weeks 1000 - 1001
CPU

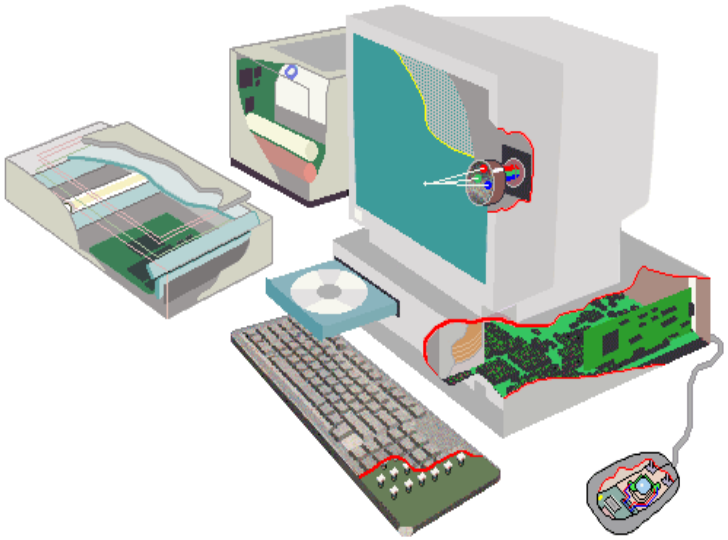
CCS1310

Computer Systems Architecture

Kostas Dimopoulos

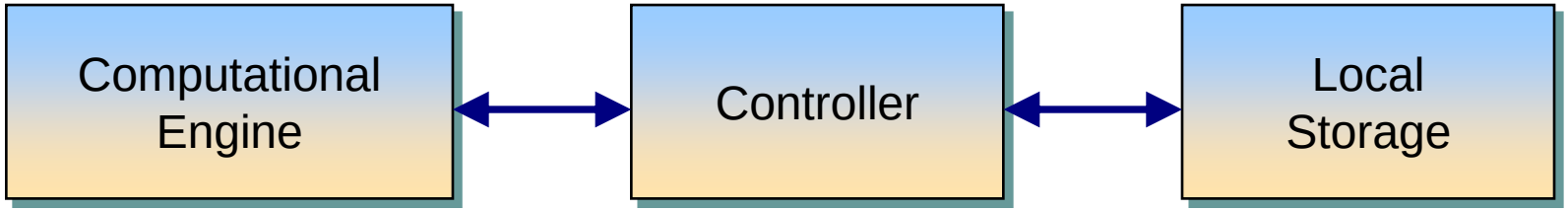
Lecture Outline

- What's inside the CPU?
- What are the key components?
- What are main characteristics of each key component?
- What is the fetch/execute cycle?
- What are instruction sets?



Structure of a Processor

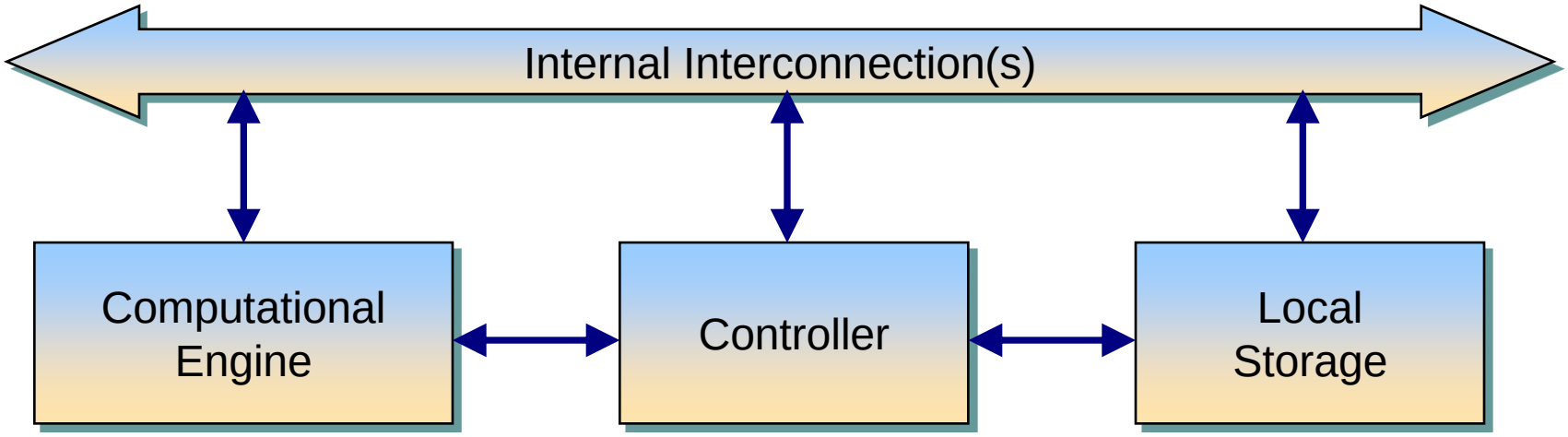
- Overall responsibility for execution
- Moves through sequence of steps
- Coordinates other units
- Timing-based operation: knows how long each unit requires and schedules steps accordingly



- Operates as directed by controller
- Performs all computational tasks
  - Arithmetic and operations
  - Logical operations
  - Operations on bits
- Performs one operation at a time as directed

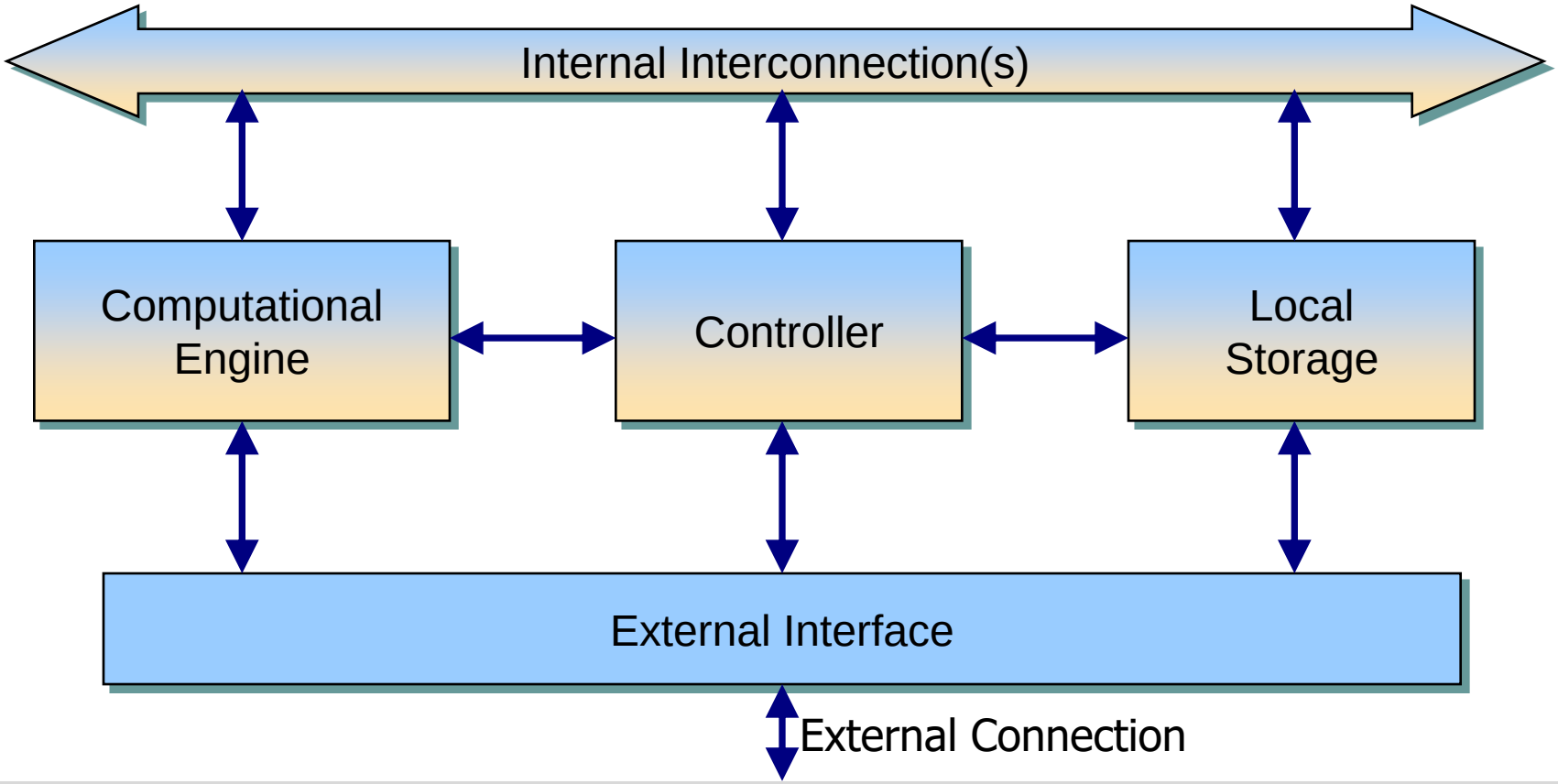
- Temporary
- Holds data values for operations
- Values must be inserted (e.g., loaded from memory) before the operation can be performed

Structure of a Processor



- Allow transfer of values among units of the processor
- Also called **data paths**

Structure of a Processor



- Handles communication between processor and rest of computer system
- Provides interaction with external memory as well as external I / O devices

Processor Categories

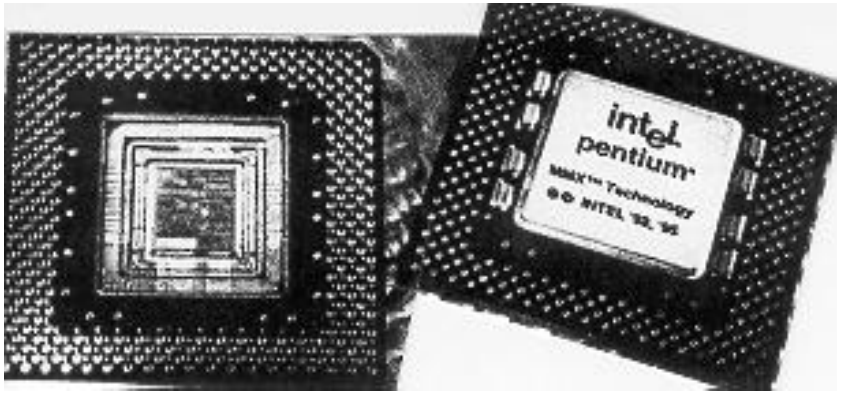
- Processors can be used by hardware devices in a variety of roles:
  - Coprocessors
    - Operate in conjunction and under the control of another processor. A special-purpose processor that performs a single task at high speed (such as I/O interfacing or encryption, string processing, floating-point arithmetic and signal processing.)
    - E.g. FPU (Floating Point Unit)., GPU etc.
  - Microcontrollers (MCUs)
    - Programmable devices dedicated to the control of a physical system (external hardware). Operations do not require computation. They have tiny memory and slow processing capabilities, but require very limited power to function
    - E.g. automatic doors, airplane landing gear, implantable medical devices, remote controls, toys etc.

Processor Categories (cont'd)

- Embedded system processors
  - Run sophisticated electronic devices. More powerful than microcontrollers but less powerful than general purpose processors.
  - E.g. control DVD player, including commands received from a remote control as well as from the front panel.
- Microsequencers
  - Control coprocessors and other engines **within a larger processor**. Does not perform any operation itself. It generates addresses used by the microprogram in the control unit. It is used as a part of the control unit of a CPU or as a stand-alone generator for address ranges.
  - E.g. ask data movement unit to move two values in FPU, ask FPU to perform addition, ask data movement unit to place result in memory.
- General-purpose processors
  - The CPU in a PC

Central Processing Unit

- The heart (and brain) of any computer
- Carries out the program's instructions
  - Operates on data in the computer's memory
- Most CPUs are programmable, general-purpose microprocessors
  - All circuits stored on a single **Integrated Circuit**



The first microprocessor was the Intel 4004 (in 1971). It could only add and subtract 4 bits at a time.

The first microprocessor into a home computer was the Intel 8080 (1974) and could manipulate 8 bits.

The first microprocessor that made a real hit the market was the Intel 8088 (1979) which was incorporated in the IBM PC.

Advances of Intel Processors

Name	Date	Transistors	Microns	Clock Speed	Data Width	MIPS
8080	1974	6,000	6	2 MHz	8 bits	0.64
8088	1979	29,000	3	5 MHz	16 bits, 8 bit bus	0.33
80286	1982	134,000	1.5	6 MHz	16 bits	1
80386	1985	275,000	1.5	16 MHz	32 bits	5
80486	1989	1,200,000	1	25 MHz	32 bits	20
Pentium	1993	3,100,000	0.8	60 MHz	32 bits, 64 bits bus	100
Pentium II	1997	7,500,000	0.35	233 MHz	32 bits , 64 bits bus	~300
Pentium III	1999	9,500,000	0.25	450 MHz	32 bits , 64 bits bus	~510
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits , 64 bits bus	~1,700
Pentium 4 Pr	2004	125,000,000	0.09	3.6 GHz	32 bits , 64 bits bus	~7,000

- **Transistors:** number of transistors on the chip. Number increases.
- **Microns:** is the width, in microns, of the smallest wire on the chip (human hair is 50 microns thick. As the feature size on the chip goes down, the number of transistors rises.
- **Clock speed:** is the maximum rate that the chip can be clocked.
- **Data Width:** width of the ALU. An 8- bit ALU can add/ subtract/ multiply/ etc. two 8-bit numbers, while a 32-bit ALU can manipulate 32-bit numbers. An 8-bit ALU would have to execute 4 instructions to add two 32-bit numbers, while a 32-bit ALU can do it in one instruction.

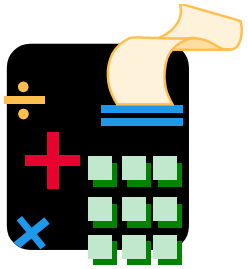
Main Components of the CPU

Control Unit

- Supervises the operation of entire computer system
- Interprets and controls execution of instructions

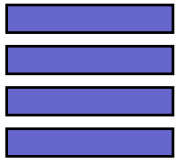
Arithmetic/Logic Unit

- Provides logical and computational capabilities
- Data is held temporarily and manipulated
- Calculations are made



Registers

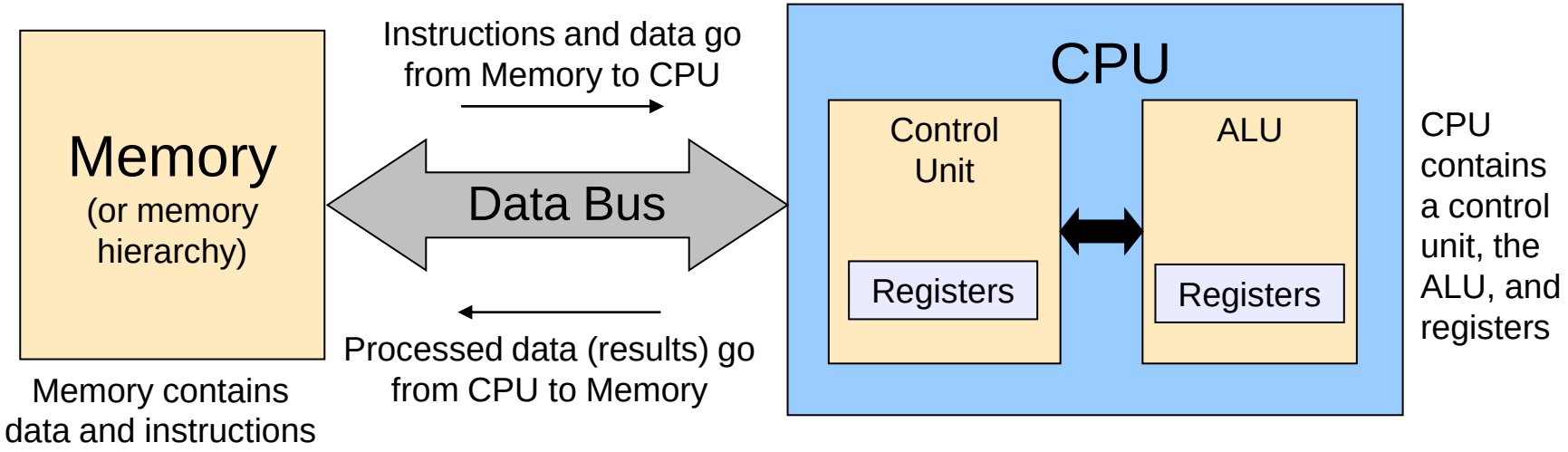
- Temporary storage area for data



Clock

- Triggers and synchronizes hardware operations

CPU–Memory High Level Layout



- Programs and data are stored in memory
- CPU executes program instructions and processes data
- The CPU performs the following:
  - Fetches instructions and data from memory
  - Decodes and executes instructions
  - Stores results back into memory (to be displayed on the screen or stored onto disk)
  - Goes to the next instruction (if there is one)

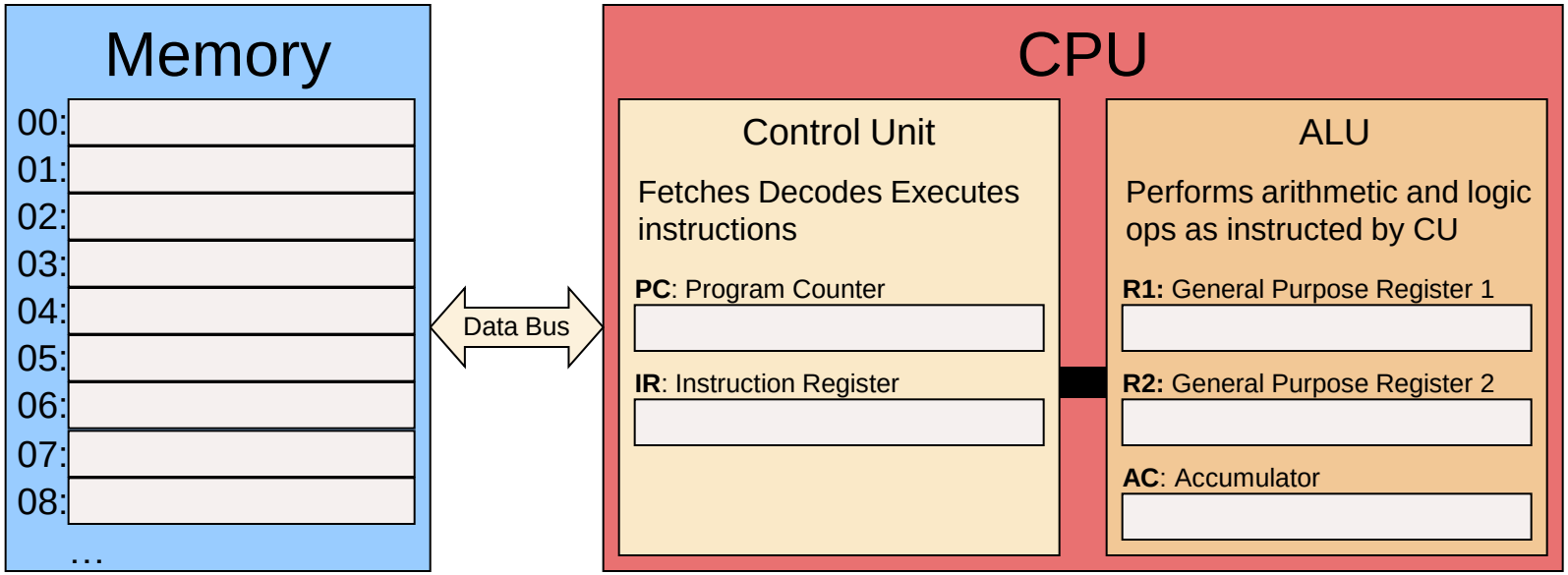
Registers

- A register is a single, permanent location within the CPU used for a particular defined purpose
- It holds a binary value temporarily for storage, for manipulation, and/or for simple calculations
- **How are registers used in a computer?** They hold:
  - Data being processed
  - An instruction being executed
  - A memory or I/O address to be accessed
  - Special binary codes used for other purposes
- **What operations can be performed on registers?**
  - Can be loaded with values from other locations (other registers or memory locations) ---the previous value is destroyed
  - Data from another location can be added to or subtracted from the value previously stored in the register, leaving sum or difference
  - Data can be rotated right or left by one or more bits

Registers of the CPU

- Control Unit:
  - Program Counter (PC): holds the address of the current/next instruction being executed
  - Instruction Register (IR): holds the actual instruction being executed
  - Memory Address Register (MAR): holds address of a memory location to be opened for data
  - Memory Data Register (MDR): holds a data value that will be stored to or retrieved from memory location addressed by MAR
    - MDR is also called MBR (B stands for buffer)
- Arithmetic Logic Unit:
  - Accumulator: holds data used for arithmetic operations

CPU: a closer look



Instruction cycle: execution of a single program instruction by the CPU

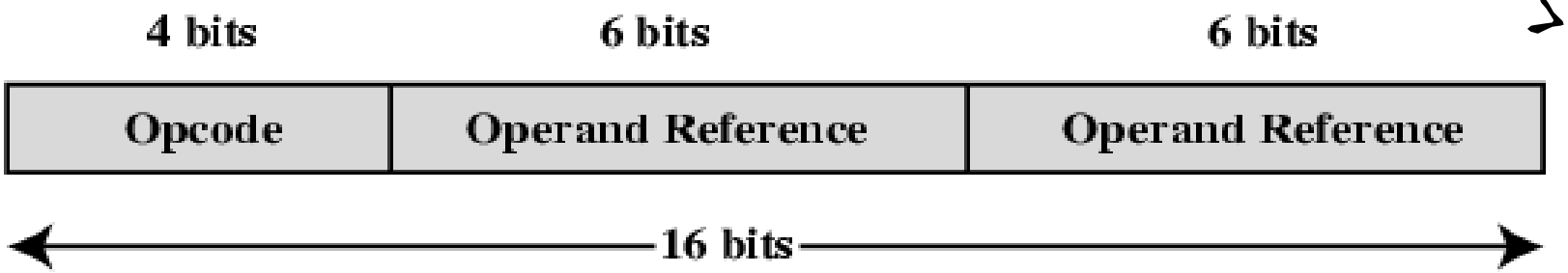
- **Fetch**: control unit loads instruction from memory location defined in **PC** into **IR**
- **Decode**: control unit interprets instruction
- **Execute**: control unit executes instruction (tells ALU what to do)
- **Program Counter** is **incremented** to next program instruction (cycle repeats)

Instruction Set

- CPUs support a set of very simple instructions that typically fall into the following categories:
  - Data Movement (e.g. load, store, copy etc.)
  - Arithmetic (e.g. add, subtract, etc.)
  - Logic (e.g. less than, equal to, etc.)
  - Program Control (e.g. jump, halt, etc.)
  - Coprocessor instructions
- A program consists of a sequence of instructions
- Each instruction specifies:
  - the operation to perform
  - the data to use for the operation (if necessary)
- Instructions are stored and processed in machine language
  - this language consists solely of bit patterns

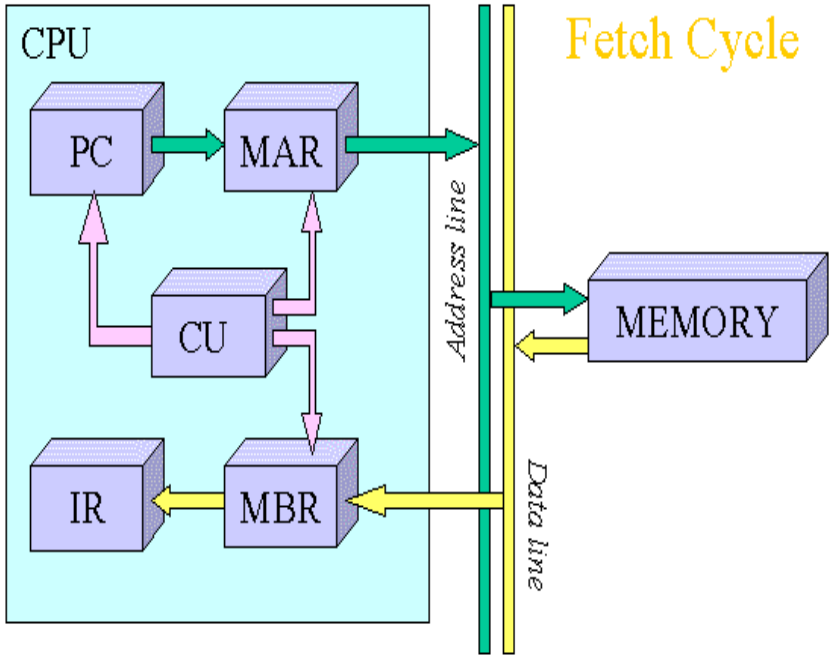
Instruction Representation

- Each instruction has a unique bit pattern
- For human understanding (well, programmers anyway) a symbolic representation is used
  - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
  - ADD A, B



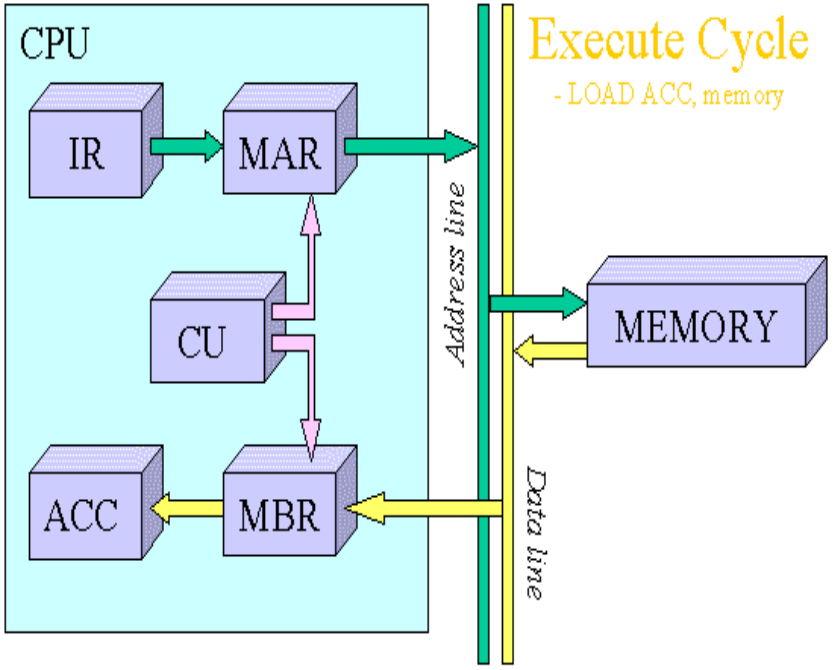
CPU Fetch Cycle

- The address stored in the program counter (PC) is transferred to the memory address register
- CPU transfers the instruction located at the address stored in the memory address register (MAR) to the memory buffer register (MBR)
- Transfer from memory to CPU is coordinated by the control unit (CU)
- The newly fetched instruction is transferred to the instruction register (IR)
- Unless told otherwise, CU increments PC to point to the next address location in memory



CPU Execute Cycle
(LOAD ACC Memory)

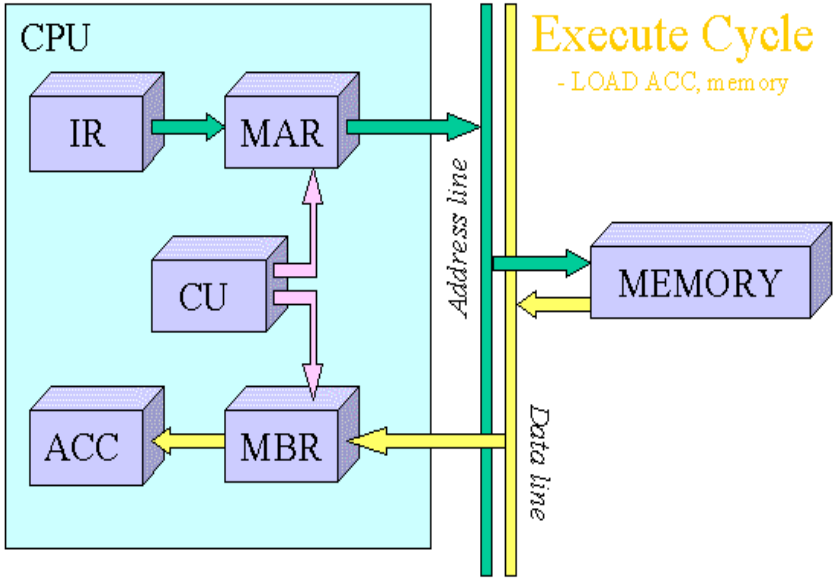
- The operation loads the accumulator (ACC) with data that is stored in the memory location specified in the instruction
- The address portion of the instruction is transferred from the IR to the MAR
- The CPU then transfers the data located at the address stored in the MAR to the MBR via the data lines connecting the CPU to memory
- Transfer from memory to CPU is coordinated by the CU. To finish the cycle, the newly fetched data is transferred to the ACC.



IR[address portion] → MAR
MAR → Memory → MBR
MBR → ACC

CPU Execute Cycle
(ADD ACC Memory)

- This operation adds the data stored in the ACC with data that is stored in the memory location specified in the instruction using the ALU
- The address portion of the instruction is transferred from the IR to the MAR
- The CPU then transfers the data located at the address stored in the MAR to the MBR via the data lines connecting the CPU to memory
- The CU controls the transfer from memory to CPU. The ALU adds the data stored in the ACC and the MBR.
- To finish the cycle, the result of the addition is stored in ACC for future use



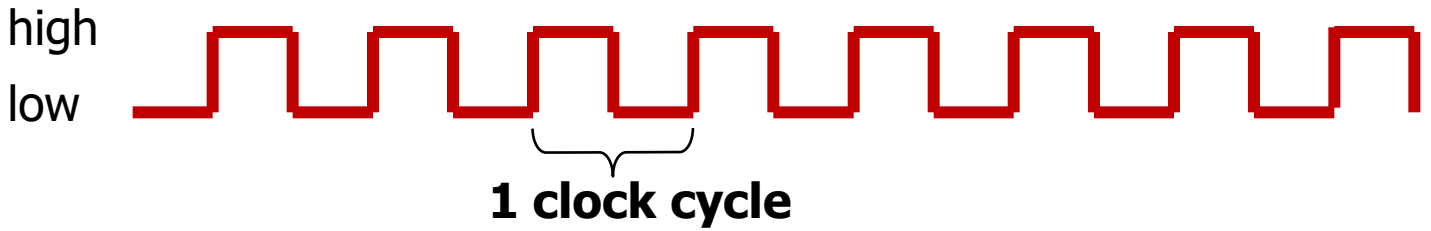
IR[address portion] → MAR
MAR → Memory → MBR
MBR + ACC → ALU
ALU → ACC

CPU Execute Cycle Categories

- CPU – Memory
  - Data may be transferred from memory to the CPU or from the CPU to memory
- CPU – I/O
  - Data may be transferred from an I/O module to the CPU or from the CPU to an I/O module
- Data Processing
  - The CPU may perform some arithmetic or logic operation on data via the ALU
- Control
  - An instruction may specify that the sequence of operation may be altered

Where Does the Clock Come in?

- CPUs use a digital clock pulse to synchronize the activities of the different hardware units



- Clock speed = #cycles per second
- For example, an **ALU** performs **one step** of an executing operation each time a clock pulse occurs
- Hence, an instruction with 20 steps will have a duration of 20 clock cycles

Clock Cycles and Speed

- A **66** MHz processor executes twice as fast as a **33** MHz processor, right?
- NO!
- The cycle time does not tell you how much work is getting done in each cycle
  - A 66 MHz computer requiring 15 cycles to perform an operation will perform worse than a 33 MHz computer than only requires 6 cycles to perform the same operation

Simple Set of Assembly Instructions

- Humans are not good in remembering bit patterns
- A set of words is defined to represent the bit patterns
- This collection of words is called **assembly language**
- An assembler translates the words into their bit patterns

LOADA mem	Load register A from memory address
LOADB mem	Load register B from memory address
CONB con	Load a constant value into register B
SAVEB mem	Save register B to memory address
SAVEC mem	Save register C to memory address
ADD	Add A and B and store the result in C
SUB	Subtract A and B and store the result in C
MUL	Multiply A and B and store the result in C
DIV	Divide A and B and store the result in C
COM	Compare A and B and store the result in test
JUMP addr	Jump to an address
JEQ addr	Jump if equal to address
JNEQ addr	Jump if not equal to address
JG addr	Jump if greater than to address
JGE addr	Jump if greater than or equal to address
JL addr	Jump if less than to address
JLE addr	Jump if less than or equal to address
STOP	Stop execution

Compiling into Assembly

5! (factorial): Sample program:

```
num = 1;
fac = 1;

for( ; num <= 5 ; num++) {
   fac = fac * num;
}
```

Example assumes that RAM addresses start:

- at 128 for the data,
- at 0 for the program instructions.

A, B, and C are CPU registers

0	CONB 1	num=1
1	SAVEB 128	
2	CONB 1	fac =1
3	SAVEB 129	
4	LOADA 128	If num > 5 then jump to 17
5	CONB 5	
6	COM	
7	JG 17	
8	LOADA 129	fac = fac * num
9	LOADB 128	
10	MUL	
11	SAVEC 129	
12	LOADA 128	num = num + 1
13	CONB 1	
14	ADD	
15	SAVEC 128	
16	JUMP 4	Loop back to if statement
17	STOP	
…		
128		This is where num is stored
129		This is where fac is stored

Number of Addresses

- 3 addresses
  - Operand 1, Operand 2, Result
  - a = b + c;
- 2 addresses
  - One address doubles as operand and result
  - a = a + b
  - Reduces length of instruction
- 1 address
  - Implicit second address: usually accumulator register
  - Common on early machines

Design Decisions

- Operation repertoire
  - How many operations?
  - What do they do?
  - How complex are they?
- Instruction formats
  - Length of operation code field?
  - Number of addresses?
- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?

Design Factors

- An important factor in computer design prior to 1980 was that all **memory was expensive**
  - Instructions were made **short** and **powerful**
- Lots of instructions of different lengths (simple and complicated)
- Each instruction might take **several clock ticks** to complete
- Interface with the computer's data memory in many different ways:
  - either dealing directly with the memory data
  - or demanding that data first be stored into temporary locations (registers)
  - or some mix of the two

Design Factors (cont'd)

- Time passed and memory became cheaper
- Proposed: simple instructions all of uniform length and that do a smaller amount of work

- Example:

  1 instruction in 10 clock ticks

  VS

  10 instructions each in 1 clock tick

CISC Architecture

- Complex Instruction Set Computer
- Characteristics:
  - instructions generally take more than 1 clock to execute
  - instructions of a variable size
  - instructions interface with memory in multiple mechanisms
  - lots and lots of instructions, some simple, some very complex
  - Initially, no pipelining possible

RISC Architecture

- Reduced Instruction Set Computer
- Characteristics:
  - instructions execute in one clock pulse
  - instructions of a fixed size
  - a small number of primitive instructions
  - instructions interface with memory via fixed mechanism
  - pipelining (a way to do more than one instruction at a time)

CISC vs RISC Example

CISC	RISC
MULT 128, 129 (example assumes that the MULT instruction stores the result of the multiplication in the register of the first operand)	LOAD A, 128 LOAD B, 129 PROD A, B STORE 128, A

Programs compiled in High Level Languages create an executable file (machine language). Which executable programs would be bigger? CISC or RISC?

Variations in Machine Architecture

Expression: A = B x C + D

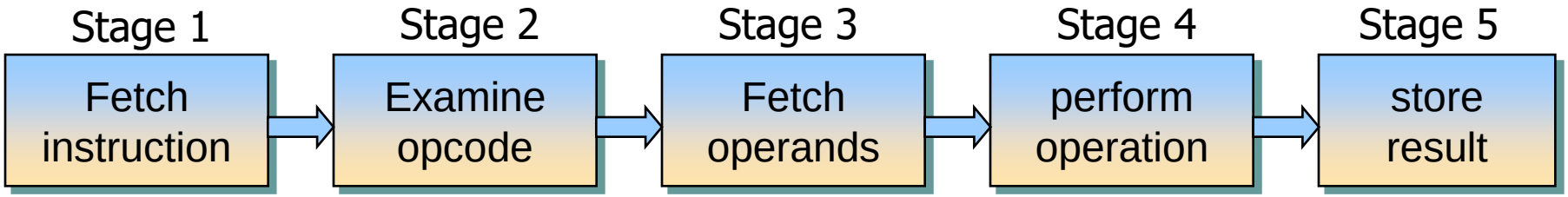
Three addresses	
mult B, C, A	A ← BxC
add D, A, A	A ← A+D

Two addresses	
load B, A	A ← B
mult C, A	A ← AxC
add D, A	A ← A+D

One address	
load B	Acc ← B
mult C	Acc ← Acc x C
add D	Acc ← Acc + D
store A	A ← Acc

Pipelining

- A RISC processor executes 1 instruction per clock cycle but more accurately **it completes 1 instruction on each clock cycle**
- To enable high speed, parallel hardware units are used that perform one step of the fetch-execute cycle
  - Hardware is arranged in a multistage pipeline –results from one stage are passed to the next hardware unit



Pipelining (cont'd)

- The speed of the pipeline arises because all stages can operate in parallel

Time

Clock	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
1	Inst. 1	-	-	-	-
2	Inst. 2	Inst. 1	-	-	-
3	Inst. 3	Inst. 2	Inst. 1	-	-
4	Inst. 4	Inst. 3	Inst. 2	Inst. 1	-
5	Inst. 5	Inst. 4	Inst. 3	Inst. 2	Inst. 1
6	Inst. 6	Inst. 5	Inst. 4	Inst. 3	Inst. 2
7	Inst. 7	Inst. 6	Inst. 5	Inst. 4	Inst. 3
8	Inst. 8	Inst. 7	Inst. 6	Inst. 5	Inst. 4

Computer Performance Evaluation: CYCLES PER INSTRUCTION

- Most computers run synchronously utilizing a CPU clock running at a constant clock rate, where:
- **Clock rate = 1 / clock cycle time**
- A computer machine instruction is comprised of a number of elementary or micro operations which vary in number and complexity depending on the instruction and the exact CPU organization and implementation
  - A micro operation is an elementary hardware operation that can be performed during one clock cycle
  - Examples: register operations: shift, load, clear, increment, ALU operations: add, subtract, etc.
- Thus a single machine instruction may take one or more cycles to complete, termed as Cycles Per Instruction (CPI)

Computer Performance Evaluation: PROGRAM EXECUTION TIME

- For a specific program compiled to run on a specific machine "A", the following parameters are provided:
  - The total instruction count of the program
  - The average number of cycles per instruction (average CPI)
  - Clock cycle of machine "A"
- How are we to measure the performance of a machine running this program?
  - Intuitively the machine is said to be faster or has better performance running this program, if the total execution time is shorter
  - Thus the inverse of the total measured program execution time is a possible performance measure or metric:
  - **PerformanceA = 1 / Execution TimeA**
  - What factors affect performance? How do we compare the performance of different machines, and improve performance?

Comparing Computer Performance using Execution Time

- To compare the performance of two machines “A”, “B” running a given program:

$$\textbf{Performance}_A = 1 / \textbf{Execution Time}_A$$
$$\textbf{Performance}_B = 1 / \textbf{Execution Time}_B$$

- Machine A is **n** times faster than machine B means:

$$n = \textbf{Performance}_A / \textbf{Performance}_B$$
$$= \textbf{Execution Time}_B / \textbf{Execution Time}_A$$

- Example (for a given program):

  Execution time on machine A: $\text{Execution}_A = 1$ sec

  Execution time on machine B: $\text{Execution}_B = 10$ sec

  $\text{Performance}_A / \text{Performance}_B = \text{Execution Time}_B / \text{Execution Time}_A$

  $= 10/1 = 10$

The performance of machine A is 10 times the performance of machine B when running this program, or: Machine A is said to be 10 times faster than machine B when running this program

CPU Execution Time

- A program is comprised of a number of instructions (instruction count)
  - Measured in: instructions/program
- The average instruction takes a number of cycles per instruction (CPI) to be completed
  - Measured in: cycles/instruction
- CPU has a fixed clock cycle time C = 1/clock rate
  - Measured in: seconds/cycle

CPU Execution Time (cont'd)

CPU execution time, measured in seconds/program, is the product of the previous three parameters as follows:

$$\mathbf{CPUExecTime = InstructionCount \times CPI \times ClockCycleTime}$$

OR

$$\mathbf{CPUExecTime = \frac{Seconds}{program} = \frac{Instructions}{program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}}$$

Example: CPU Execution Time

- A program is running on a specific machine with the following parameters:
  - Total instruction count: 10,000,000 instructions
  - Average CPI for the program: 2.5 cycles/instruction
  - CPU clock rate: 200 MHz
- What is the execution time for this program:

CPU time = Instruction count x CPI x Clock cycle

= 10,000,000 x 2.5 x 1 / clock rate

= 10,000,000 x 2.5 x 5×10^{-9}

= .125 seconds

CPU Performance Factors

	Instruction Count	Cycles per Instruction	Clock Cycle
Program	x	x	
Compiler	x	x	
Instruction Set Architecture (ISA)	x	x	
CPU Organization		x	x
Technology			x

Example: Performance Comparison

- From the previous example: a program is running on a specific machine with the following parameters:
  - Total instruction count: 10,000,000 instructions
  - Average CPI for the program: 2.5 cycles/instruction
  - CPU clock rate: 200 MHz
- Using the same program with these changes:
  - A new compiler used: New instruction count 9,500,000, New CPI: 3.0
  - Faster CPU implementation: New clock rate = 300 MHZ
- What is the speedup with the changes?

Speedup = $(10{,}000{,}000 \times 2.5 \times 5\times10^{-9}) / (9{,}500{,}000 \times 3 \times 3.33\times10^{-9})$

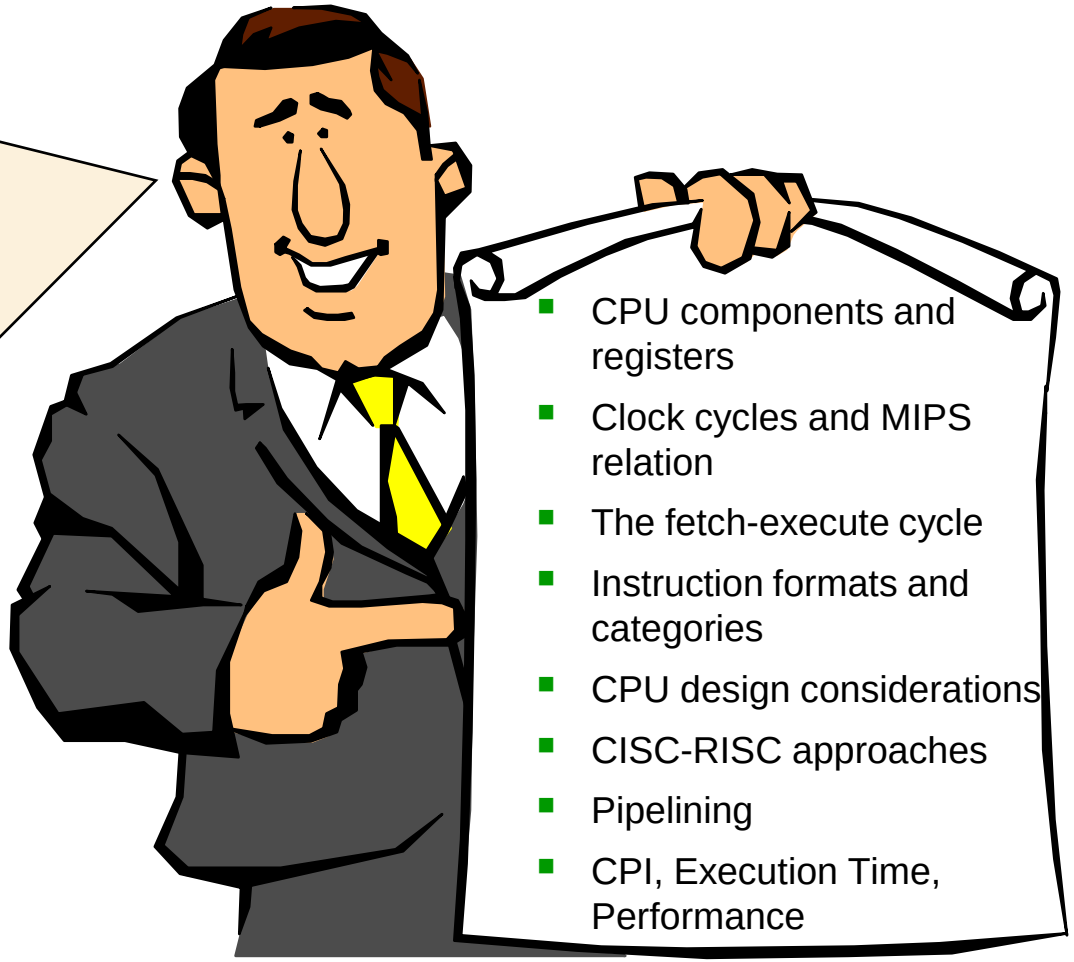
= .125 / .095 = 1.32 or 32 % faster after changes

Summary

Processors can be categorized according to flexibility and their role. A CPU is a programmable, special-purpose processor that contains many components.

Each CPU operates with a set of predefined instructions and uses a fetch-execute cycle in order to execute them in 1 or more clock cycles.

There are many different ways that an instruction set, and therefore a CPU, can be designed.
CISC and RISC are two different CPU architectures each of which has its own advantages and disadvantages.



- CPU components and registers
- Clock cycles and MIPS relation
- The fetch-execute cycle
- Instruction formats and categories
- CPU design considerations
- CISC-RISC approaches
- Pipelining
- CPI, Execution Time, Performance