC \leftarrow R6$ *overwrite the PC with the value in register R6*

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

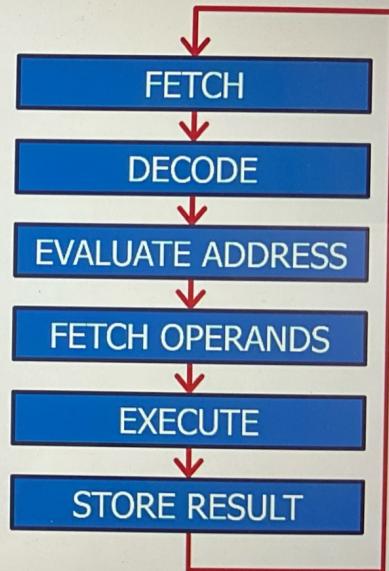
JMP (an unconditional jump)

register that contains the value that will be written into the PC

★ (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**

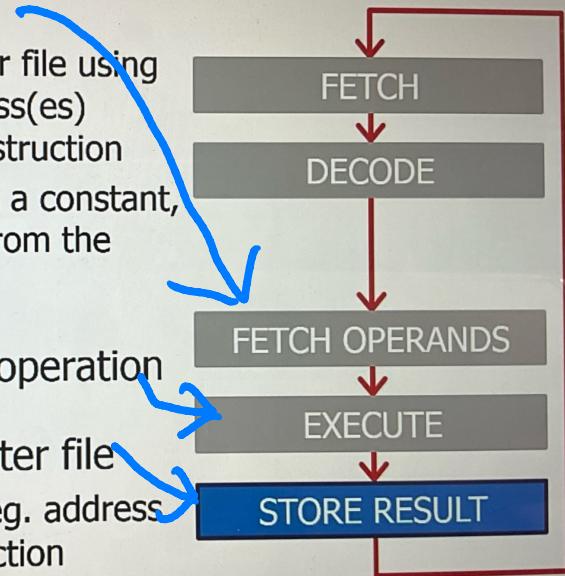


★ Both Fetch / Decode are needed in any operation type

For Operate:

Next Phases: Operate

- Get the operands
 - Read from register file using source reg. address(es) encoded in the instruction
 - If instruction uses a constant, extract its value from the instruction
- Perform the ALU operation
- Write to the register file
 - Use destination reg. address encoded in instruction
 - Use data produced by ALU



1. Get operands from source addresses

2. Perform the operation

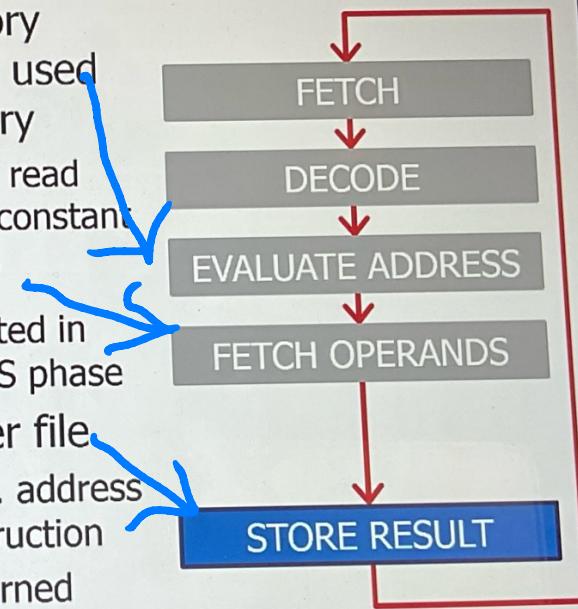
3. Put / write the result back in register

file

For Data Movement : (Load)

Next Phases: Data Move (Load)

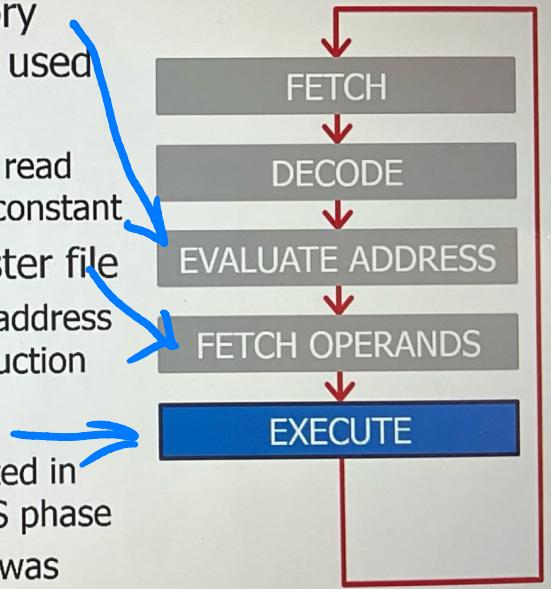
- Calculate the memory address that will be used to read from memory
 - May involve register read and/or extracting a constant
- Read from memory
 - Use address calculated in EVALUATE ADDRESS phase
- Write to the register file
 - Use destination reg. address encoded in the instruction
 - Use data value returned from memory



Data Movement (Store) :

Next Phases: Data Move (Store)

- Calculate the memory address that will be used to write to memory
 - May involve register read and/or extracting a constant
- Read from the register file
 - Use source register address encoded in the instruction
- Write to memory
 - Use address calculated in EVALUATE ADDRESS phase
 - Use data value that was read from the register file

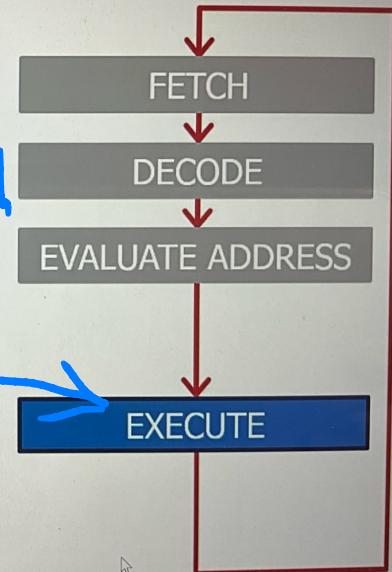


R (storing Result phase means to store in "register file", NOT memory)

Control :

Next Phases: Control

- Calculate the memory address that will be written to the PC
 - This is the address of the next instruction that will be executed
 - May involve register read and/or extracting a constant
- Determine if the PC should be updated (if conditional branch), and if so do it
 - Update if the condition is met or is unconditional jump



* (change the printing of the PC,
it can not follow the order
based on conditions)