Computer Architecture Assembly Language

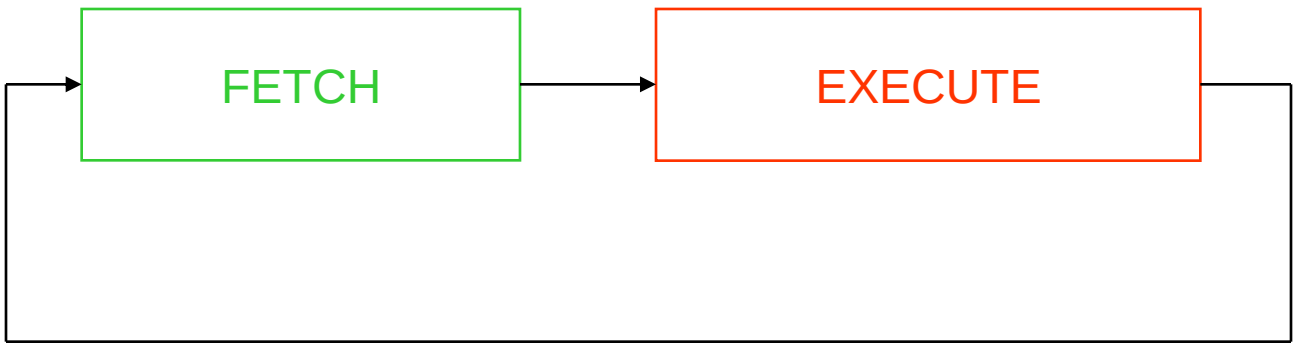
Computer executes a PROGRAM stored in MEMORY.

Basic scheme is - DO FOREVER:

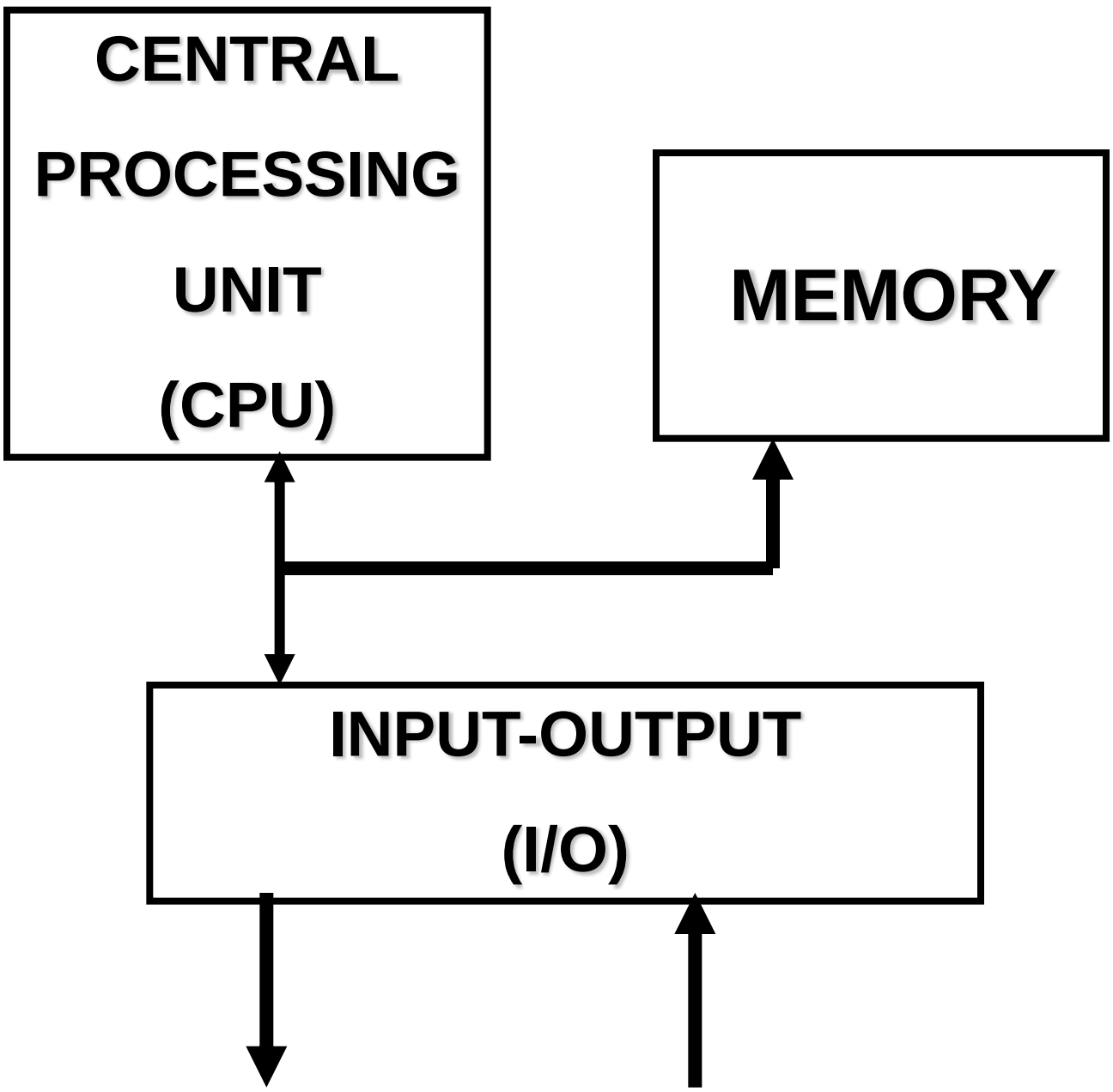
1. FETCH an instruction (from memory).
2. EXECUTE the instruction.

This is the FETCH-EXECUTE cycle.

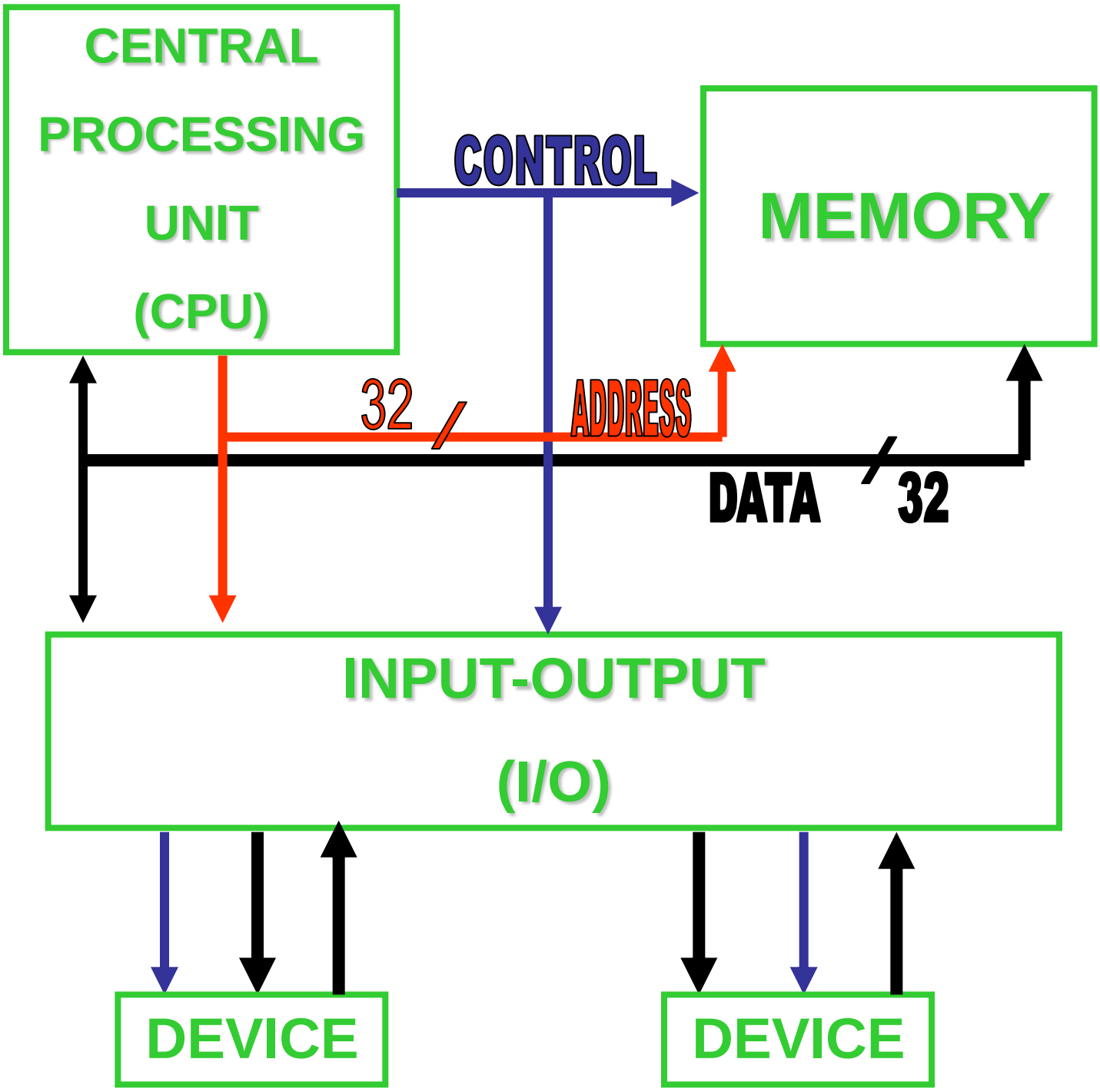
More complicated in REAL machines (e.g. interrupts).



Block Diagram of a Computer



Refined Block Diagram



Data path

Contains: registers, program counter, ALU, and address/data interconnections or BUSes.

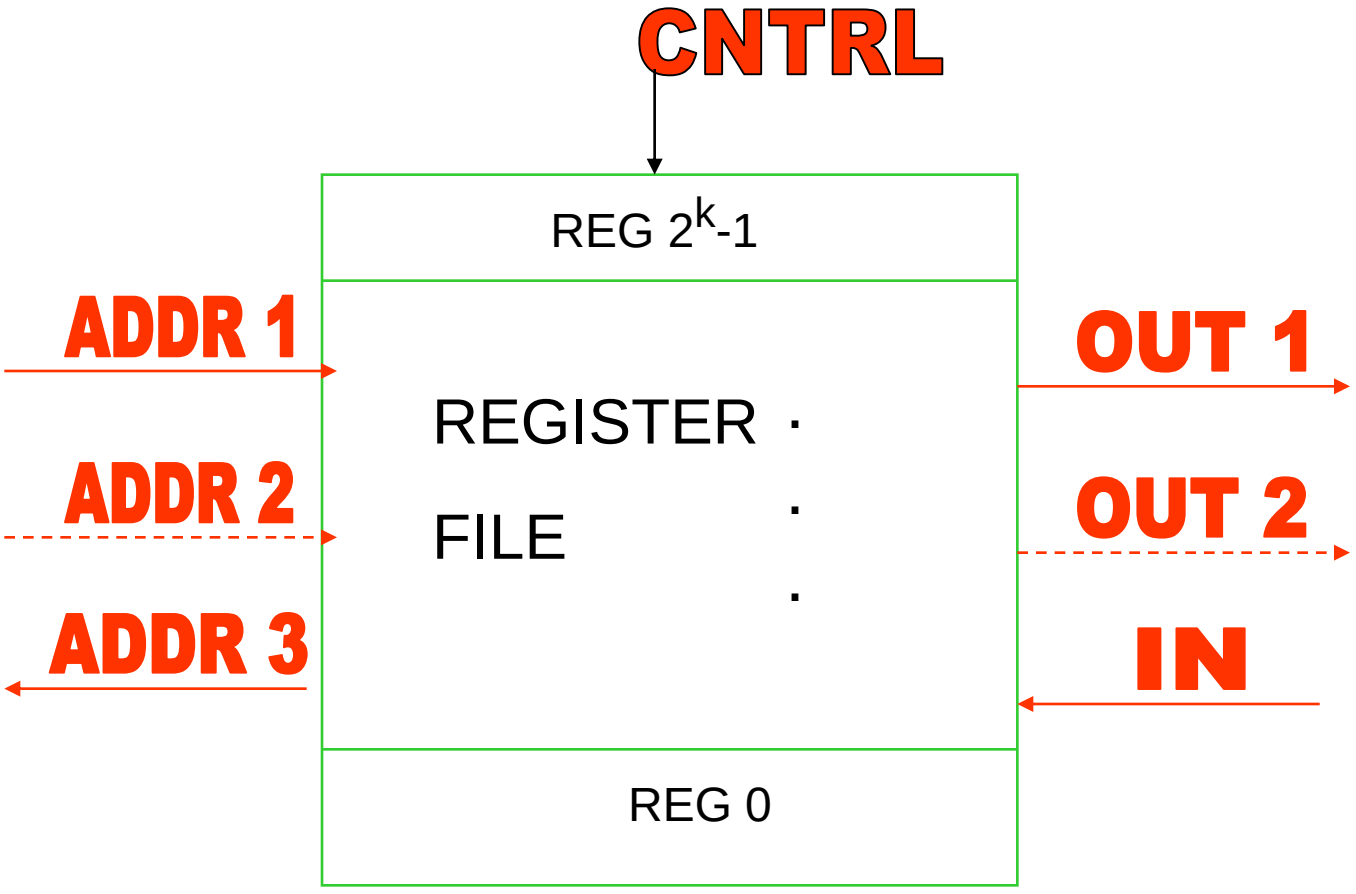
Registers (Accumulators)

Basic operations: WRITE and READ.

Important properties: WIDTH (in bits) and ACCESS TIME.

Sometimes - other operations possible (e.g. shift, compare, increment, mask).

Most CPUs have 1 to 32 registers.



Flags

Each FLAG represents a BIT of important information:

MACHINE STATUS (error, interrupt, mode)

COMPUTATION STATUS (carry, overflow, zero, sign)

Usually also ``packed'' into a special ``register''

Arithmetic Logic Unit (ALU)

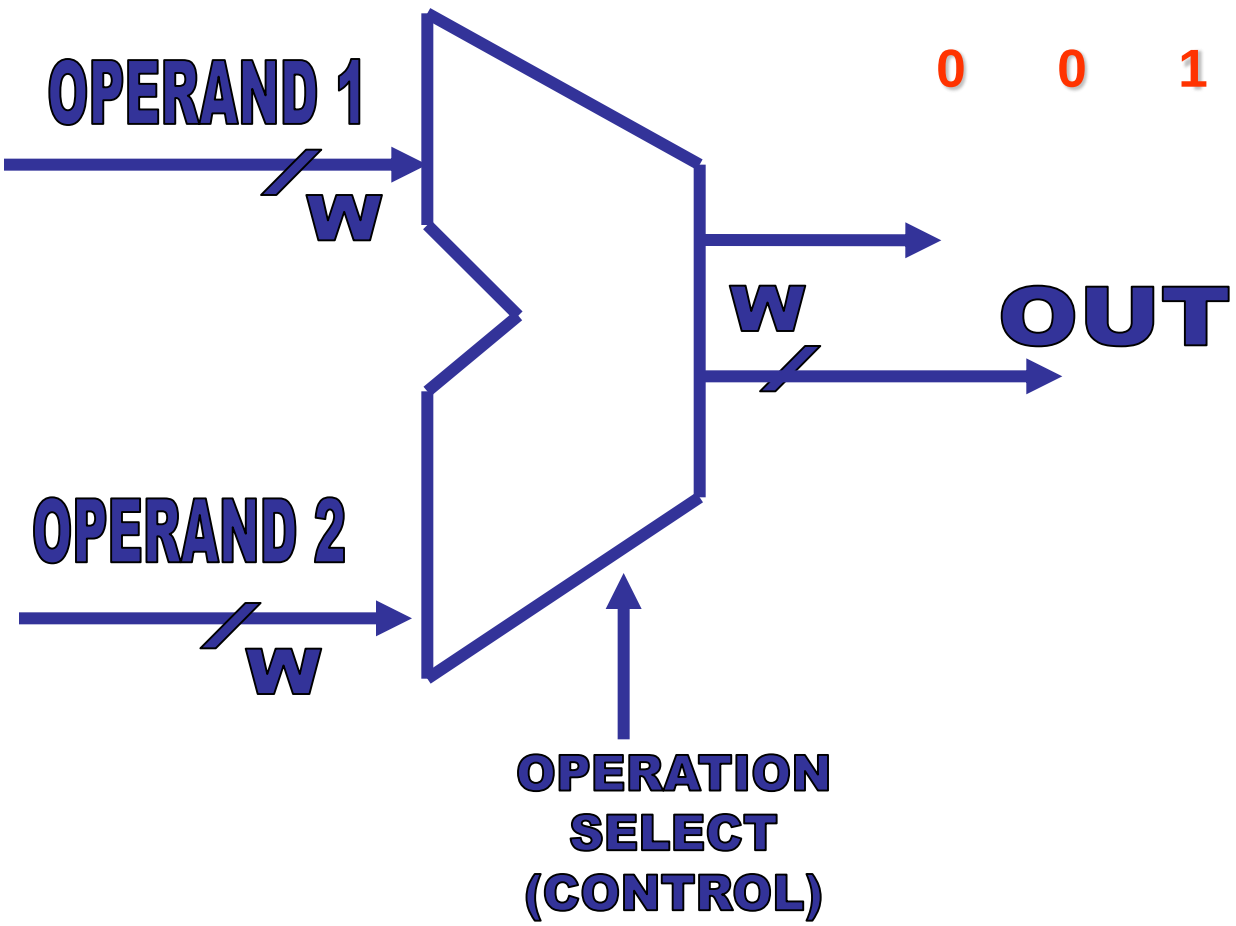
Performs actual computations:

Arithmetic (add, subtract, multiply, negate)

Logical (bitwise or, and, invert)

Example: bitwise and

1	0	1	0
0	1	1	0
0	0	1	0

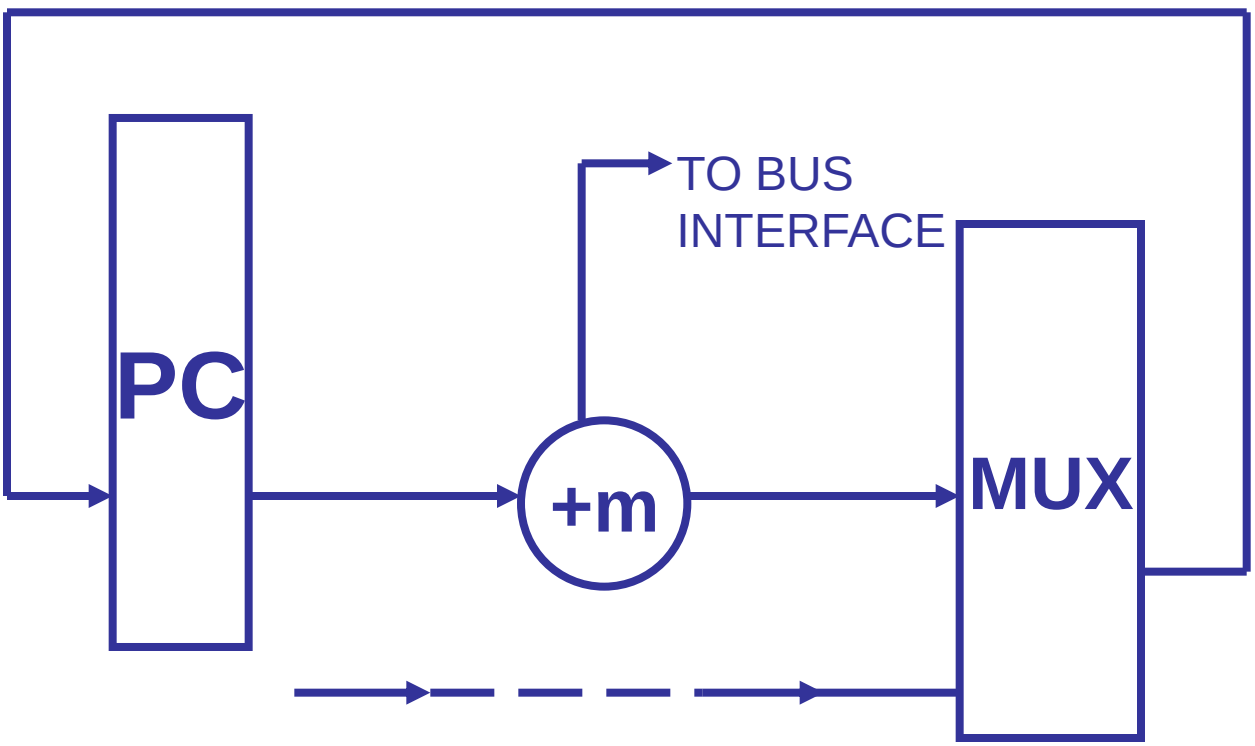


Instruction Sequencing

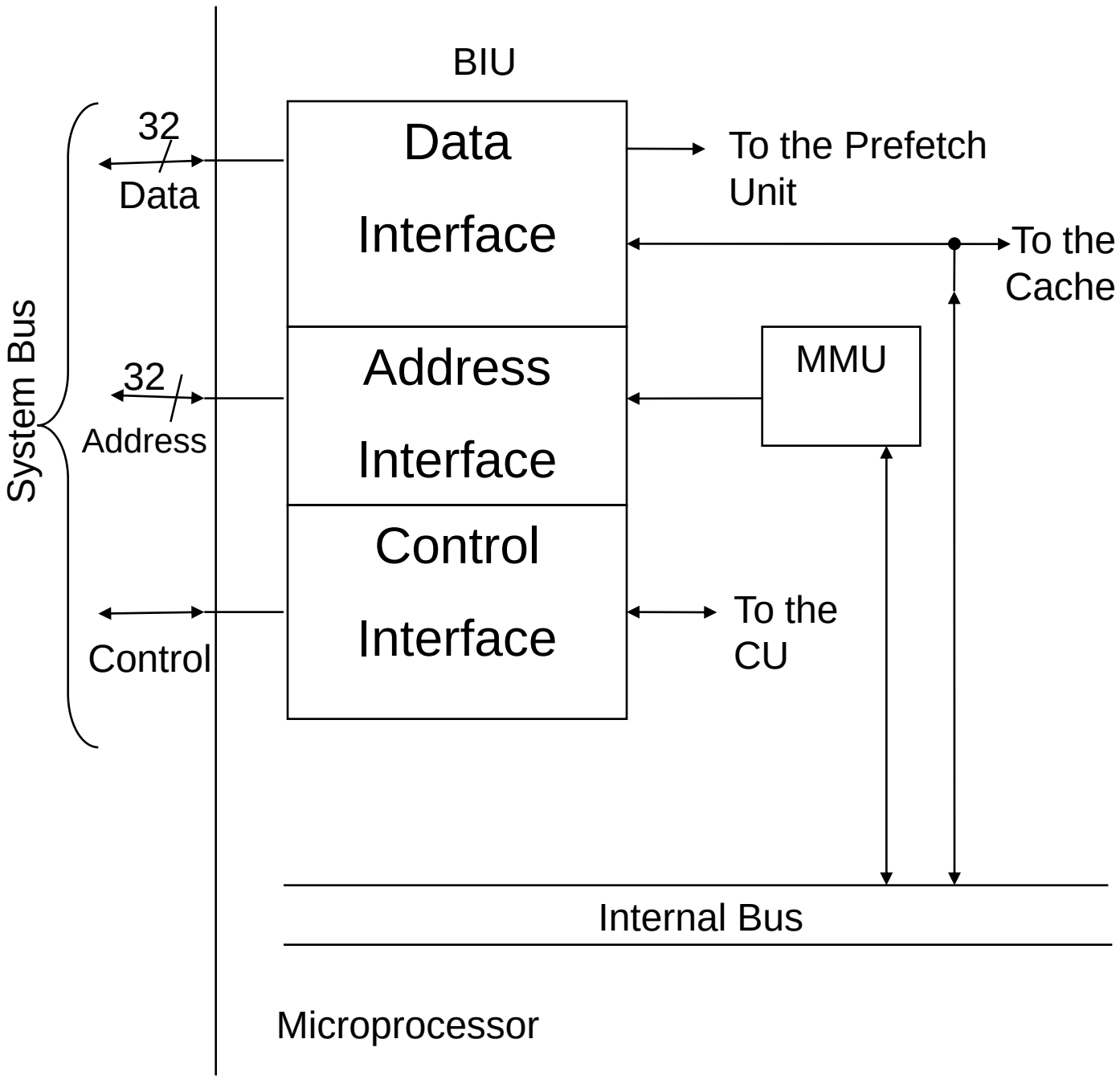
Instructions usually fetched from consecutive memory locations.

Use ``incrementer'' to advance PC

Except for JUMP, CALL, or INTERRUPT.



Bus Interface



Control

Generates control/timing signals

Selects OPERATIONS performed in:

- ALU - select function
- Register file - which to read, where to write
- Program counter - advance or jump?
- Bus control: memory address from where? Read or write?
- Interrupts

Performance

Timing is based on a CLOCK CYCLE or FREQUENCY (e.g. 4GHz).

Every action takes 1 or more clock cycle.

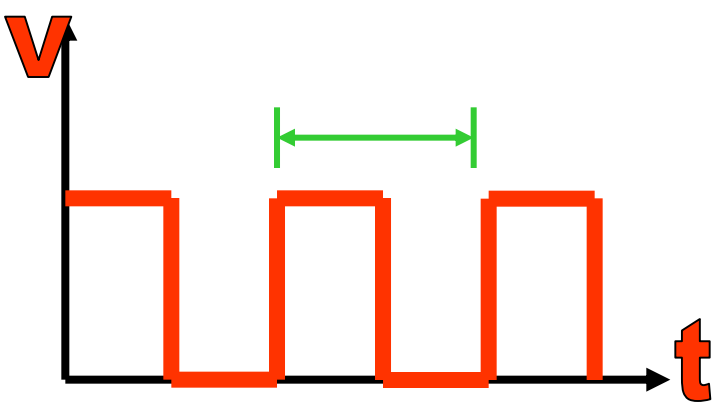
Main memory access usually more than 1 cycle - memory ACCESS TIME is critical!

EXECUTION contains several steps - may require MEMORY ACCESS.

Performance depends heavily on:

- How many instructions to execute program or function?
- How many cycles per instruction?

Thus, PERFORMANCE ENHANCEMENTS: instruction prefetch, cache, pipelining, etc.



Programming in Assembly Language

ASSEMBLY LANGUAGE is (almost) 1 to 1 with MACHINE CODE

Assembly language constructs are:

- Symbolic version of machine instructions
- Labels (standing for constants and memory addresses)
- Pseudo-operations

ASSEMBLER converts program to object file in 2 passes:

- Pass I: translate symbolic instructions into binary code, create SYMBOL TABLE of labels.
- Pass II: translate labels into (relocatable) addresses, fix binary code, and create object file with relocation information.