EE 213 Computer Organization and Assembly Language

Week # 2, Lecture # 6

27th Dhu'l-Hijjah, 1439 A.H

7th September 2018

Minds open...

... Laptops closed

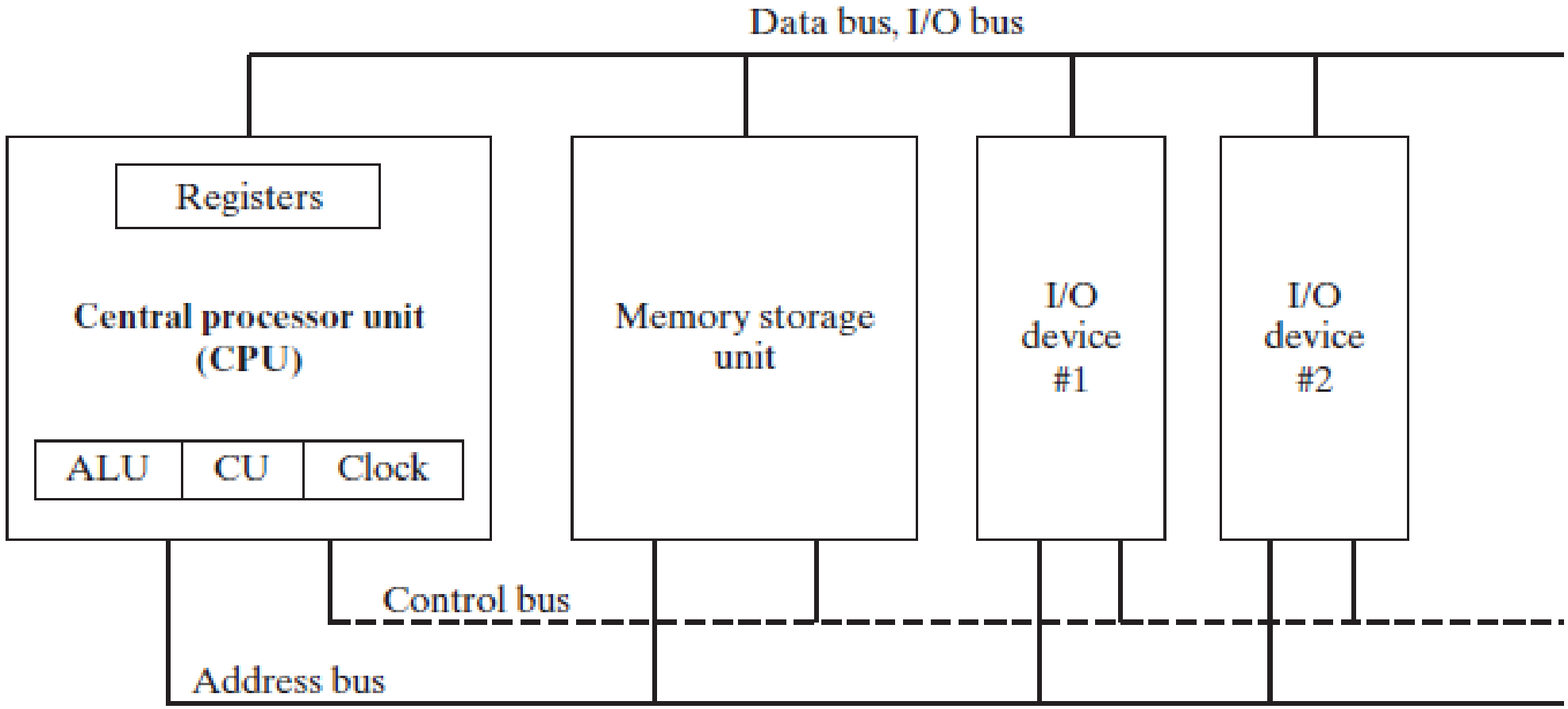
This presentation helps in delivering the lecture.
Take notes, interact and read text book to learn and gain knowledge.

Today's Topics

- Basic Diagram of a micro-computer
- Instruction Cycle
- Simplified CPU block diagram
- Abstracted Micro-architecture of Core i7 (2013)
- Reading and Writing Memory
- Reading and Writing Input Output Devices
- Mode of Operations of x86 processors
- Address Space of x86 processors

Please read chapter # 2 (2.1 and 2.2) over the weekend.

FIGURE 2–1 Block diagram of a microcomputer.

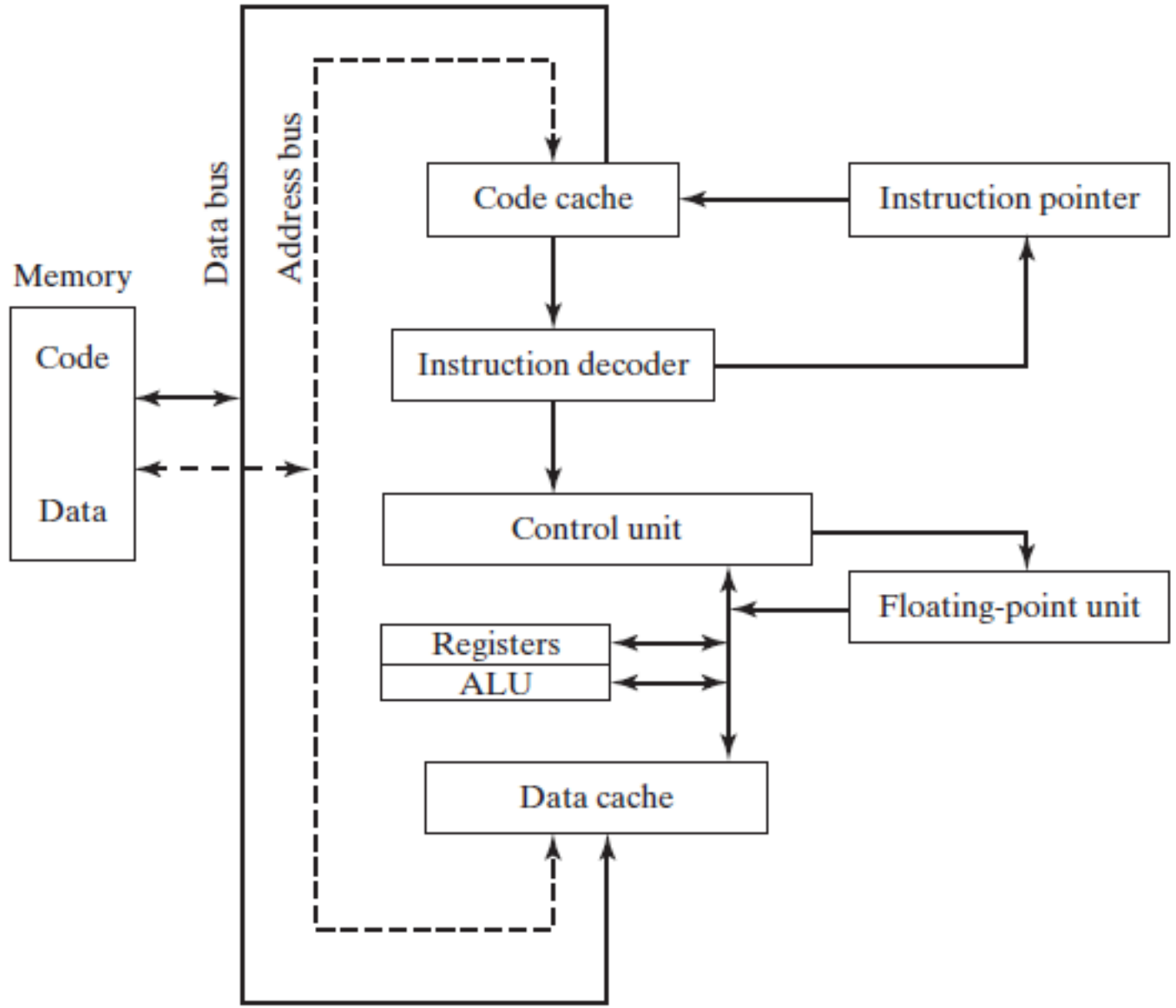


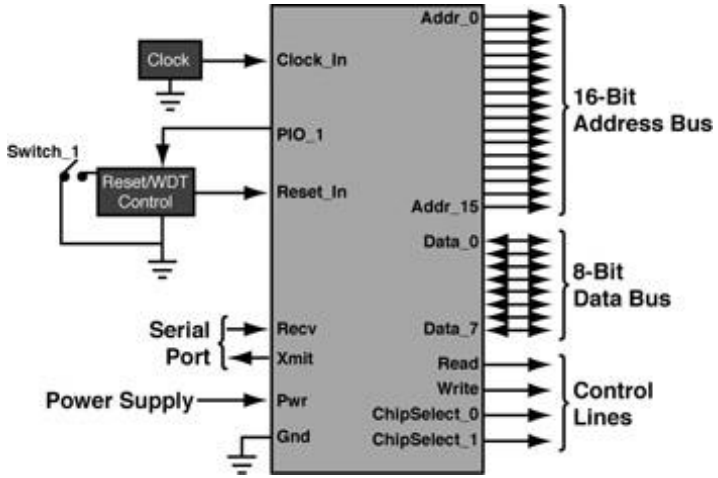
2.1.2 Instruction Execution Cycle

A single machine instruction does not just magically execute all at once. The CPU has to go through a predefined sequence of steps to execute a machine instruction, called the *instruction execution cycle*. Let's assume that the instruction pointer register holds the address of the instruction we want to execute. Here are the steps to execute it:

1. First, the CPU has to **fetch the instruction** from an area of memory called the *instruction queue*. Right after doing this, it increments the instruction pointer.
2. Next, the CPU **decodes** the instruction by looking at its binary bit pattern. This bit pattern might reveal that the instruction has operands (input values).
3. If operands are involved, the CPU **fetches the operands** from registers and memory. Sometimes, this involves address calculations.
4. Next, the CPU **executes** the instruction, using any operand values it fetched during the earlier step. It also updates a few status flags, such as Zero, Carry, and Overflow.
5. Finally, if an output operand was part of the instruction, the CPU **stores the result** of its execution in the operand.

FIGURE 2–2 Simplified CPU block diagram.



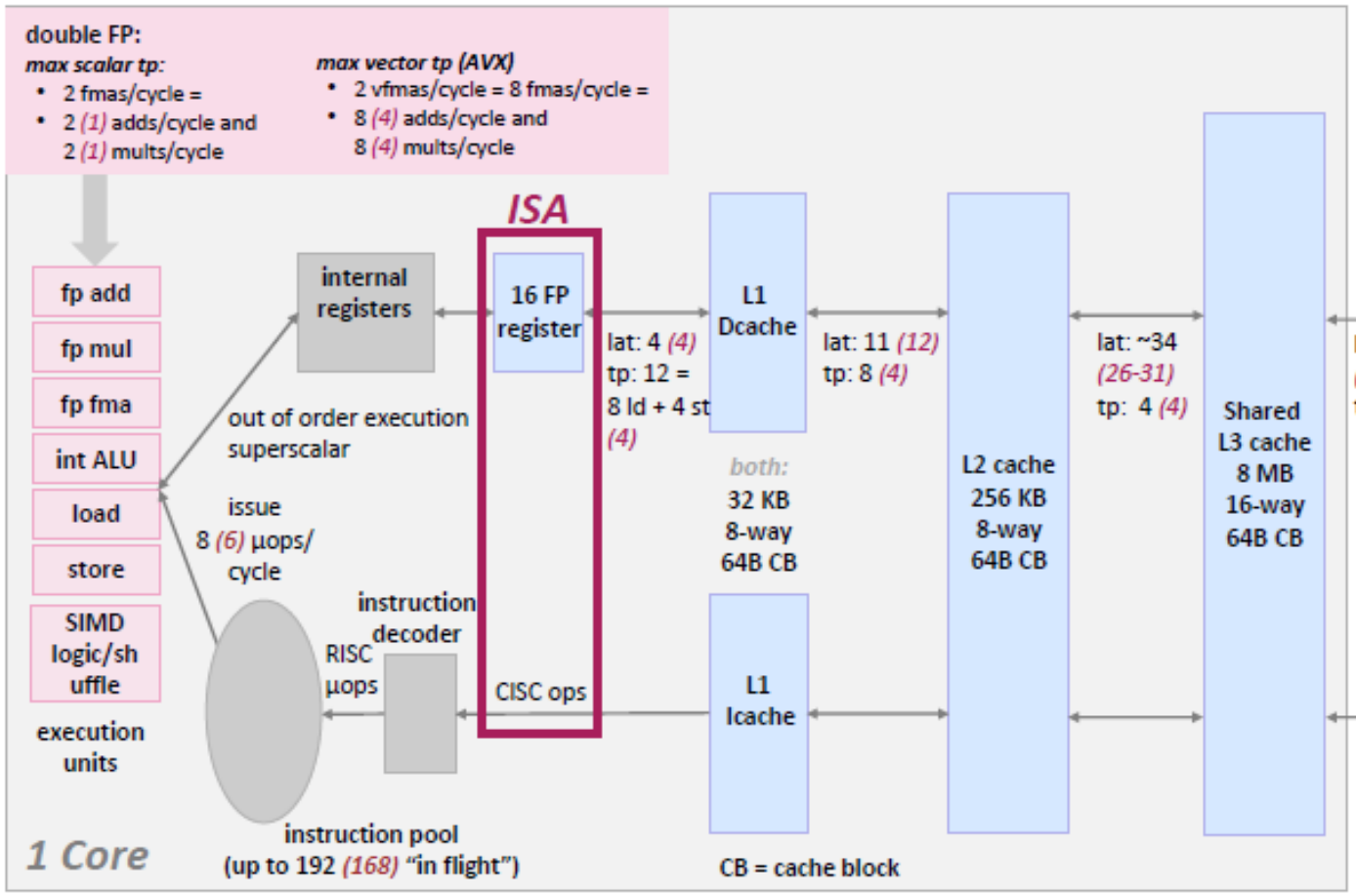


Abstracted Microarchitecture: Example Core i7 Haswell (2013) and *Sandybridge (2011)*

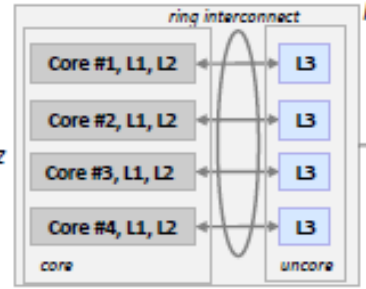
Throughput (tp) is measured in doubles/cycle. For example: 4 *(2)*
Latency (lat) is measured in cycles
1 double floating point (FP) = 8 bytes
fma = fused multiply-add
Rectangles not to scale

Haswell | *Sandy Bridge*

Memory hierarchy:
- Registers
- L1 cache
- L2 cache
- L3 cache
- Main memory
- Hard disk



Core i7-4770 Haswell:
4 cores, 8 threads
3.4 GHz
(3.9 GHz max turbo freq)
2 DDR3 channels 1600 MHz



processor die

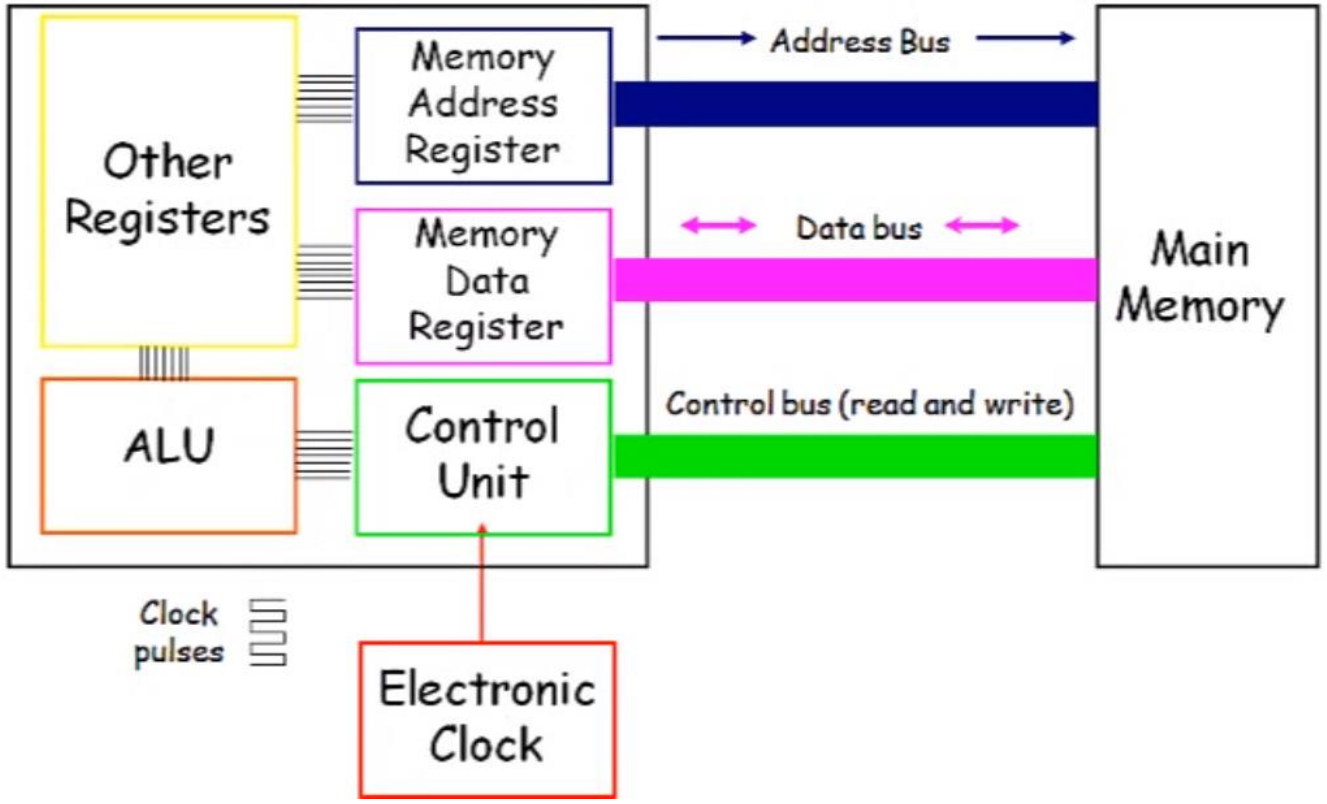
RAM

Source: Intel manual (chapter 2), http://www.7-cpu.com/cpu/Haswell.html

2.1.3 Reading from Memory

As a rule, computers read memory much more slowly than they access internal registers. This is because reading a single value from memory involves four separate steps:

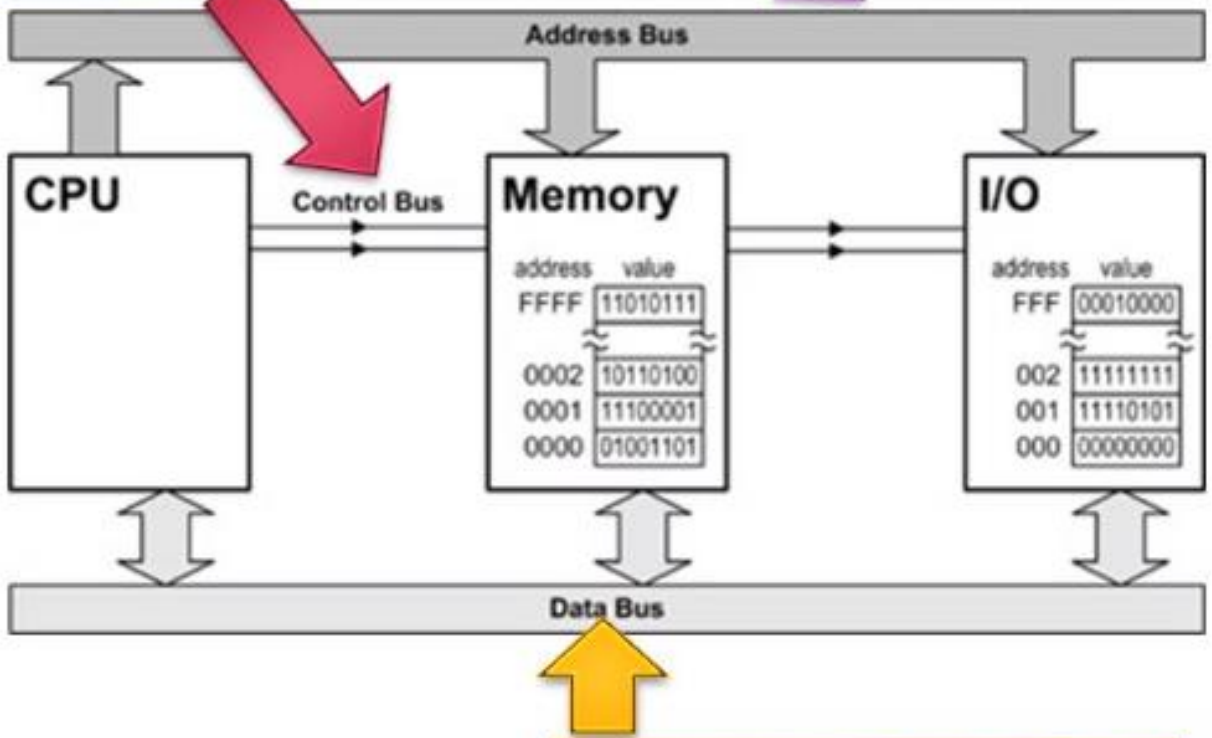
1. Place the address of the value you want to read on the address bus.
2. Assert (change the value of) the processor's RD *(read)* pin.
3. Wait one clock cycle for the memory chips to respond.
4. Copy the data from the data bus into the destination operand.



I/O PRINCIPLE

broadcast the location it wishes to read from (or write to).

trigger a read or write operation, and select either memory or I/O.



Data is sent to/from the CPU.

Basic Program Execution Registers

Registers are high-speed storage locations directly inside the CPU, designed to be accessed at much higher speed than conventional memory. When a processing loop is optimized for speed, for example, loop counters are held in registers rather than variables. Figure 2-3 shows the *basic program execution registers*. There are eight general-purpose registers, six segment registers, a processor status flags register (EFLAGS), and an instruction pointer (EIP).

Instruction Pointer The EIP, or *instruction pointer*, register contains the address of the next instruction to be executed. Certain machine instructions manipulate EIP, causing the program to branch to a new location.

EFLAGS Register The EFLAGS (or just *Flags*) register consists of individual binary bits that control the operation of the CPU or reflect the outcome of some CPU operation. Some instructions test and manipulate individual processor flags.

A flag is *set* when it equals 1; it is *clear* (or reset) when it equals 0.

Status Flags The status flags reflect the outcomes of arithmetic and logical operations performed by the CPU. They are the Overflow, Sign, Zero, Auxiliary Carry, Parity, and Carry flags. Their abbreviations are shown immediately after their names:

- The **Carry** flag (CF) is set when the result of an *unsigned* arithmetic operation is too large to fit into the destination.
- The **Overflow** flag (OF) is set when the result of a *signed* arithmetic operation is too large or too small to fit into the destination.
- The **Sign** flag (SF) is set when the result of an arithmetic or logical operation generates a negative result.
- The **Zero** flag (ZF) is set when the result of an arithmetic or logical operation generates a result of zero.
- The **Auxiliary Carry** flag (AC) is set when an arithmetic operation causes a carry from bit 3 to bit 4 in an 8-bit operand.
- The **Parity** flag (PF) is set if the least-significant byte in the result contains an even number of 1 bits. Otherwise, PF is clear. In general, it is used for error checking when there is a possibility that data might be altered or corrupted.