Introduction and Overview of Computer Architecture and Organization

WHY STUDY COMPUTER ORGANIZATION AND ARCHITECTURE?

- The computer lies at the heart of computing.
- Without it most of the computing disciplines today would be a branch of theoretical mathematics.
- To be a professional in any field of computing today, one should not regard the computer as just a black box that executes programs by magic.
- All students of computing should acquire some understanding of a computer system's functional components, their characteristics, their performance, and their interactions.

WHY STUDY COMPUTER ORGANIZATION AND ARCHITECTURE?

- Students need to understand computer architecture in order to structure a program so that it runs more efficiently on a real machine.
- In selecting a system to use, they should be able to understand the tradeoff among various components, such as CPU clock speed vs. memory size.

Computer

- Takes Input
  - Data
- Processes it according to stored instructions
  - Instructions: Software, Programs
- Produces results as output
  - Information (numbers, words, sounds, images)

Types of Computers

- Computer
  - Special Purpose (Embedded Systems)
    - Oven
    - Television
    - Mobile
  - General Purpose (User-programmable)
    - Personal Computers
    - Mainframes or enterprise systems
    - Workstations
    - Notebook
    - Supercomputers

Computer Architecture

- **Computer architecture** refers to
  - those attributes of a system visible to a programmer
  - those attributes that have a direct impact on the logical execution of a program.
- Examples
  - Instruction set
  - The number of bits used to represent various data types (e.g., numbers, characters)
  - I/O mechanisms, and techniques for addressing memory.
- It is an architectural design issue whether a computer will have a multiply instruction.

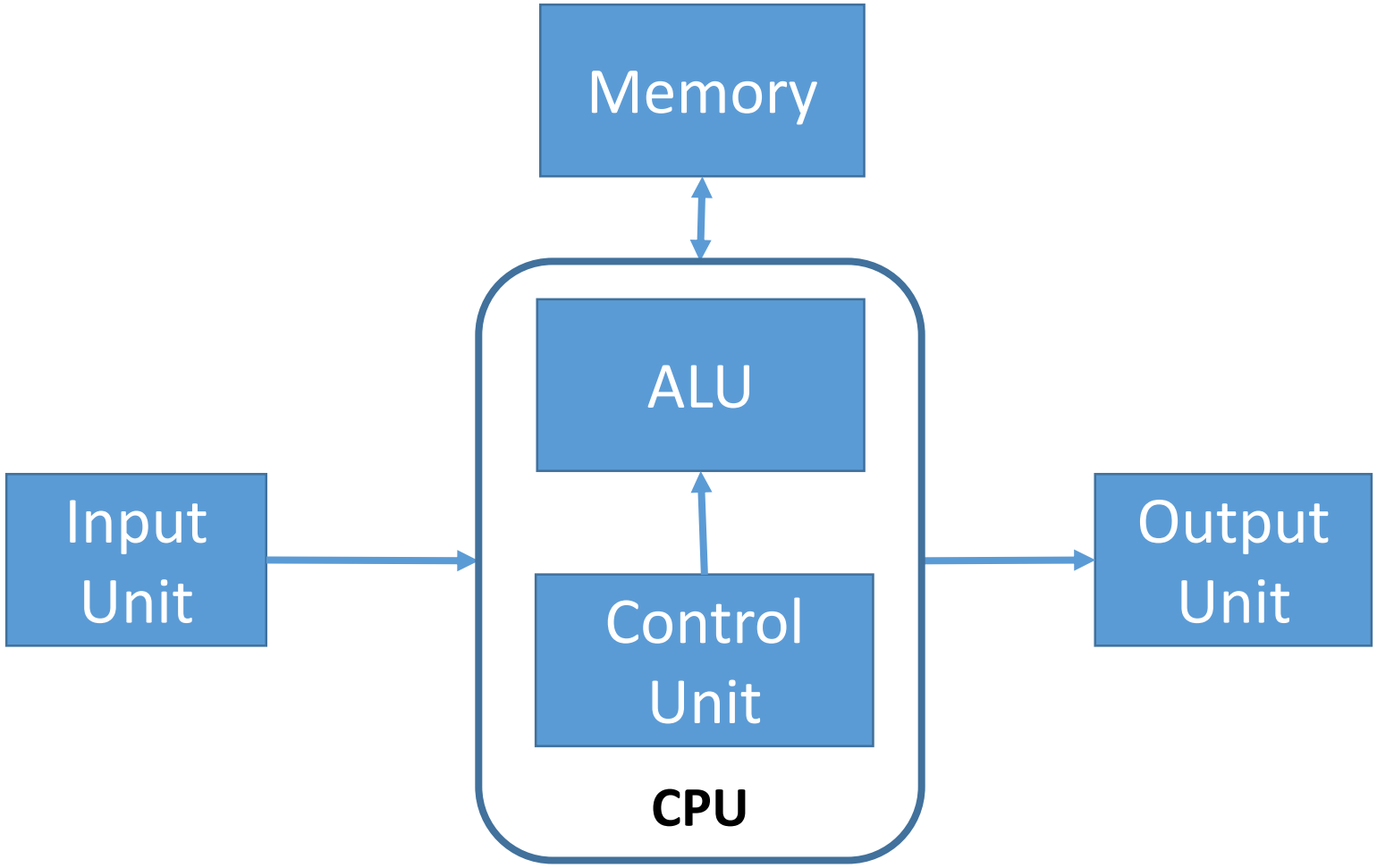
Computer Organization

- **Computer organization** refers to the operational units and their interconnections that realize the architectural specifications.
- Organizational attributes include those hardware details transparent to the programmer
- Examples
  - Control signals;
  - Interfaces between the computer and peripherals;
  - The memory technology used.
- It is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system.

Computer Architecture and Organization

- A particular architecture may span many years and encompass a number of different computer models, its organization changing with changing technology.
- IBM System/370 architecture - This architecture was first introduced in 1970 and included a number of models.
- The customer with modest requirements could buy a cheaper, slower model and, if demand increased, later upgrade to a more expensive, faster model

Functional Components of a Computer



Input Unit

Mouse 

Keyboard 

Joystick 

Light Pen 

Touch Pad 

Microphone 

Track Ball 

Scanner 

Digital Camera 

Memory – Physical Device to store programs or data

- This memory is of two fundamental types:
  - Main memory (Primary Memory)
    - Volatile – looses information when power is removed.
    - Main Storage
  - Secondary memory.
    - Non-volatile
    - Secondary or Mass Storage

Main Memory

- Closely connected to the processor.
- Stored data are quickly and easily changed.
- Holds the programs and data that the processor is actively working with.
- Interacts with the processor millions of times per second.
- Needs constant electric power to keep its information.
- Fast
- Expensive
- Low Capacity
- Works directly with the processor

Main Memory

ROM

RAM

Secondary Memory

- Connected to main memory through the bus and a controller.
- Stored data are easily changed, but changes are slow compared to main memory.
- Used for long-term storage of programs and data.
- Before data and programs can be used, they must be copied from secondary memory into main memory.
- Does not need electric power to keep its information.
- Slow
- Cheap
- Large Capacity
- Not connected directly to the processor

Secondary Memory

Output Devices

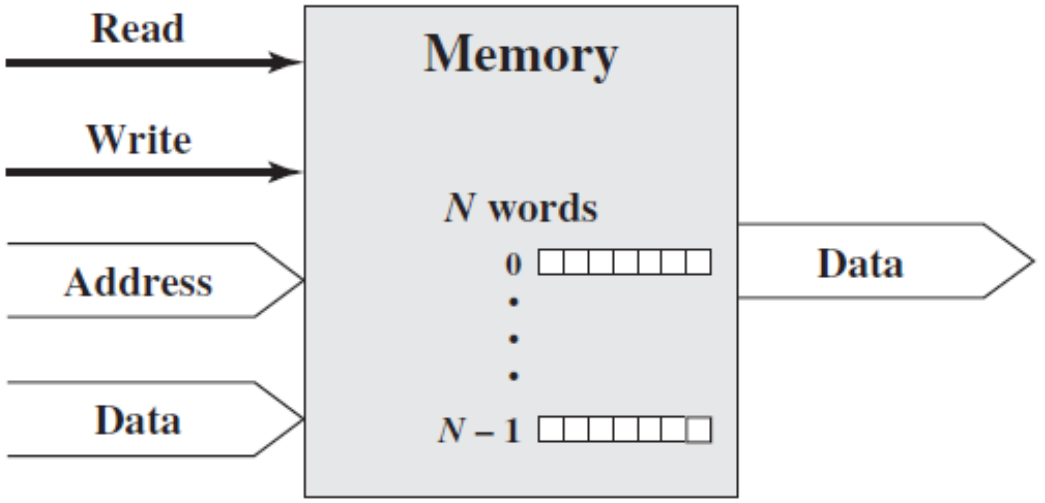
CPU – Central Processing Unit

Interconnection of Components

- The functional components of a computer need to communicate with each other to perform a task.
- Therefore, computer is a network of components.
- The collection of paths connecting the various components is called the *interconnection structure.* The design of this structure will depend on the exchanges that must be made among modules.

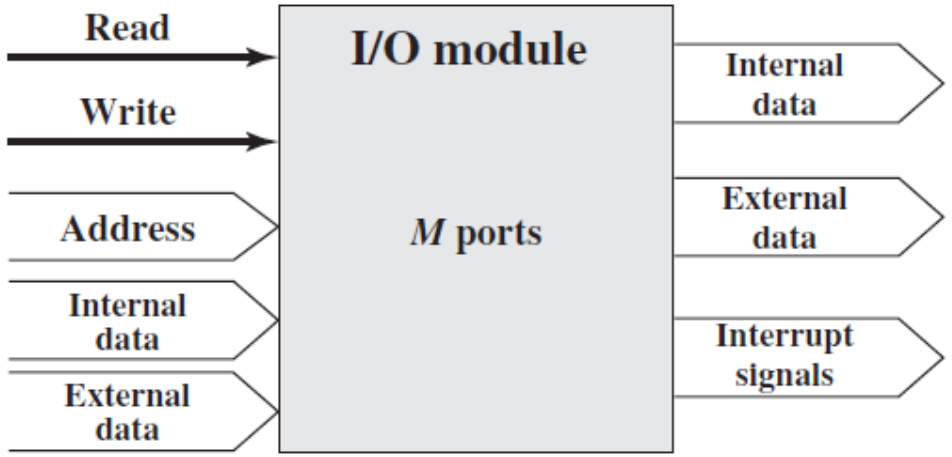
Input and output of a Memory Component

- Two Operations
  - Read
    - Input
      - Enable Read Control signal
      - Provide the address on address bus from wher data to read
    - Output
      - Data from the given address
  - Write
    - Input
      - Enable Write control signal
      - Provide the address on address bus where the data to be written
      - Provide the data to be written on data bus
    - Output
      - -



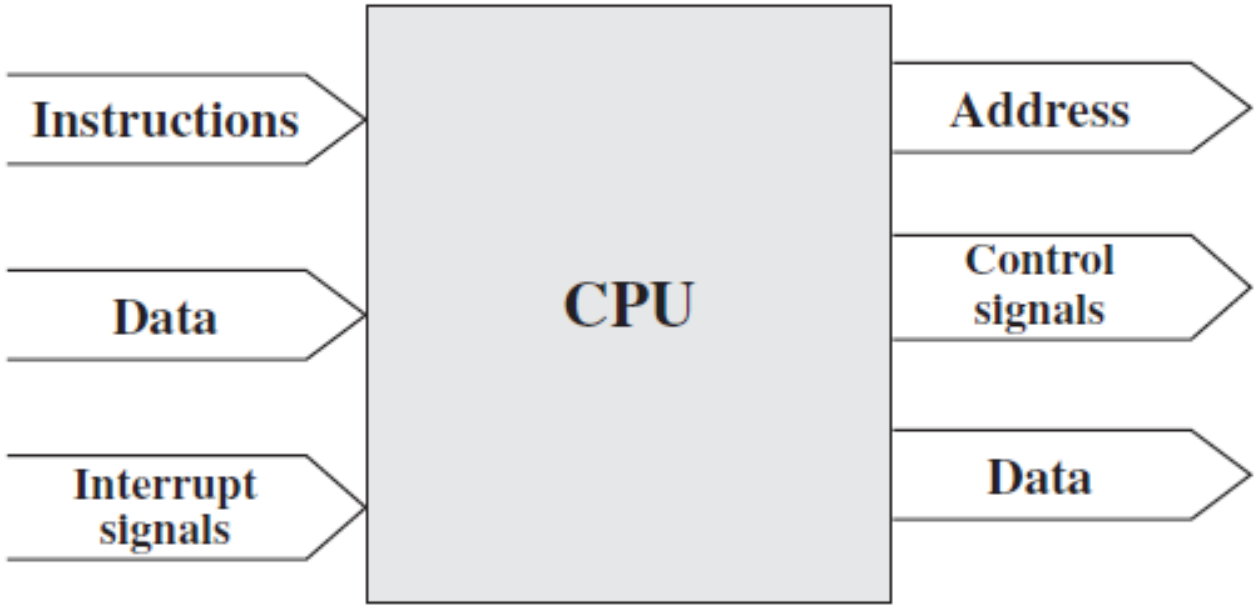
Input and Output of I/O Module

- I/O Module may control more than one external device.
- Each of the interface to an external device is referred as a *port* and given each a unique address (e.g., 0, 1, . . . ,M– 1)
- Finally, an I/O module may be able to send interrupt signals to the processor.



Input and Output of Processor component

- The processor reads in instructions and data, writes out data after processing, and uses control signals to control the overall operation of the system.
- It also receives interrupt signals.



Interconnections

- The interconnection structure must support the following types of transfers:
  - Memory ↔ Processor
  - I/O ↔ Processor
  - I/O ↔ Memory (DMA)

Bus Interconnection

- It is a path way connecting two or more devices.
- A bus consists of multiple communication pathways, or lines.
- Each line is capable of transmitting signals representing binary 1 and binary 0.
- Transmitting a word across a single line (bit by bit serially)
- Several lines of a bus can be used to transmit binary digits simultaneously (in parallel).
  - For example, an 8-bit unit of data can be transmitted over eight bus lines.
- A bus that connects major computer components (processor, memory, I/O) is called a *system bus*

Bus

- The lines can be classified into three functional groups: **data, address, and control lines**
- **Data Lines – Data bus** – provide a path for moving data among system modules.
  - The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the *width* of the data bus.
  - If the data bus is 32 bits wide and each instruction is 64 bits long, then the processor must access the memory module twice during each instruction cycle.

Address Bus

- **Address Lines – Address Bus** – Used to designate the source or destination of the data on the data bus.
- For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines.
- Typically, the higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port within the module.
- For example, on an 8-bit address bus, address 01111111 and below might reference locations in a memory module with 128 words of memory, and address 10000000 and above refer to devices attached to an I/O module.

Control Bus

- The **control lines** are used to control the access to and the use of the data and address lines.
- Because the data and address lines are shared by all components, there must be a means of controlling their use.
- Control signals transmit both command and timing information among system modules.
- Timing signals indicate the validity of data and address information.
- Command signals specify operations to be performed.

Typical control lines include

- **Memory write:** Causes data on the bus to be written into the addressed location
- **Memory read:** Causes data from the addressed location to be placed on the bus
- **I/O write:** Causes data on the bus to be output to the addressed I/O port
- **I/O read:** Causes data from the addressed I/O port to be placed on the bus
- **Transfer ACK:** Indicates that data have been accepted from or placed on the bus
- **Bus request:** Indicates that a module needs to gain control of the bus
- **Bus grant:** Indicates that a requesting module has been granted control of the bus
- **Interrupt request:** Indicates that an interrupt is pending
- **Interrupt ACK:** Acknowledges that the pending interrupt has been recognized
- **Clock:** Is used to synchronize operations
- **Reset:** Initializes all modules

The operation of the bus

- If one module wishes to send data to another, it must do two things:
  - Obtain the use of the bus, and
  - Transfer data via the bus.
- If one module wishes to request data from another module, it must
  - Obtain the use of the bus, and
  - Transfer a request to the other module over the appropriate control and address lines.
  - It must then wait for that second module to send the data.

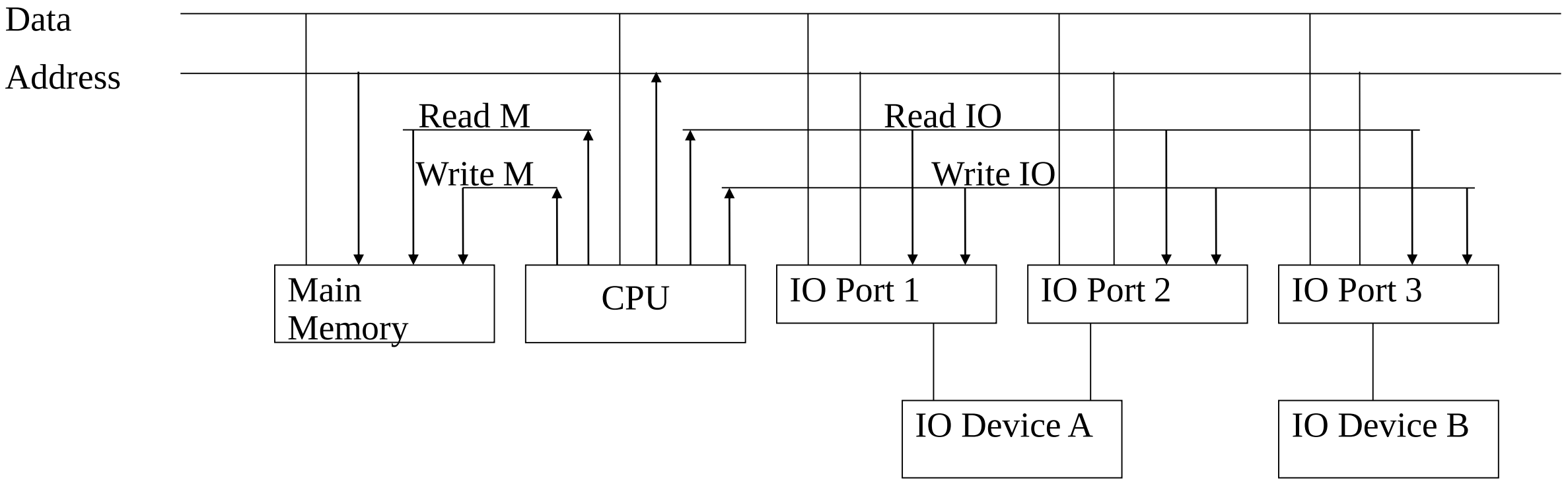
I/O Bus and Memory Bus

- *Functions of Buses*
  - *MEMORY BUS* is for information transfers between CPU and the Main Memory
  - *I/O BUS* is for information transfers between CPU and I/O devices through their I/O interface
- Many computers use a common single bus system for both memory and I/O interface units
  - separate control lines for each function
  - common address and data lines for both functions
- Some computer systems use two separate buses,
  - one to communicate with memory and the other with I/O interfaces

Isolated I/O – I/O Mapped I/O

- Separate I/O read/write control lines in addition to memory read/write control lines
- Separate (isolated) memory and I/O address spaces
- Distinct input and output instructions
- When CPU fetches and decodes the opcode of an I/O instruction, it places the address into the common address lines.
- Also enables read/write control lines => the address in the address lines is for interface register and not for a memory word.

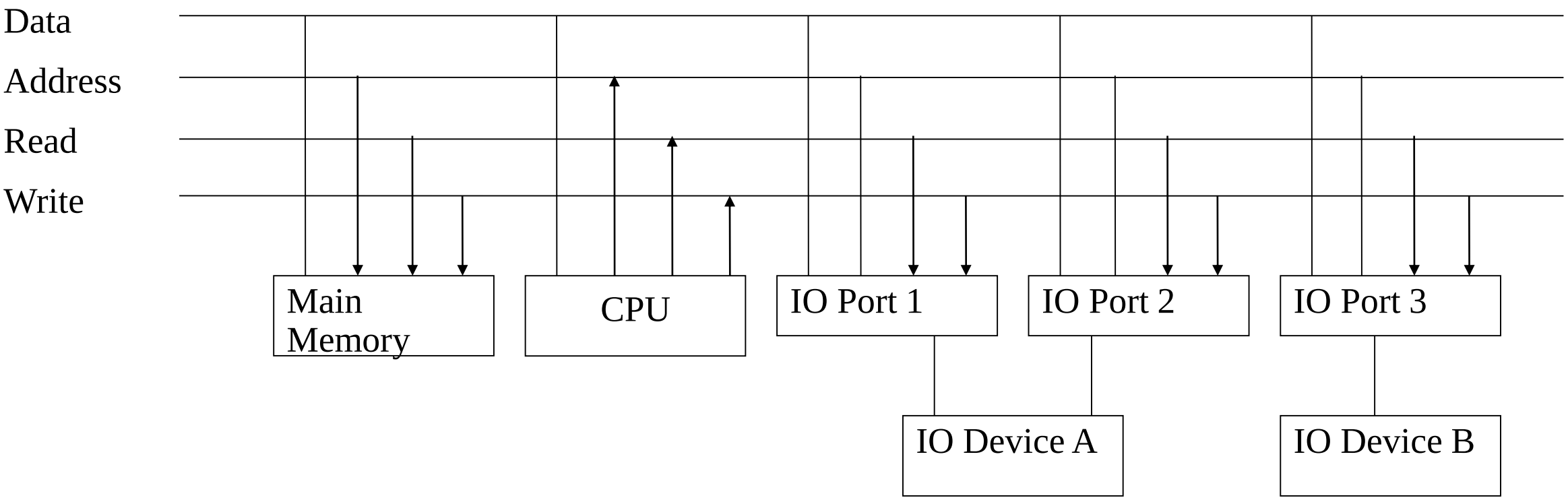
Isolated I/O


Data
Address
Read M
Write M
Read IO
Write IO
Main Memory
CPU
IO Port 1
IO Port 2
IO Port 3
IO Device A
IO Device B

Memory Mapped I/O

- A single set of read/write control lines
- No distinction between memory and I/O transfer
- Memory and I/O addresses share the common address space
  - reduces memory address range available
  - Memory addresses xffff0000 and above are used for I/O devices
- No specific input or output instruction
- The same memory reference instructions can be used for I/O transfers
- when the bus sees certain addresses, it knows they are not memory addresses, but are addresses for accessing I/O devices.

Memory Mapped I/O



Registers and Register Files

- A **register file** is an array of processor registers in a central processing unit (CPU)
- The Register File is the highest level of the memory hierarchy.
- Register file can be implemented using SRAM with multiple ports.
- Such RAMs are distinguished by having dedicated read and write ports, whereas ordinary multiport SRAMs will usually read and write through the same ports.
- Two roles
  - User-visible registers
  - Control and status registers

User-Visible Registers

- General Purpose
- Data
- Address
- Condition Codes

General Purpose Registers

- True general purpose registers – register can contain the operand for any Opcode
- Restricted – used for specific operations – floating point and stack operations. (dedicated registers)
- Data registers – used only to hold data and cannot be employed in the calculation of an operand address – Accumulator (AC)
- Address registers
  - Segment registers – holds the address of the base of the segment.
  - Index registers – used for indexed addressing and may be auto-indexed
  - Stack pointer – points to the top of the stack (if there is a user-visible stack addressing, stack is in memory)

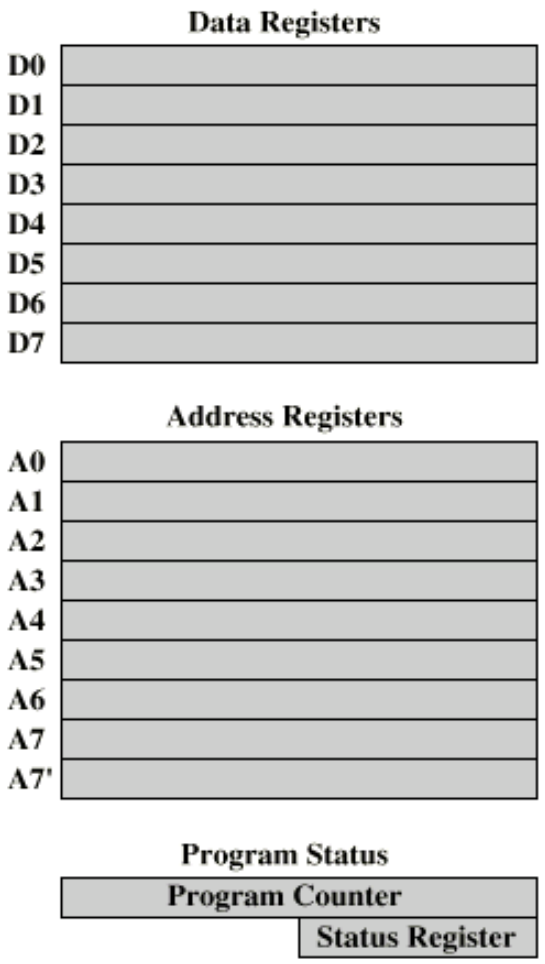
Condition Code Registers – Flags

- Condition codes are bits set by the CPU hardware as the result of operations.
- Machine instructions allow these bits to be read by implicit reference.
- Programmer cannot alter them
- In some machines, sub-routine call will result in the automatic saving of all user-visible registers, to be restored on return.
- Sets of individual bits, flags
  - e.g. result of last operation was zero
- Can be read by programs
  - e.g. Jump if zero – simplifies branch taking

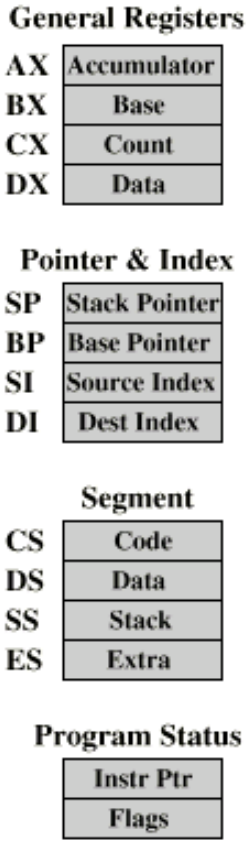
Control & Status Registers

- Not visible to the user
- May be visible in a control or operating system mode (supervisory mode)
- Registers essential to instruction execution:
  - Program Counter (PC)
  - Instruction Register (IR)
  - Memory Address Register (MAR) – connects to address bus
  - Memory Buffer Register (MBR) – connects to data bus, feeds other registers

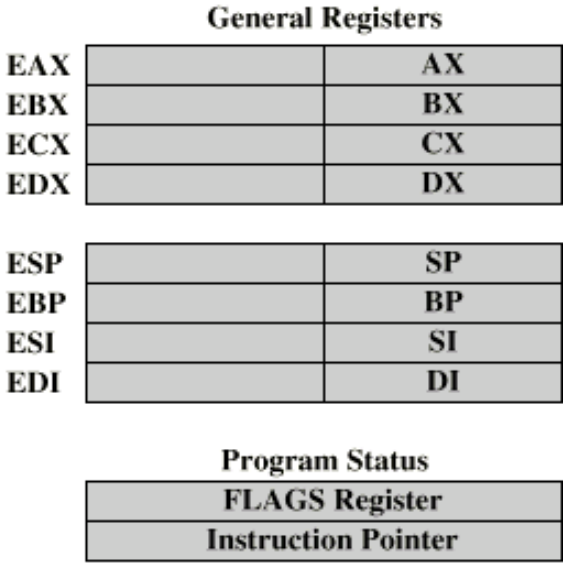
Example Register Organizations



(a) MC68000

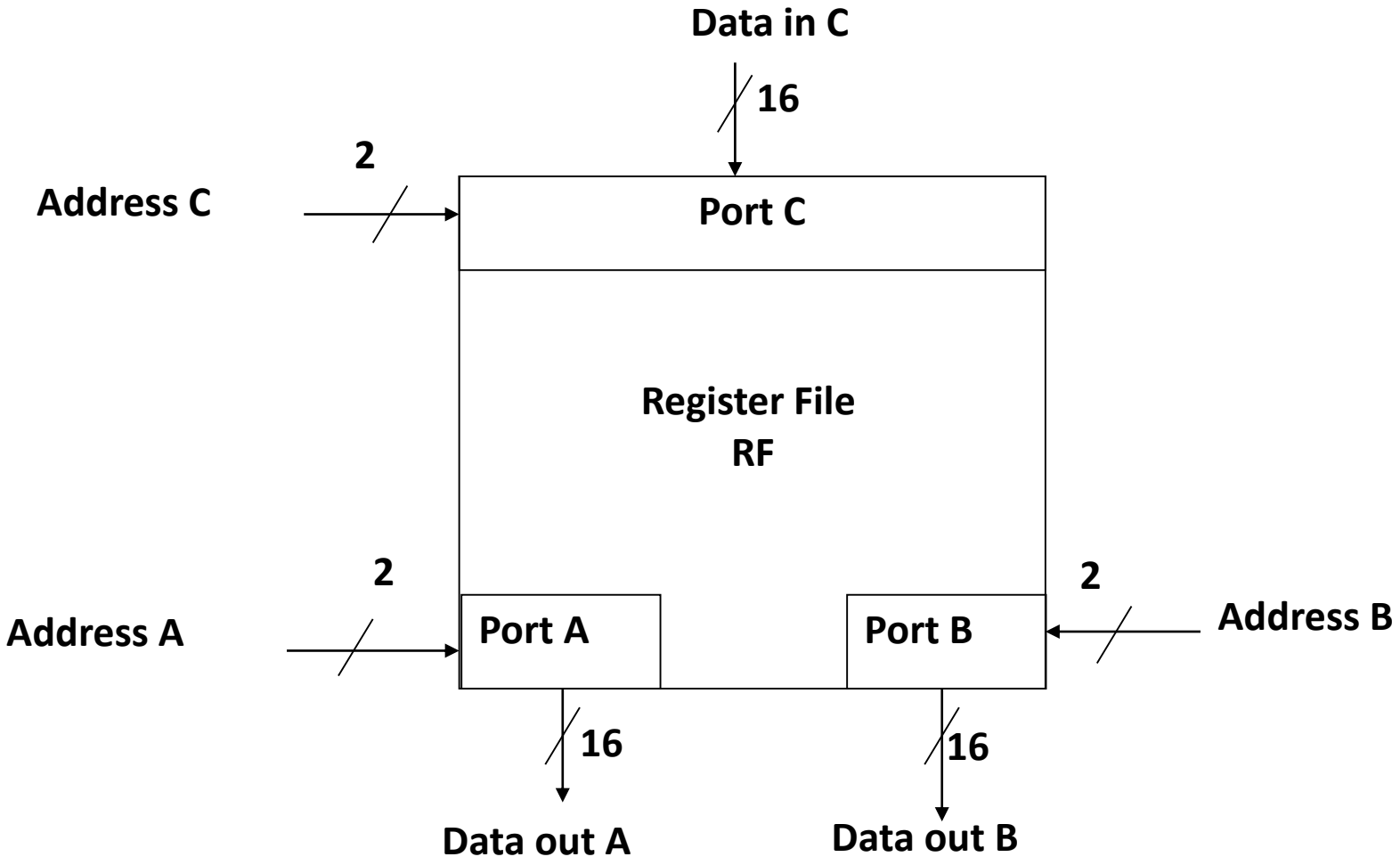


(b) 8086

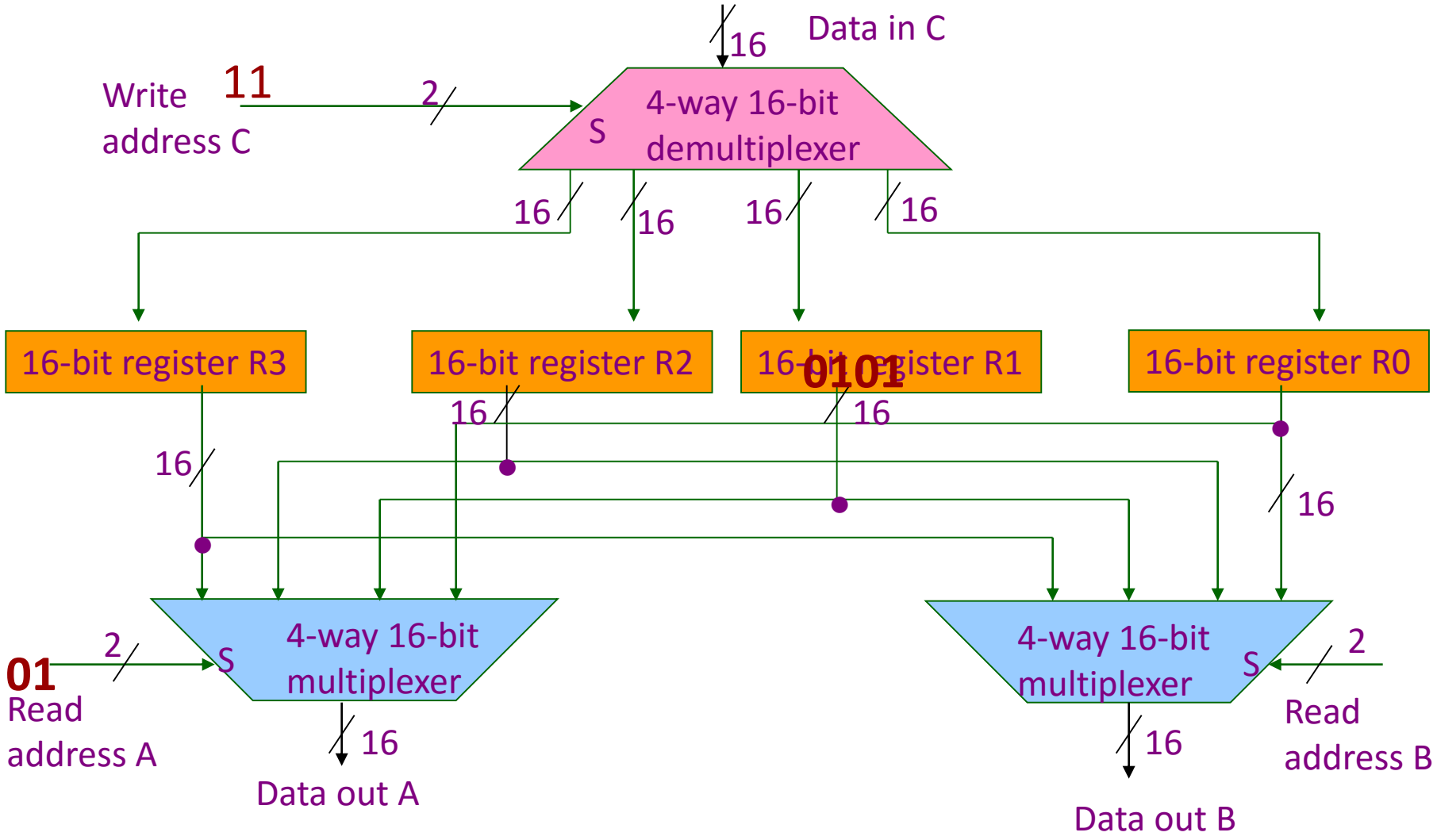


(c) 80386 - Pentium II

A register file with three access ports - symbol



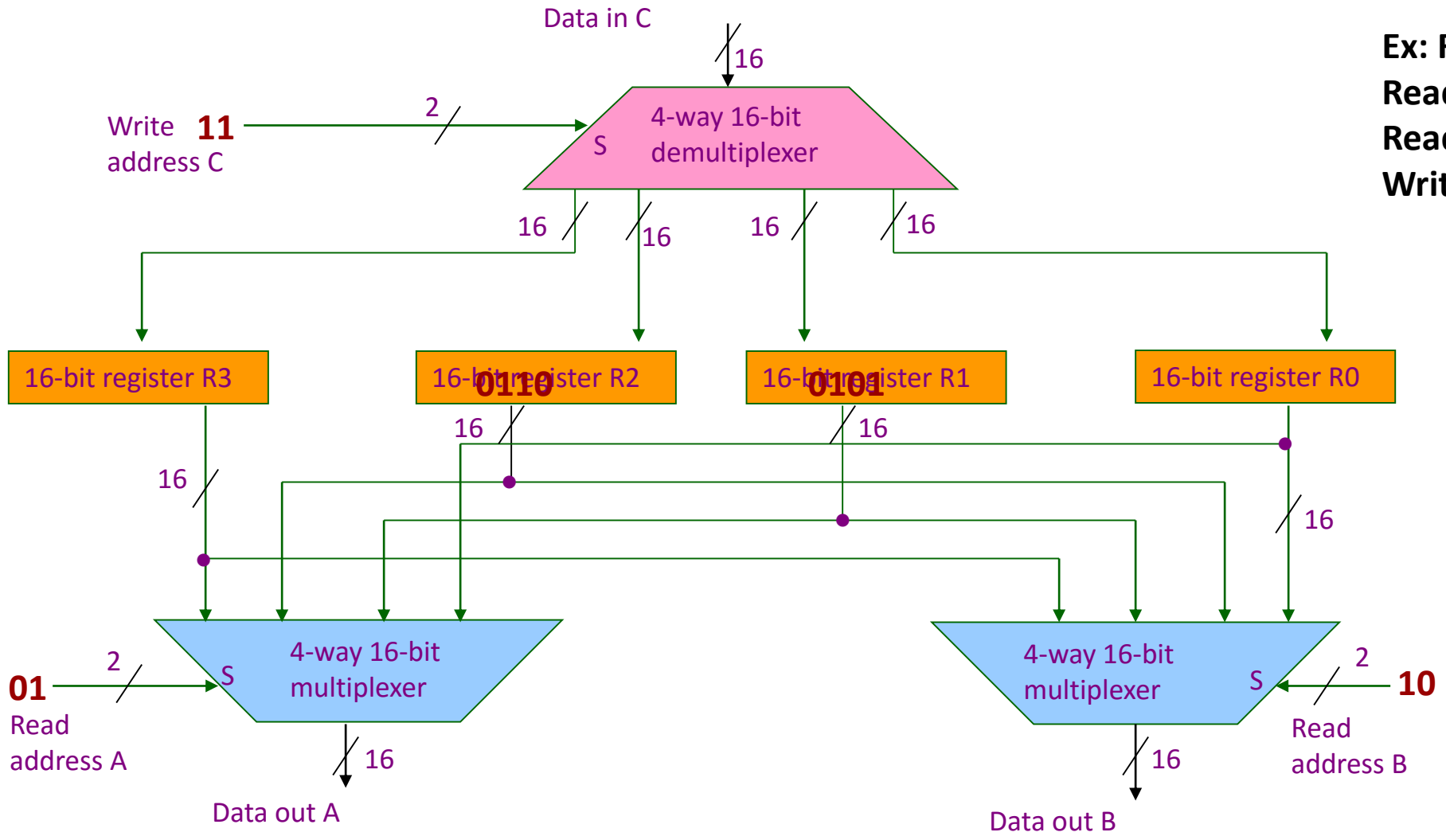
A Register File with three access ports – logic diagram



Ex: R3 ← R1 + R2
Read Address A = 01
Read Address B = 10
Write Address C = 11

A Register File with three access ports – logic diagram

1011



Ex: R3 ← R1 + R2
Read Address A = 01
Read Address B = 10
Write Address C = 11

IAS Computer

- Developed by John Von Neumann in 1940 at Princeton University.
- In IAS computer, IAS stands for

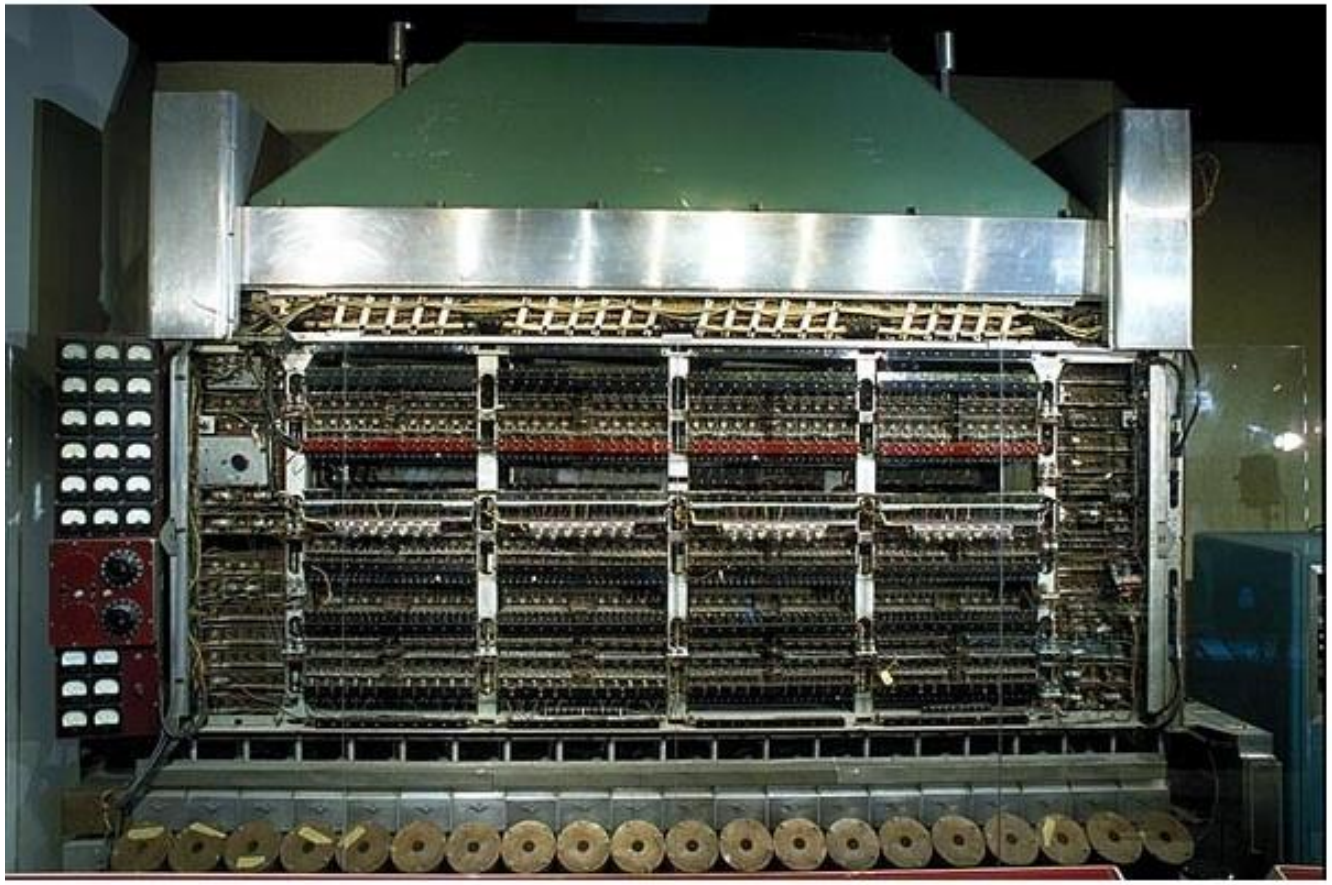
  Institute for Advanced Studies

Organization of Von-Neumann Machine (IAS Computer)

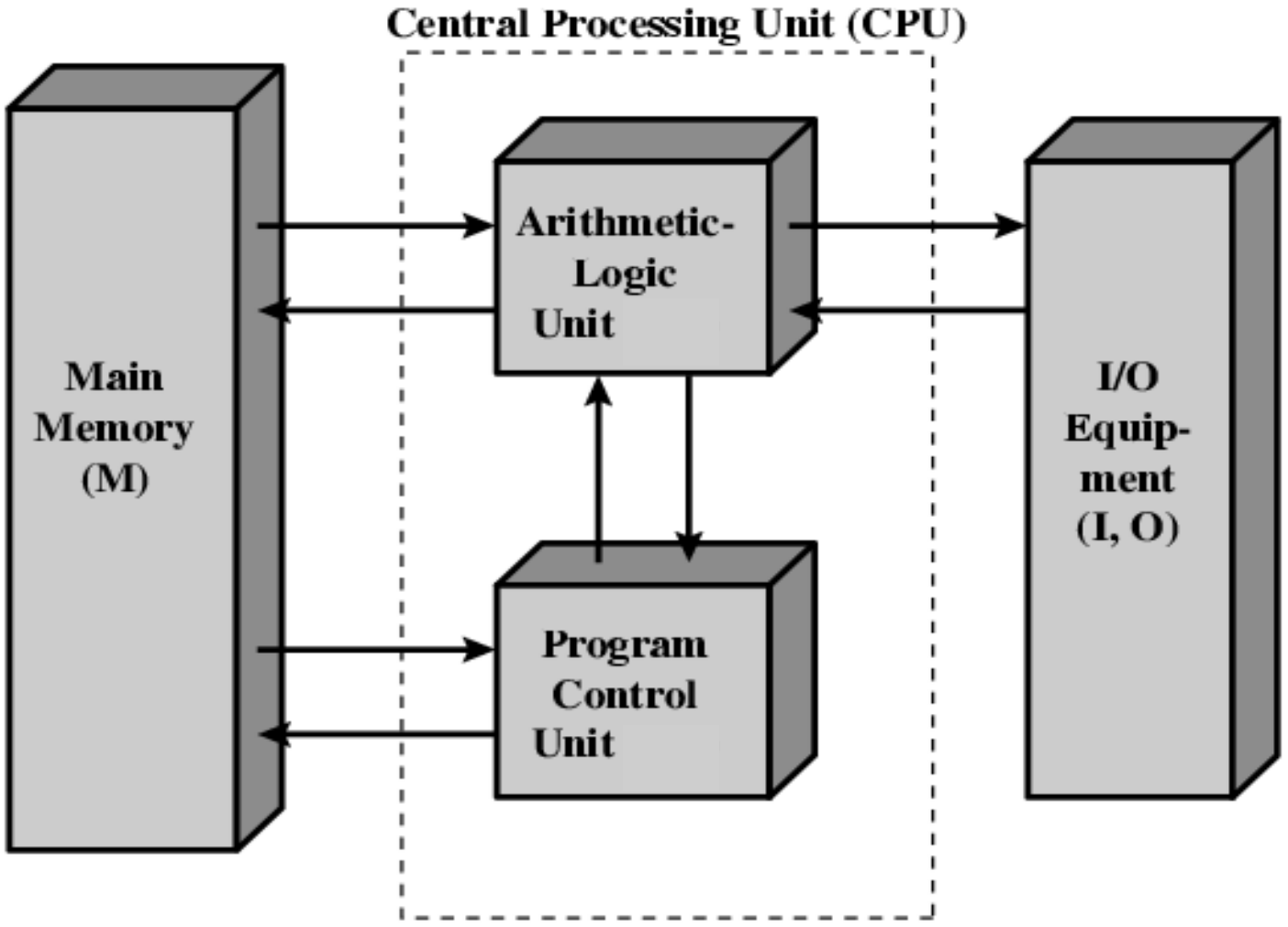
- The task of entering and altering programs for ENIAC was extremely tedious
- Stored program concept – says that the program is stored in the computer along with any relevant data
- A stored program computer consists of a p*rocessing unit* and an attached *memory system*.

IAS Computer

The IAS Computer, 1952



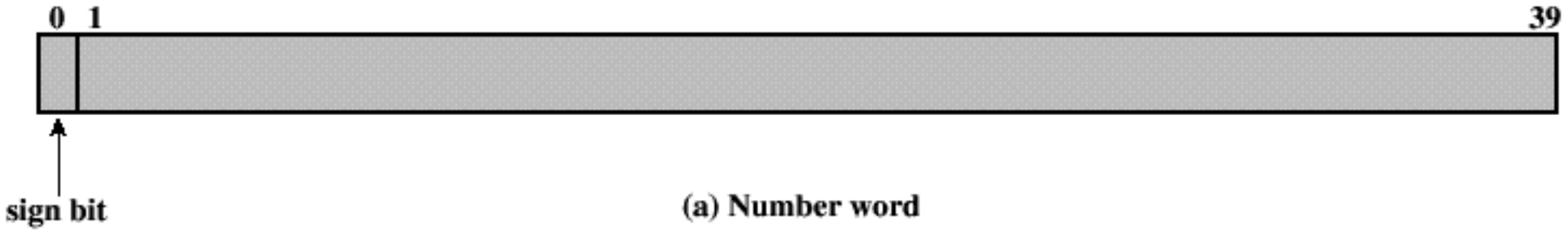
Structure of Von Neumann Machine



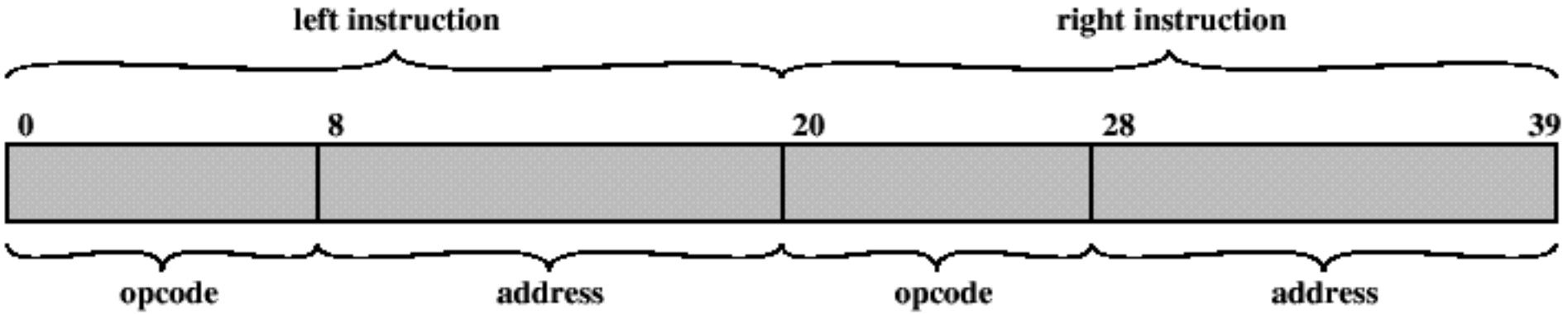
Memory of the IAS

- 1000 storage locations called words.
- Word length - 40 bits.
- A word may contain:
  - A numbers stored as 40 binary digits (bits) – sign bit + 39 bit value
  - An instruction-pair. Each instruction:
    - An opcode (8 bits)
    - An address (12 bits) – designating one of the 1000 words in memory.

Instruction Format of IAS Computer



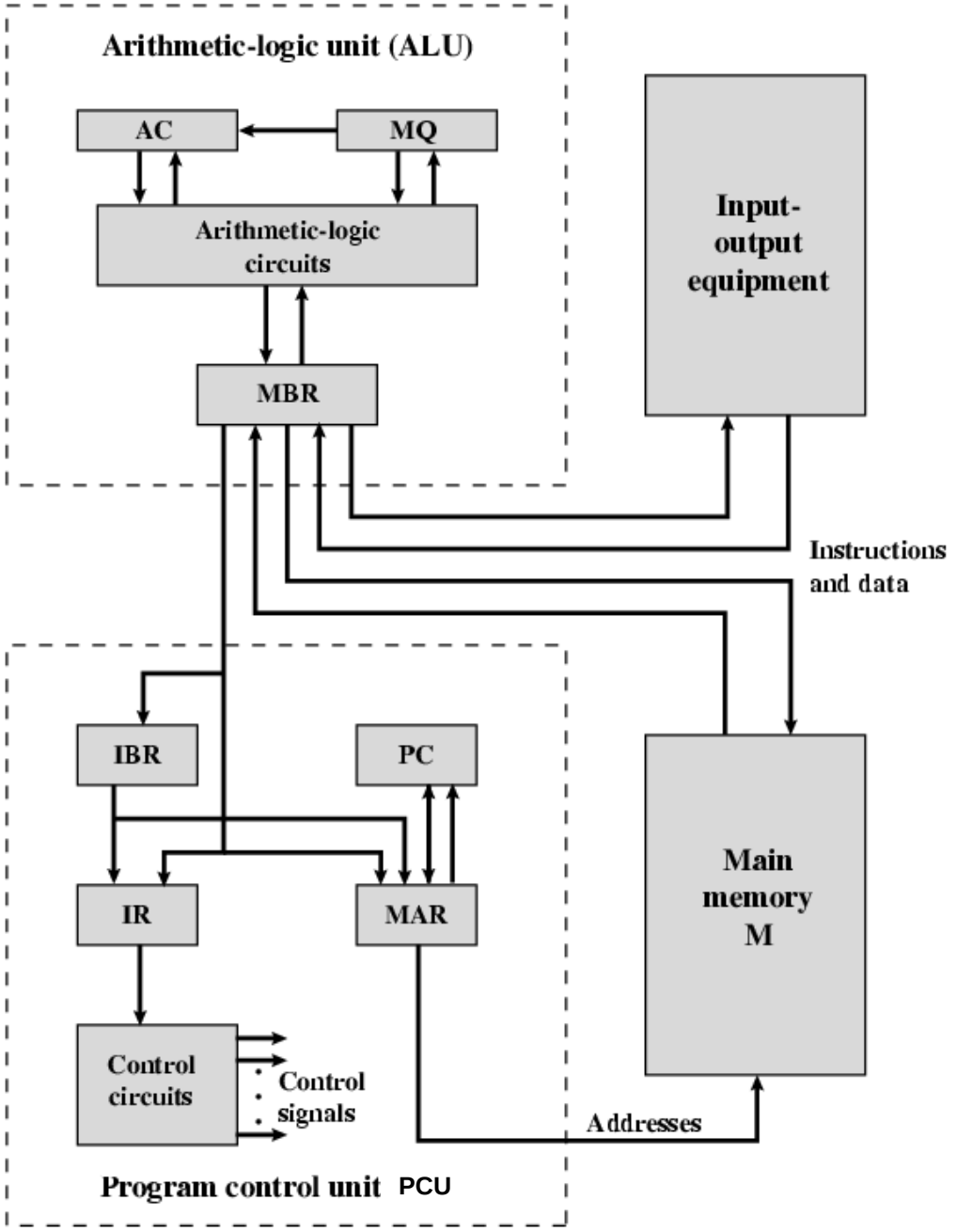
(a) Number word



(b) Instruction word

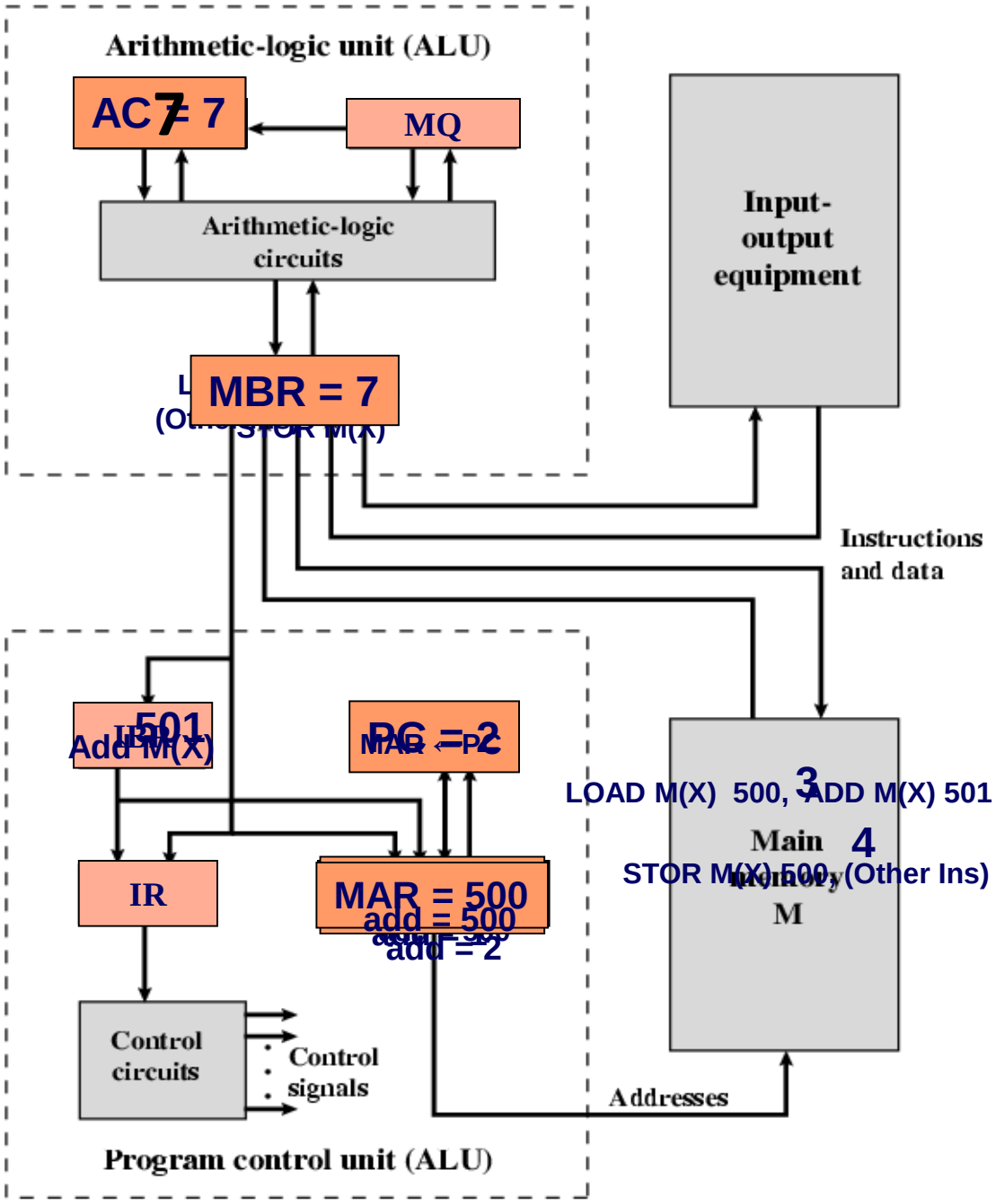
Von Neumann Machine

- **MBR: Memory Buffer Register**
  - contains the word to be stored in memory or just received from memory.
- **MAR: Memory Address Register**
  - specifies the address in memory of the word to be stored or retrieved.
- **IR: Instruction Register -** contains the 8-bit opcode currently being executed.
- **IBR: Instruction Buffer Register**
  - temporary store for RHS instruction from word in memory.
- **PC: Program Counter -** address of next instruction-pair to fetch from memory.
- **AC: Accumulator & MQ: Multiplier quotient** - holds operands and results of ALU ops.



MEMORY
1. LOAD M(X) 500, ADD M(X) 501
2. STOR M(X) 500, (Other Ins)
.....
500. 3
501. 4

PC	2
MAR	500
MBR	STOR M(X) 500, (Other Ins)
IR	STOR M(X)
IBR	(Other Ins)
AC	7

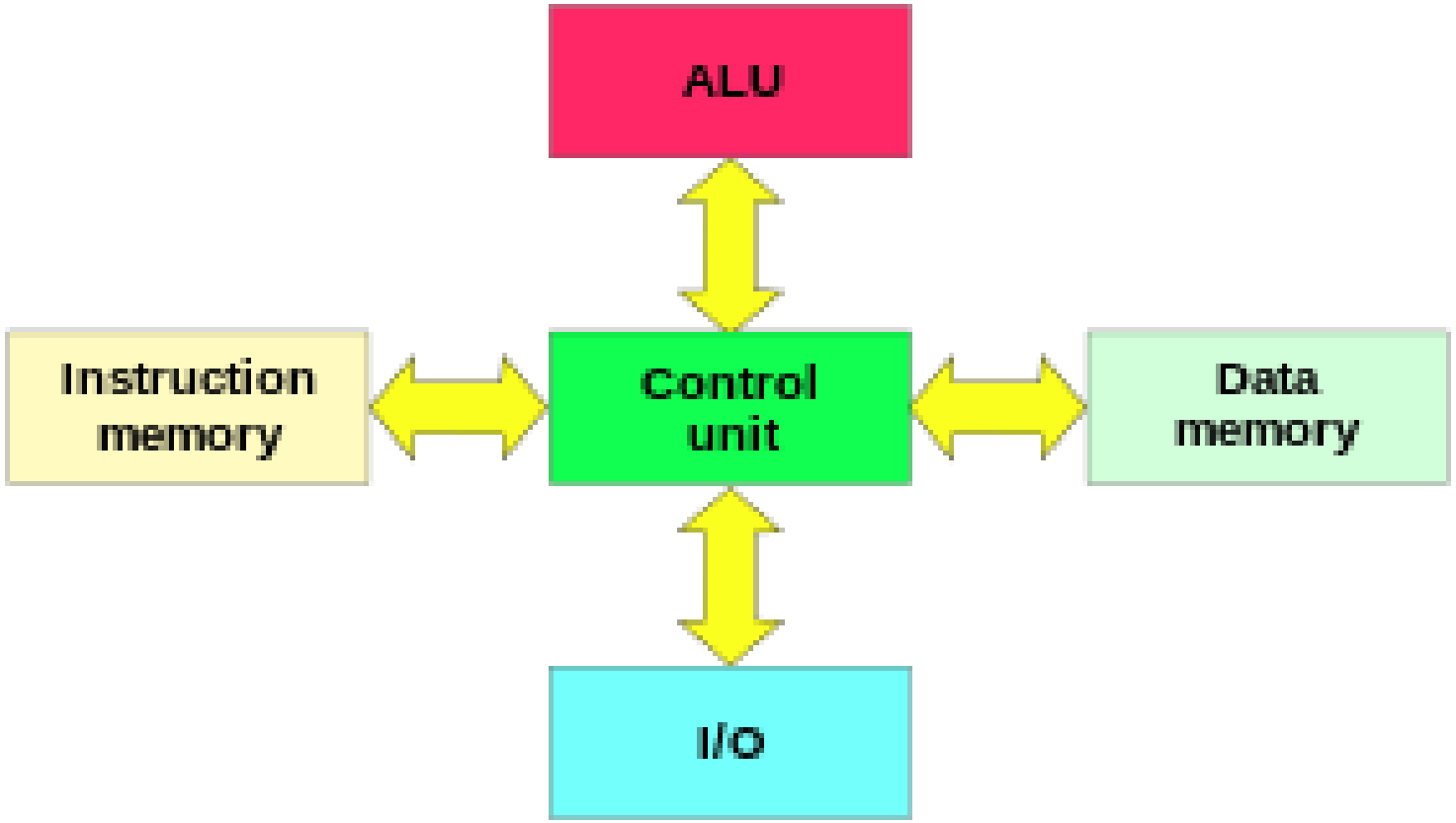


Register Transfer Operations

1. **LOAD M(X) 500, ADD M(X) 501 (PC = 1)**
   - **MAR ← PC**
   - **MBR ← M[MAR]**
   - **IBR ← MBR[20:39]**
   - **IR ← MBR[0:7]**
   - **MAR ← MBR[8:19]**
   - **MBR ← M[MAR]**
   - **AC ← MBR**
   - **IR ← IBR[0:7]**
   - **MAR ← IBR[8:19]**
   - **PC ← PC + 1**
   - **MBR ← M[MAR]**
   - **AC ← AC + MBR**

2. **STOR M(X) 500, (Other Ins)**
   - **MAR ← PC**
   - **MBR ← M[MAR]**
   - **IBR ← MBR[20:39]**
   - **IR ← MBR[0:7]**
   - **MAR ← MBR[8:19]**
   - **MBR ← AC**
   - **M[MAR] ← MBR**

Harvard Architecture



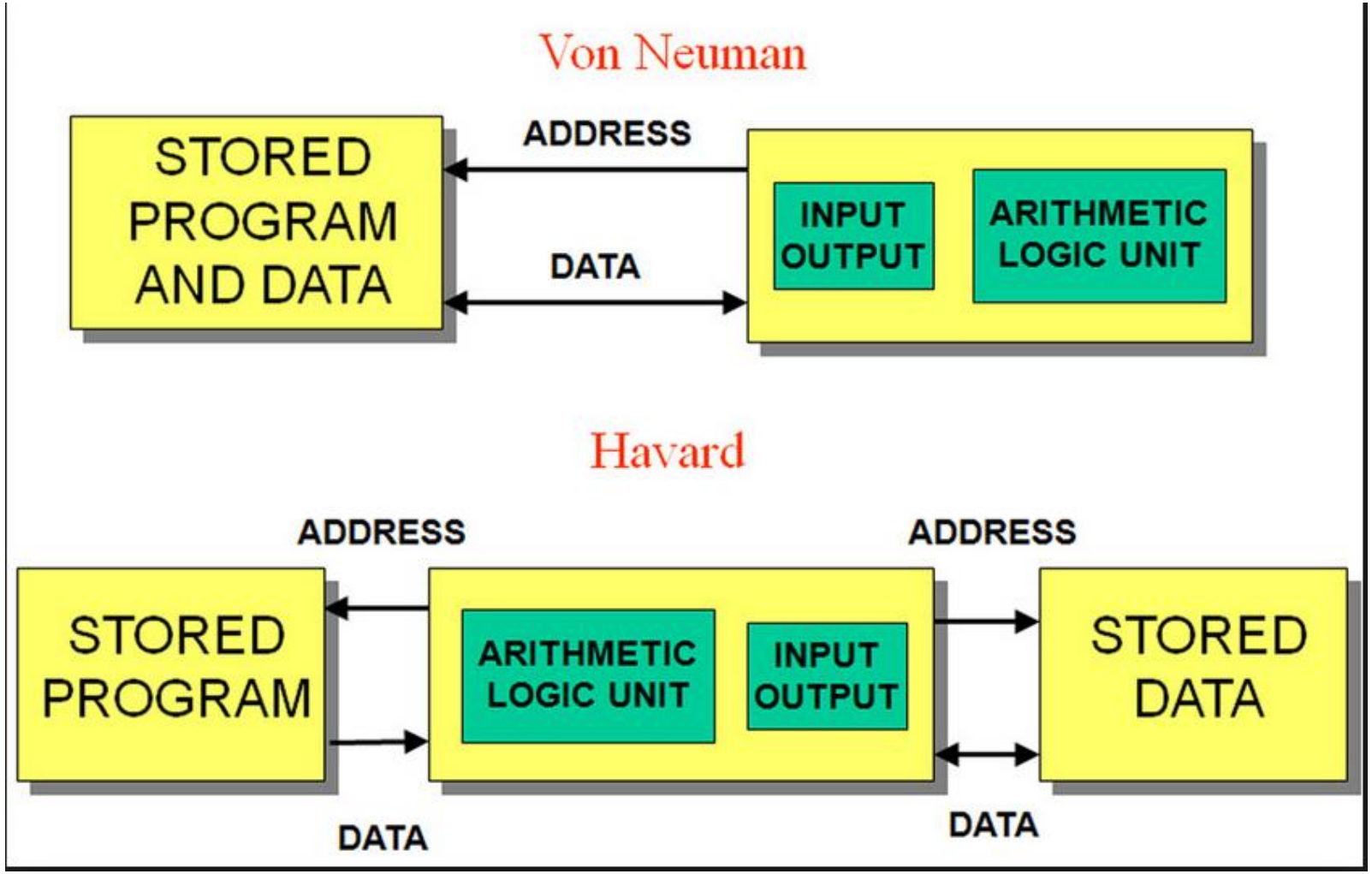
Harvard architecture

- The **Harvard architecture** is a computer architecture with physically separate storage
- These early machines had data storage entirely contained within the central processing unit, and provided no access to the instruction storage as data.
- Programs needed to be loaded by an operator; the processor could not initialize itself.

Harvard Architecture

- In a Harvard architecture, there is no need to make the two memories share characteristics.
- In particular, the word width, timing, implementation technology, and memory address structure can differ.
- Instructions – read-only memory
- Data memory – read-write memory.
- In some systems, there is much more instruction memory than data memory so instruction addresses are wider than data addresses.
- Ex: Mark1

Von-Neumann vs Harvard Architecture



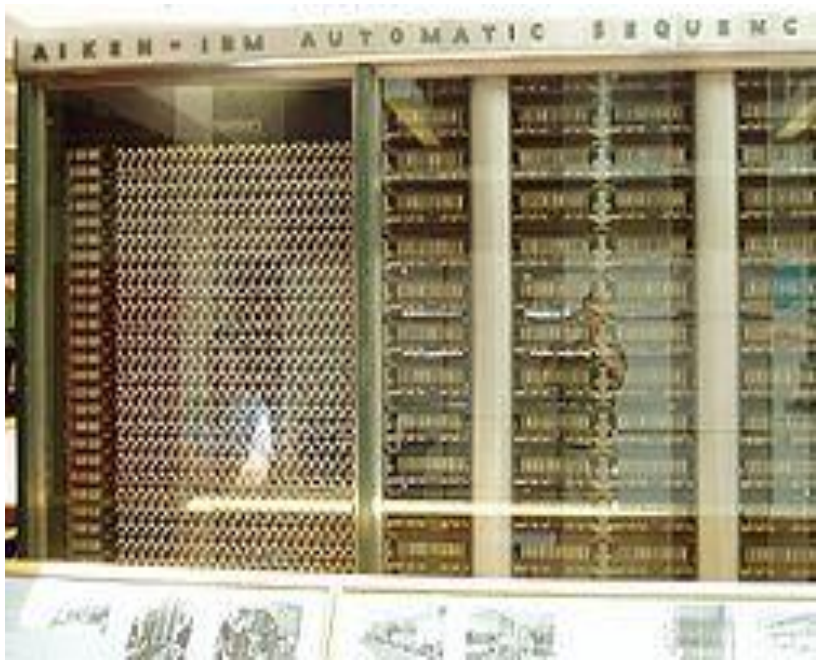
Von-Neumann vs Harvard Architecture

Von-Neumann Architecture	Harvard Architecture
The data and program are stored in the same memory	The data and program memory are separate. Both memories can use different sizes.
The code is executed serially and takes more clock cycles.	The code is executed in parallel
The programs can be optimized in lesser size	The program tend to grow big in size
Parallel access to data and program is not possible.	Two memories with two buses allow parallel access to data access and instructions
One bus is simpler for the control unit design	Control unit for 2 buses is more complicated and more expensive
Error in program can rewrite instruction and crash program execution	Program can’t write itself

Modified Harvard Architecture

- A modified Harvard architecture machine is very much like a Harvard architecture machine, but it relaxes the strict separation between instruction and data while still letting the CPU concurrently access two (or more) memory buses.
- The most common modification includes separate instruction and data caches backed by a common address space.
- While the CPU executes from cache, it acts as a pure Harvard machine.
- When accessing backing memory, it acts like a von Neumann machine
- Ex: x86 processors

Harvard Mark1



The left end consisted of electromechanical computing components

The right end included data and program readers, and automatic typewriters

Closeup of input/output and control readers

Performance of a Processor - Instruction Execution Rate

- A processor is driven by a clock with a constant frequency f or, equivalently, a constant cycle time τ , where $\tau = 1/f$.
- Instruction count – I_c – The number of machine instructions executed for that program until it runs to completion or for some defined time interval
- CPI - Average cycles per instruction CPI for a program.
- If all instructions required the same number of clock cycles, then CPI would be a constant value for a processor.
- However, on any give processor, the number of clock cycles required varies for different types of instructions, such as load, store, branch, and so on.

Instruction Execution Rate

- i – Instruction type
- CPI_i - The number of cycles required
- I_i – The number of executed instructions of type i for a given program.
- Then we can calculate an overall CPI as follows

$$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times I_i)}{I_c}$$

Processor Time

- The processor time T needed to execute a given program can be expressed as

$$T = I_c \times CPI \times \tau$$

- During the execution of an Instruction, part of the work is done by the processor, and part of the time a word is being transferred to or from memory.
- In this latter case, the time to transfer depends on the memory cycle time, which may be greater than the processor cycle time.
- We can rewrite the preceding equation as

$$T = I_c \times [p + (m \times k)] \times \tau$$

- Where,
- p – the number of processor cycles needed to decode and execute the instruction,
- m – the number of memory references needed
- k – The ratio between memory cycle time and processor cycle time

MIPS

- A common measure of performance for a processor is the rate at which instructions are executed, expressed as Millions of Instructions Per Second (MIPS), referred to as the **MIPS rate**.
- We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

MIPS Example

- Consider the execution of a program which results in the execution of 2 million instructions on a 400-MHz processor.
- The instruction mix and the CPI for each instruction type are given below based on the result of a program trace experiment:

Instruction Type	CPI	Instruction Mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	8	10%

$$CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (8 \times 0.1) = 2.24.$$

$$\text{MIPS rate is } (400 \times 10^6)/(2.24 \times 10^6) \approx 178.$$

MFLOPS

- Floating point performance is expressed as millions of floating-point operations per second (MFLOPS), defined as follows:

$$\text{MFLOPS rate} = \frac{\textit{Number of executed floating-point operations in a program}}{\textit{Execution time} \times 10^6}$$

Benchmarks

- Measures such as MIPS and MFLOPS have proven inadequate to evaluate the performance of processors.
- Because of differences in instruction sets, the instruction execution rate is not a valid means of comparing the performance of different architectures.
- For example, consider this high-level language statement:

$$A = B + C$$

- With the traditional instruction set architecture, referred to as a complex instruction set computer (CISC), this instruction can be compiled into one processor instruction:

add mem(B), mem(C), mem (A)

Benchmarks

- On a typical RISC machine, the compilation would look something like this:

```
load mem(B), reg(1);
load mem(C), reg(2);
add reg(1), reg(2), reg(3);
store reg(3), mem (A)
```

- Because of the nature of the RISC architecture, both machines may execute the original high-level language instruction in about the same time.
- If this example is representative of the two machines, then if the CISC machine is rated at 1 MIPS, the RISC machine would be rated at 4 MIPS.
- But both do the same amount of high-level language work in the same amount of time.

Benchmarks

- Measure the performance of systems using a set of benchmark programs.
- The same set of programs can be run on different machines and the execution times compared.
- Collection of benchmark suites is defined and maintained by the System Performance Evaluation Corporation (SPEC), an industry consortium.
- Ex: SPEC CPU2006, SPECjvm98, SPECweb99

Amdahl's Law

- Amdahl's law was first proposed by Gene Amdahl in [AMDA67] and deals with the potential speedup of a program using multiple processors compared to a single processor.
- Consider a program running on a single processor such that a fraction (1 – f) of the execution time involves code that is inherently serial and a fraction f that involves code that is infinitely parallelizable with no scheduling overhead.
- Let T be the total execution time of the program using a single processor.

Speedup

- Then the speedup using a parallel processor with N processors that fully exploits the parallel portion of the program is as follows:

$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on N parallel processors}}$$

$$= \frac{T(1-f)+Tf}{T(1-f)+\frac{Tf}{N}} = \frac{1}{(1-f)+\frac{f}{N}}$$

Speedup

- Two important conclusions can be drawn:
  - When f is small, the use of parallel processors has little effect.
  - As N approaches infinity, speedup is bound by 1/(1 – f), so that there are diminishing returns for using more processors.

Speedup

- Consider any enhancement to a feature of a system that results in a speedup.
- The speedup can be expressed as

$$\text{Speedup} = \frac{\text{Performance after enhancement}}{\text{Performance before enhancement}} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}}$$

- Suppose that a feature of the system is used during execution a fraction of the time *f*, before enhancement, and that the speedup of that feature after enhancement is SU_f. Then the overall speedup of the system is

$$\text{Speedup} = \frac{1}{(1 - f) + \frac{f}{SU_f}}$$

Speedup – Example

- Suppose that a task makes extensive use of floating-point operations, with 40% of the time is consumed by floating-point operations.
- With a new hardware design, the floating-point module is speeded up by a factor of K.
- Then the overall speedup is:

$$\text{Speedup} = \frac{1}{0.6 + \frac{0.4}{K}}$$

- Thus, independent of K, the maximum speedup is 1.67.

References

- Text Book – William Stallings, “Computer Organization and Architecture, Designing for performance”, 8th edition, Prentice Hall.
- Internet Sources.