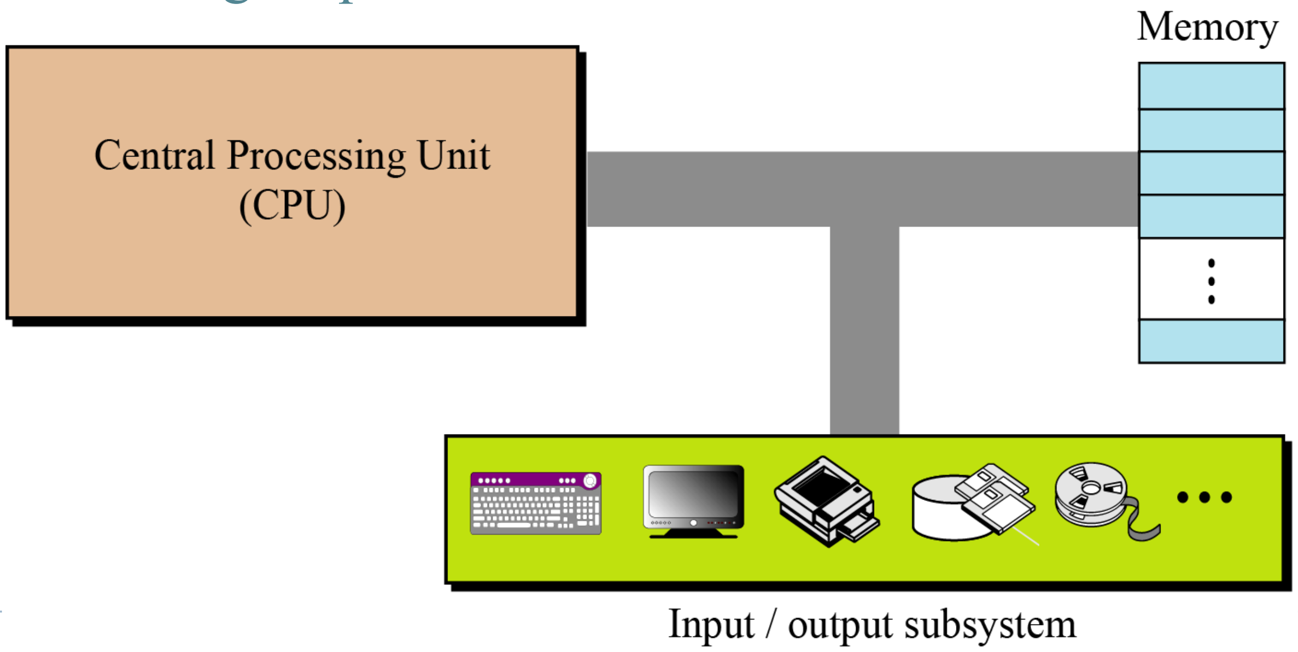
Computer Organization

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

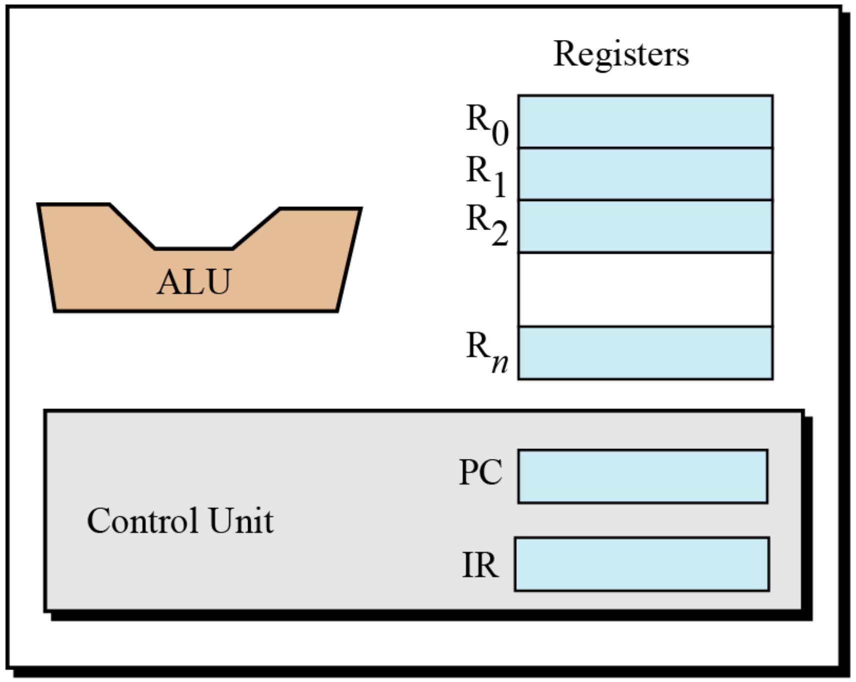
Introduction

- We discuss the organization of a stand-alone computer
  - We can divide the parts that make up a computer into three broad categories or subsystem: the *central processing unit (CPU) (中央處理單元)*, the *main memory (主記憶體)*, and the *input/output subsystem (輸入/輸出子系統)*
  - We will show how a simple, hypothetical computer can run a simple program to perform primitive arithmetic or logic operations



Central Processing Unit

- The central processing unit (CPU) performs operations on data
  - In most architectures it has three parts: an arithmetic logic unit (ALU) (算術邏輯單元), a control unit (控制單元), and a set of registers (暫存器)
- The arithmetic logic unit (ALU)
  - Performs logic, shift, and arithmetic operations on data
- The control unit
  - Controls the operation of each subsystem
  - Controlling is achieved through signals sent from the control unit to other subsystems



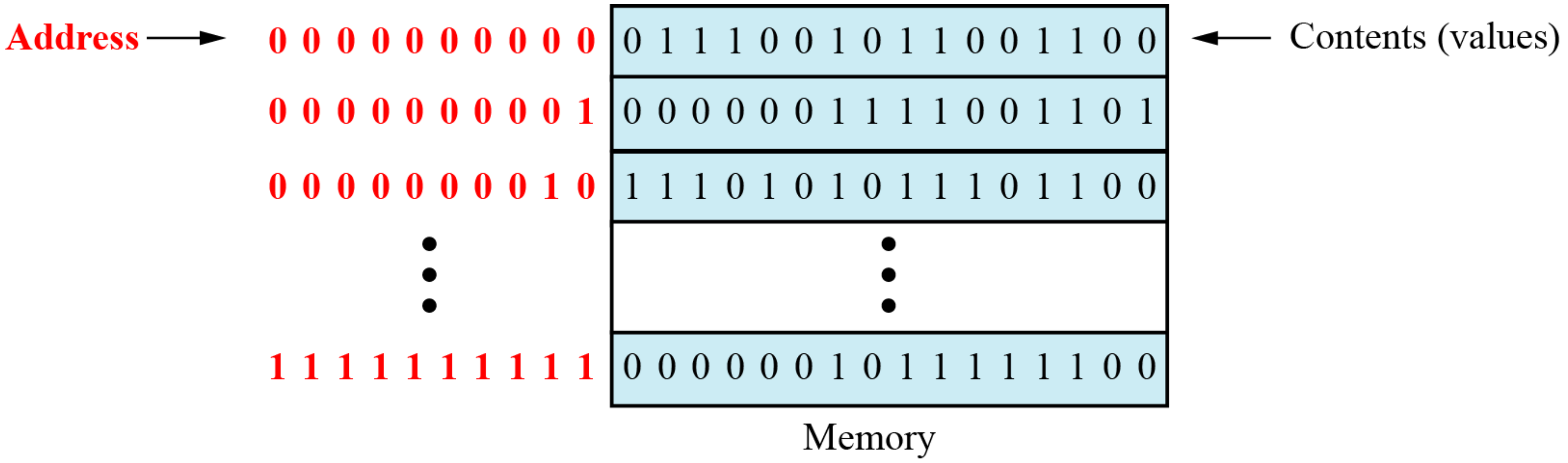
Central Processing Unit (CPU)

Central Processing Unit

- Registers are fast stand-alone storage locations that hold data temporarily
- Data registers (資料暫存器)
  - Data registers hold the input data, intermediate results, and the result of the operations to speed up the operations
  - Data registers are named R_0 to R_n
- Instruction registers (指令暫存器)
  - Computers store programs in their memory. The CPU is responsible for fetching instructions one by one from memory, storing them in the instruction register, decoding them, and executing them
- Program counter (程式計數器)
  - The program counter keeps track of the instruction currently being executed. After execution of the instruction, the counter is incremented to point to the address of the next instruction in memory

Main memory

- Main memory is the second major subsystem in a computer
  - It consists of a collection of locations, each with a unique *address (位址)* as the identifier
  - Data is transferred to and from memory in groups of bits called *words (字組)*. A word can be a group of 8 bits, 16 bits, 32 bits, or 64 bits (and growing)
    - If the word is 8 bits, it is referred to as a byte. The term "byte"(位元組) is so common in computer science that sometimes a 16-bit word is referred to as a 2-byte word



Main memory - Address space (位址空間)

- Accessing a word in memory requires an identifier
  - At the hardware level, each word is identified by an address
  - The total number of uniquely identifiable locations in memory is called the *address space*. For example, a memory with 64 kilobytes and a word size of 1 byte has an address space that ranges from 0 to 65,535. The memory units are shown below

Unit	*Exact Number of Bytes*	*Approximation*
kilobyte	2^{10} (1024) bytes	10^3 bytes
megabyte	2^{20} (1,048,576) bytes	10^6 bytes
gigabyte	2^{30} (1,073,741,824) bytes	10^9 bytes
terabyte	2^{40} bytes	10^{12} bytes

https://en.wikipedia.org/wiki/Metric_prefix

Main memory - Address space

- Because computers operate by storing numbers as bit patterns, a memory address is also represented as a bit pattern
  - So if a computer has 64 kilobytes (2^{16}) of memory with a word size of 1 byte, we need a bit pattern of 16 bits to define an address
  - Addresses can be represented as unsigned integers (we do not have negative addresses). In other words, the first location is referred to as address 0000000000000000 (address 0), and the last location is referred to as address 1111111111111111 (address 65535)
  - In general, if a computer has N words of memory, we need an unsigned integer of size $\log_2 N$ bits to refer to each memory location

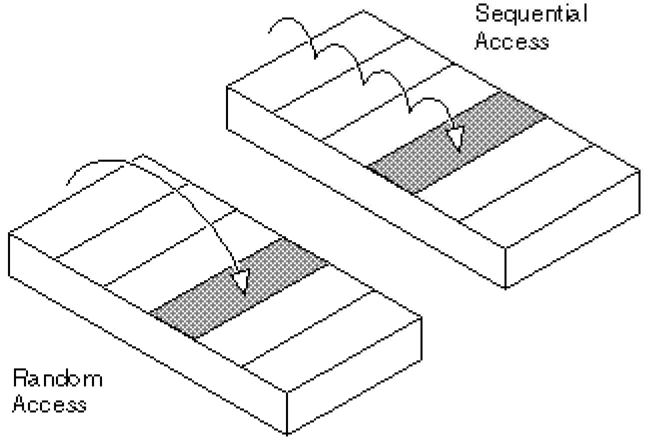
Main memory - Address space

- A computer has 32 MB of memory. Each word in this computer is 1 bytes. How many bits are needed to address any single word in memory?

- A computer has 128 MB of memory. Each word in this computer is eight bytes. How many bits are needed to address any single word in memory?

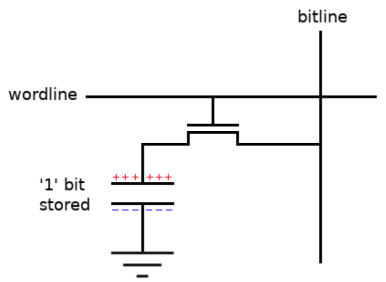
Main memory – memory type

- Random access memory (RAM) (隨機存取記憶體)
  - In a random access device, a data item can be accessed without the need to access all data items located before it
  - RAM can be read from and written to. Importantly, we can overwrite the locations of the RAM
  - RAM is *volatile (揮發性)*: the information (program or data) is lost if the computer is powered down
- Static RAM (SRAM)
  - Uses flip-flop gates to hold data. The gates hold their state (0 or 1), which means that data is stored as long as the power is on and there is no need to refresh memory locations
  - SRAM is fast but expensive



https://www.webopedia.com/definitions/random-access/

Main memory – memory type



https://www.allaboutcircuits.com/technical-articles/introduction-to-dram-dynamic-random-access-memory/

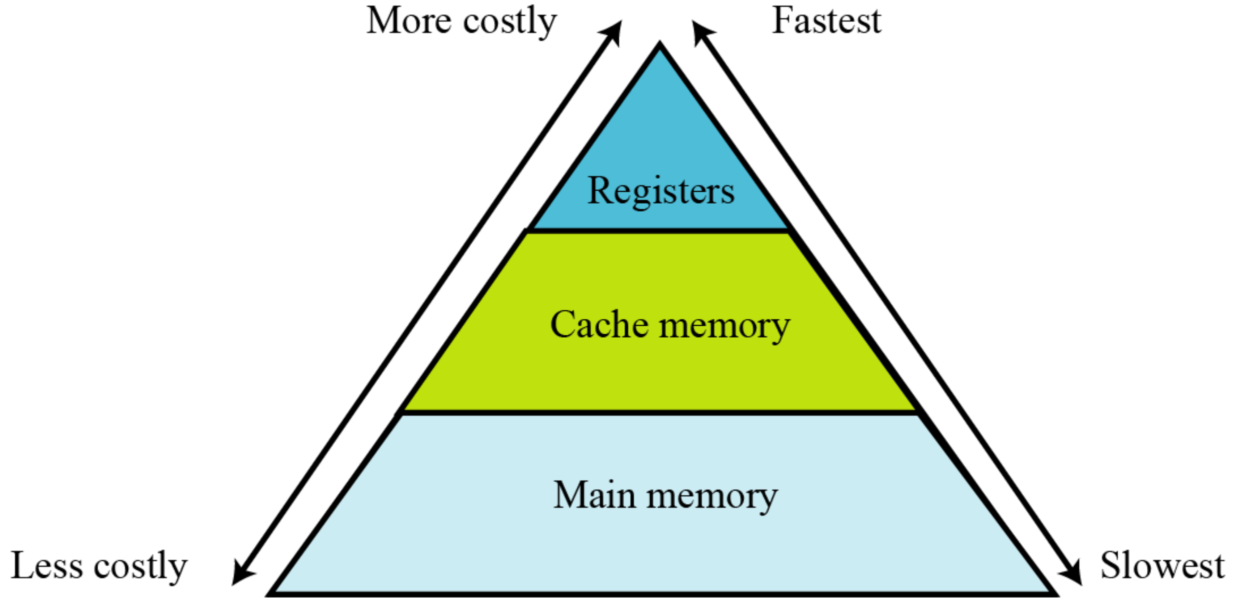
- Dynamic RAM (DRAM)
  - It uses capacitors, electrical devices that can store energy, for data storage. If a capacitor is charged, the state is 1; if it is discharged, the state is 0. Because a capacitor loses some of its charges with time, DRAM memory cells need to be refreshed periodically
  - DRAMs are slow but inexpensive
- Read-only memory (ROM) (唯讀記憶體)
  - The contents are written by the manufacturer, and the CPU can read from, but not write to ROM
  - Its advantage is that it is *nonvolatile (非揮發性)*—its contents are not lost if you turn off the computer
  - For example, some computers come with ROM that holds the *boot program (啟動程式)* that runs when we switch on the computer

Main memory – memory type

- Programmable read-only memory (PROM) (可程式規劃唯讀記憶體)
  - This type of memory is blank when the computer is shipped. The user of the computer, with some special equipment, can store programs on it
  - When programs are stored, it behaves like ROM and cannot be overwritten
- Erasable programmable read-only memory (EPROM) (可清除可程式化唯讀記憶體)
  - It can be programmed by the user, but can also be erased with a special device that applies ultraviolet light
  - To erase EPROM memory requires physical removal and reinstallation of the EPROM
- Electrically erasable programmable read-only memory (EEPROM) (電壓可清除可程式化唯讀記憶體)
  - EEPROM can be programmed and erased using electronic impulses without being removed from the computer

Main memory – Memory hierarchy

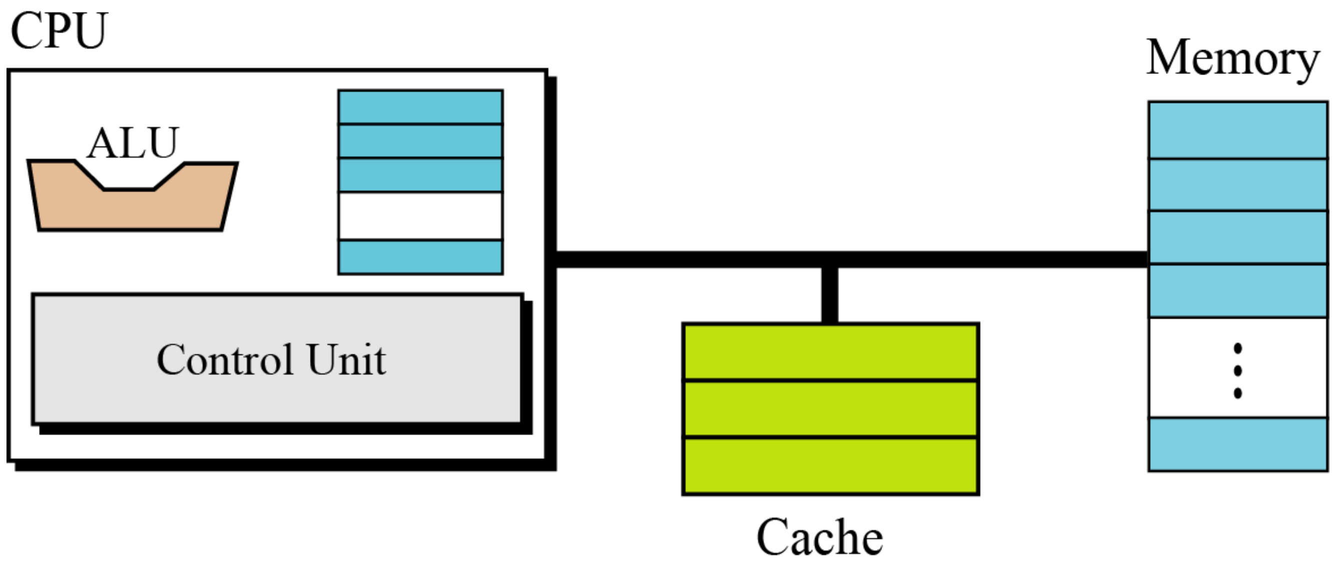
- Computer users need a lot of memory, especially memory that is very fast and inexpensive. The solution is *hierarchical* levels of memory
  - Use a very small amount of costly high-speed memory where speed is crucial. The registers inside the CPU are of this type
  - Use a moderate amount of medium-speed memory to store data that is accessed often. Cache memory is of this type
  - Use a large amount of low-speed memory for data that is accessed less often. The main memory is of this type



Main memory – Cache memory (快取記憶體)

- Cache memory is faster than main memory but slower than the registers
- Cache memory at any time contains a copy of a portion of main memory
- When the CPU needs to access a word in main memory

1. The CPU checks the cache
2. If the word is there, it copies the word. If not, the CPU accesses the main memory and copies a block of memory starting with the desired word into the cache
3. The CPU accesses the cache and copies the word



Input/output subsystem

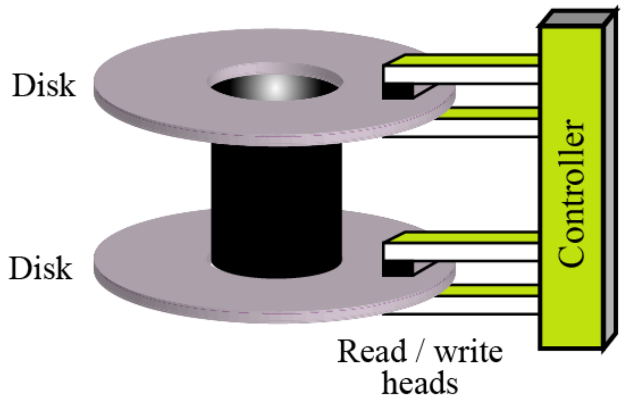
- The third major subsystem in a computer is the collection of devices referred to as the input/output (I/O) subsystem
  - This subsystem allows a computer to communicate with the outside world, and to store programs and data even when the power is off
- Non-storage devices
  - Non-storage devices allow the CPU/memory to communicate with the outside world, but they cannot store information
  - Keyboard, monitor, mice and joysticks
  - Printer
    - It is an output device that creates a permanent record and is a nonstorage device because the printed material cannot be directly entered into a computer again unless someone scans it

Input/output subsystem - Storage devices

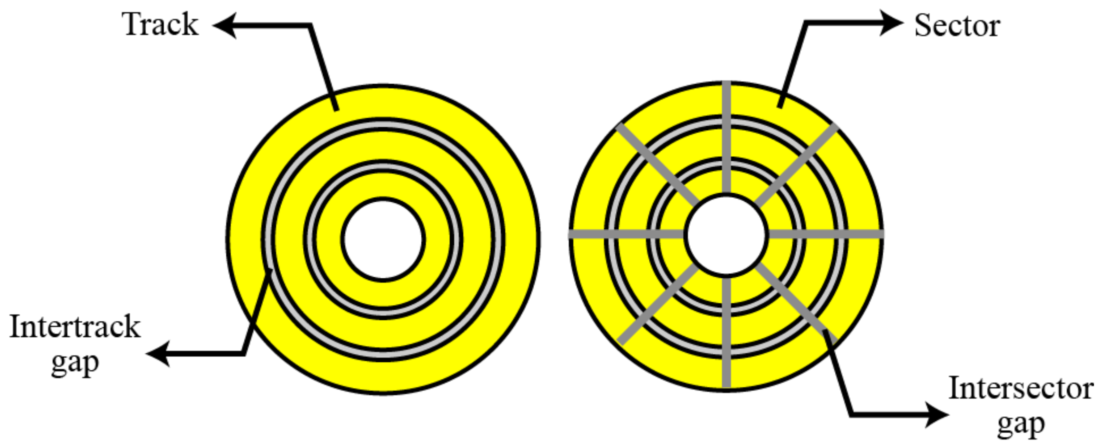
- Storage devices, although classified as I/O devices, can store large amounts of information to be retrieved at a later time
  - Cheaper than main memory, and their contents are nonvolatile
  - They are sometimes referred to as *auxiliary storage devices*
- *Magnetic storage devices (磁性儲存設備)* use magnetization to store bits of data. If a location is magnetized, it represents 1, if not magnetized, it represents 0
- *Optical storage devices (光學儲存設備)* use laser light to store and retrieve data. The use of optical storage technology followed the invention of the compact disk (CD) used to store audio information

Input/output subsystem - Magnetic storage devices

- Magnetic disks (磁碟)
  - It consists of disks stacked on top of each other and are coated with a thin magnetic film
  - Information is stored on and retrieved from the surface of the disk using a read/write head
  - Surface organization
    - To organize data stored on the disk, each surface is divided into *tracks (磁軌)*, and each track is divided into *sectors (磁區)*. The tracks are separated by an *intertrack gap (磁軌間隙)*, and the sectors are separated by an *intersector gap (磁區間隙)*



a. Disk drive



b. Tracks and Sectors

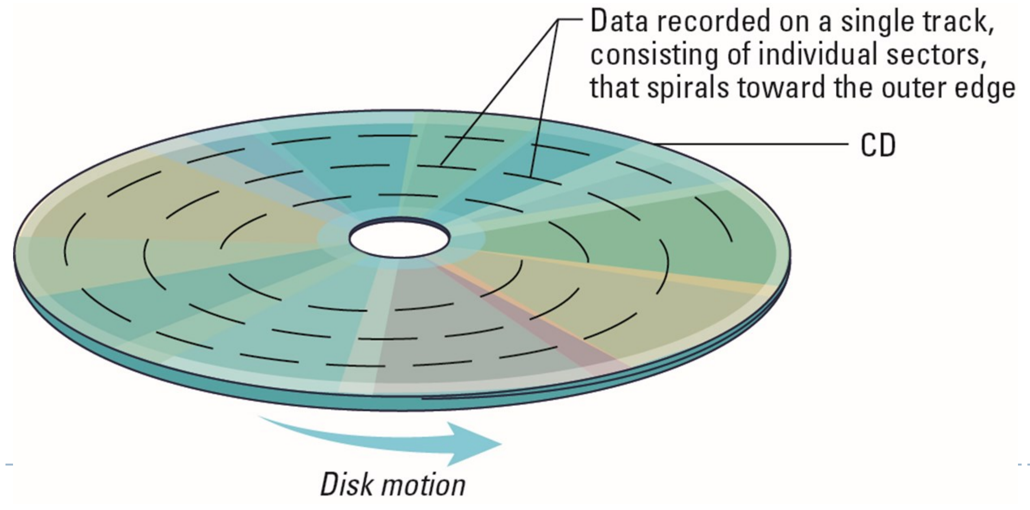
Input/output subsystem - Magnetic storage devices

- Magnetic disks
  - Data access
    - A magnetic disk is considered a random access device. However, the smallest storage area that can be accessed at one time is a sector
  - Performance
    - The *rotational speed (轉速)* defines how fast the disk is spinning
    - The *seek time (搜尋時間)* defines the time to move the read/write head to the desired track
    - The *transfer time (轉移時間)* defines the time to move data from the disk to the CPU/memory

https://en.wikipedia.org/wiki/Magnetic_storage#/media/File:Laptop-hard-drive-exposed.jpg

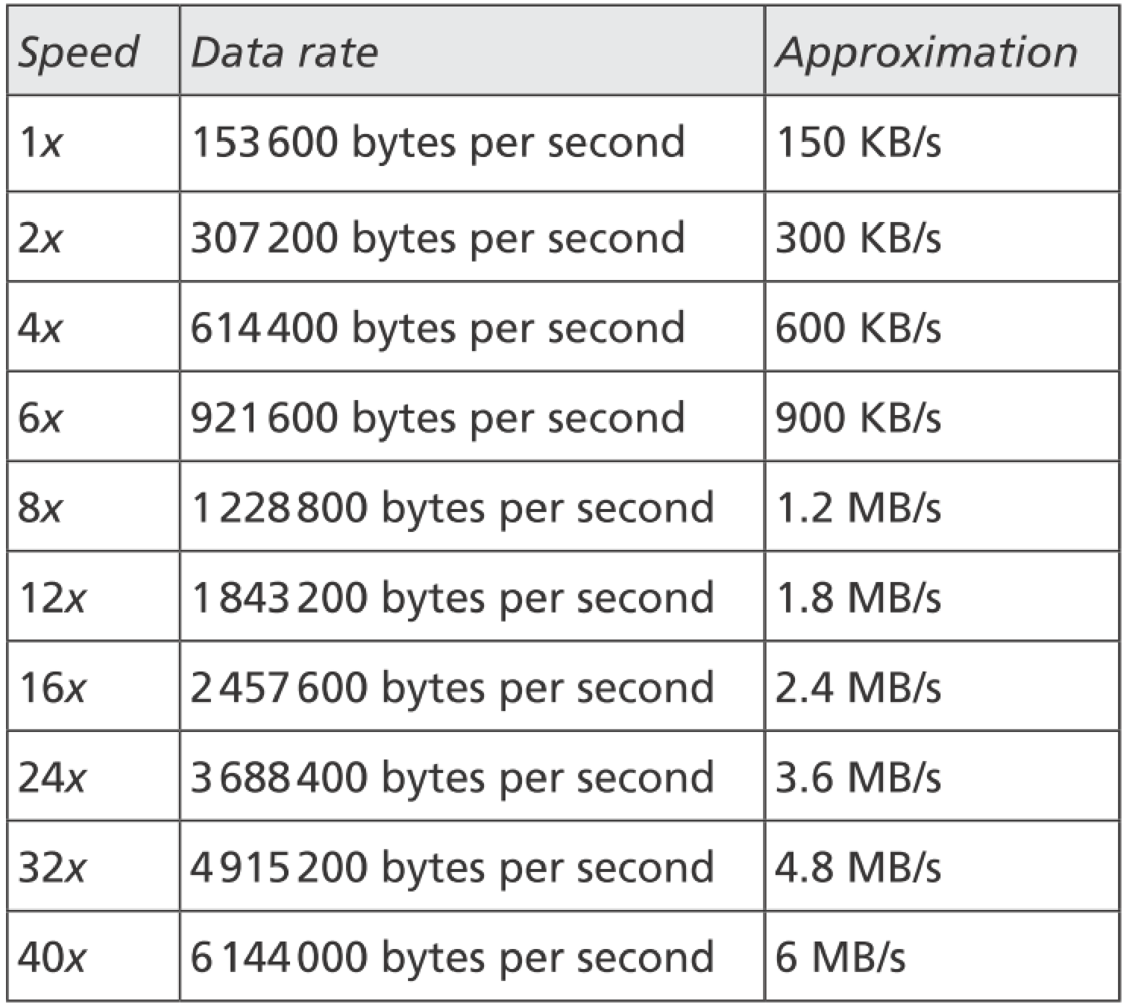
Input/output subsystem - Optical storage devices

- Compact disk read-only memory (CD-ROM) (唯獨光碟)
  - A CD-ROM drive is more robust and checks for errors compared with audio CD
  - They are 12 centimeters (approximately 5 inches) in diameter and consist of reflective material covered with a clear protective coating
  - Reading
    - Information is recorded on them by creating variations in their reflective surfaces. This information can then be retrieved by means of a laser that detects pits (0) and lands (1) on the reflective surface of the CD as it spins
  - Three steps are used to create a large number of CD-ROMs


Data recorded on a single track, consisting of individual sectors, that spirals toward the outer edge
CD
Disk motion

Input/output subsystem - Optical storage devices

- CD-ROM
  - Format
    - A frame made up of 24 bytes for the user data, 8 bytes for error correction
    - A sector made up of 98 frames (2,352 bytes)
    - CDs have capacities in the range of 600 to 700MB
  - Speed
    - CD-ROM drives come in different speeds. Single speed is referred to as 1x, double speed 2x, and so on. If the drive is single speed, it can read up to 153,600 bytes per second



Speed	*Data rate*	*Approximation*
1*x*	153 600 bytes per second	150 KB/s
2*x*	307 200 bytes per second	300 KB/s
4*x*	614 400 bytes per second	600 KB/s
6*x*	921 600 bytes per second	900 KB/s
8*x*	1 228 800 bytes per second	1.2 MB/s
12*x*	1 843 200 bytes per second	1.8 MB/s
16*x*	2 457 600 bytes per second	2.4 MB/s
24*x*	3 688 400 bytes per second	3.6 MB/s
32*x*	4 915 200 bytes per second	4.8 MB/s
40*x*	6 144 000 bytes per second	6 MB/s

Input/output subsystem - Optical storage devices

- Compact disk recordable (CD-R) (可燒錄光碟)
  - It allows users to create one or more disks without going through the expense involved in creating a large number of CD-ROMs.
  - It is particularly useful for making backups. You can write once to CD-R disks, but they can be read many times. It is sometimes called *write once, read many (WORM)*
  - Reading
    - CD-Rs can be read by a CD-ROM or a CD-R drive
  - Format and speed
    - The format, capacity, and speed of CD-Rs are the same as CD-ROMs

Input/output subsystem - Optical storage devices

- Compact disk rewritable (CD-RW) (可複寫光碟)
  - CD-Rs can be written only once. To overwrite previous materials, new technology is used. It is sometimes called an erasable optical disk
  - Reading
    - The drive uses the same type of low-power laser beam as CD-ROM and CD-R to detect pits (0) and lands (1)
  - Erasing
    - The drive uses a medium-power laser beam to change pits to lands
  - Format and speed
    - The format, capacity, and speed of CD-RWs are the same as CD-ROMs

Input/output subsystem - Optical storage devices

- Digital versatile disk (DVD)
  - The capacity of a CD-ROM (650 MB) is insufficient to store video information
  - The beam is a red laser instead of infrared
- BDs (Blu-ray Disks)
  - Uses a laser in the blue-violet spectrum of light (instead of red), is able to focus its laser beam with very fine precision
  - As a result, single-layer BDs (Blu-ray Disks) provides over five times the capacity of a DVD. Newer multilayer BD formats can reach capacities in the 100GB range

Feature	*Capacity*
Single-sided, single-layer	4.7 GB
Single-sided, dual-layer	8.5 GB
Double-sided, single-layer	9.4 GB
Double-sided, dual-layer	17 GB

Input/output subsystem – Flash drives

- Flash drives
  - Bits are stored by sending electronic signals directly to the storage medium where they cause electrons to be trapped in tiny chambers that are non-volatile
  - It uses *electronic circuitry* for reading and writing, therefore, is faster than magnetic or optic technology that is based on physical motion
  - It is not sensitive to physical shock and therefore is suited for the portable device
  - However, the vulnerability of their tiny storage chambers dictates that they are not as reliable as optical disks for truly long-term applications

https://en.wikipedia.org/wiki/USB_flash_drive#/media/File:SanDisk-Cruzer-USB-4GB-ThumbDrive.jpg

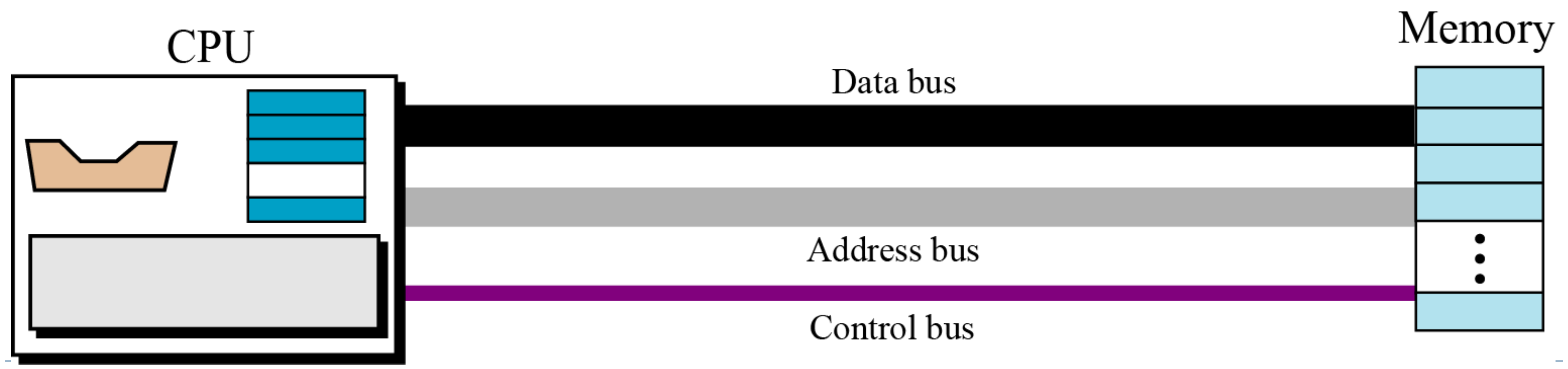
Input/output subsystem – Flash drives

- SSDs (solid-state drives)
  - They are explicitly designed to take the place of magnetic hard disks and are larger flash memory devices
  - SSDs remain more expensive than hard disks of comparable size, and thus, are still considered a high-end option
  - SSD sectors suffer from the more limited lifetime of all flash memory technologies
- SD (Secure Digital) memory cards
  - Another application of flash technology is found in SD cards
  - SDHC (High Capacity) memory cards can provide up to 32 GBs and the next generation SDXC (Extended Capacity) memory cards can exceed a TB

https://en.wikipedia.org/wiki/SD_card#/media/File:SD_Cards.svg

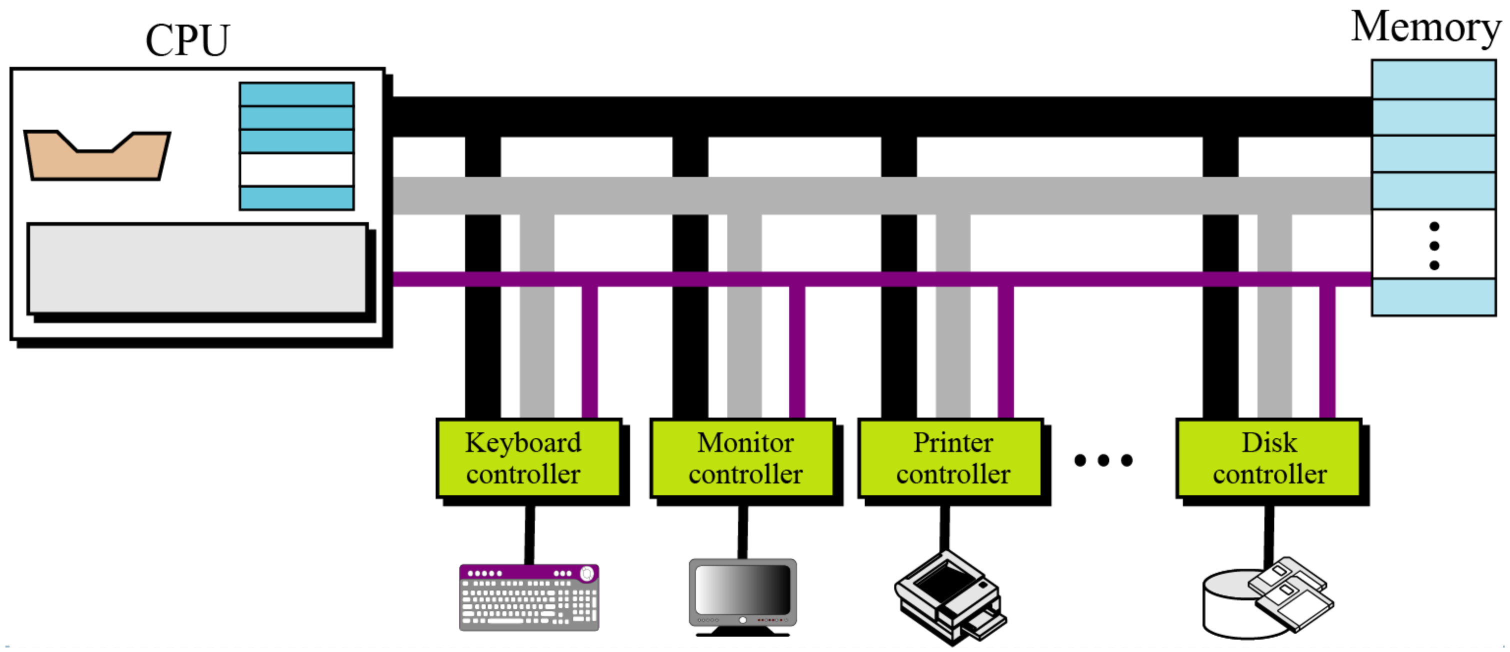
Subsystem interconnection

- We explore how the previous three subsystems are interconnected
- Connecting CPU and memory
  - The *data bus (資料匯流排)* is made of several connections, each carrying 1 bit at a time. The number of connections depends on the size of the word used by the computer
  - The *address bus (位址匯流排)* allows access to a particular word in memory. The number of connections in the address bus depends on the address space of the memory
  - The *control bus (控制匯流排)* carries communication between the CPU and memory. For example, there must be a code, sent from the CPU to memory, to specify a read or write operation



Subsystem interconnection

- Connecting I/O devices
  - Since many I/O devices are magnetic or optical devices, whereas the CPU and memory are electronic devices they can not be connected directly to the previous bus
  - They also operate at a much slower speed than the CPU/memory
  - Input/output devices are therefore attached to the buses through *input/output controllers*

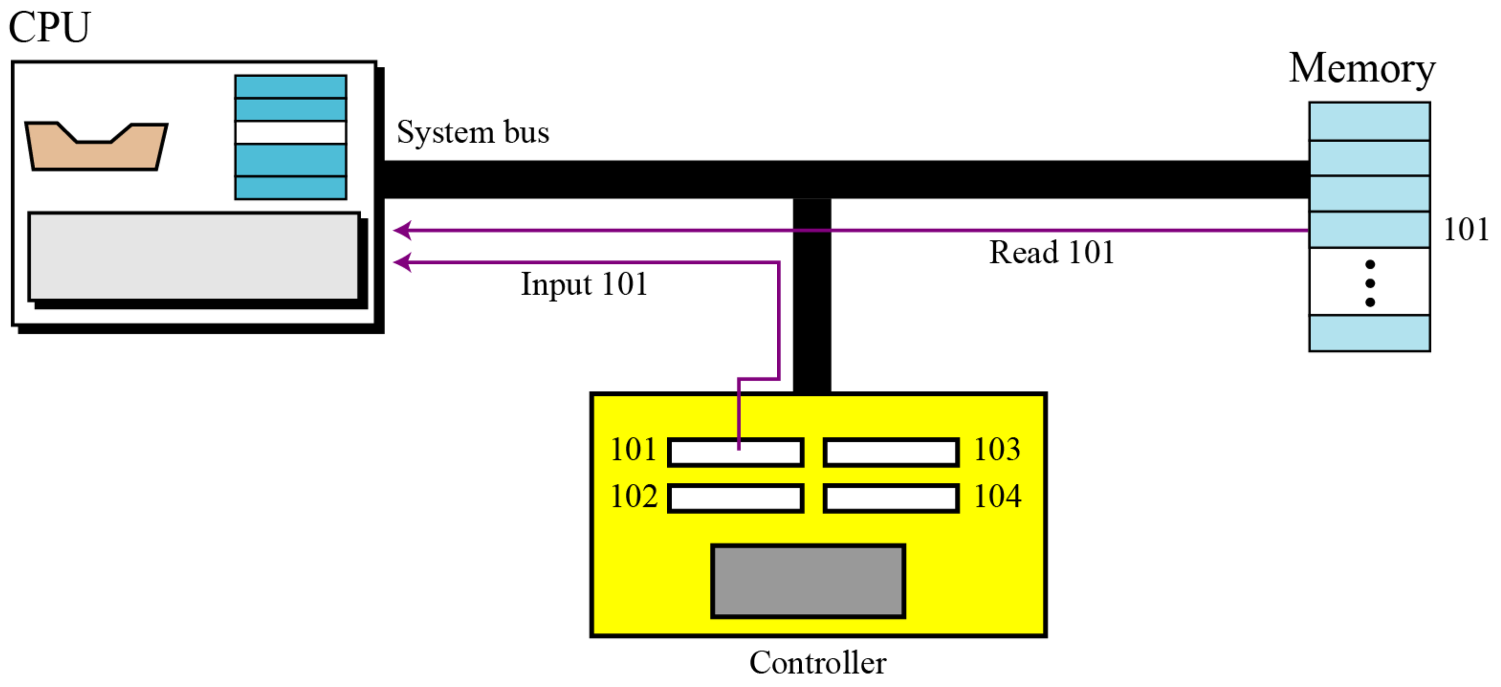


Subsystem interconnection

- Addressing input/output devices
  - The CPU usually uses the same bus to read or write data to main memory and I/O device
    - If the instruction refers to a word in main memory, data transfer is between main memory and the CPU. If it identifies an I/O device, data transfer is between the I/O device and the CPU
  - There are two methods for handling the addressing of I/O devices: *isolated I/O (獨立式)* and *memory-mapped I/O (記憶體對映)*

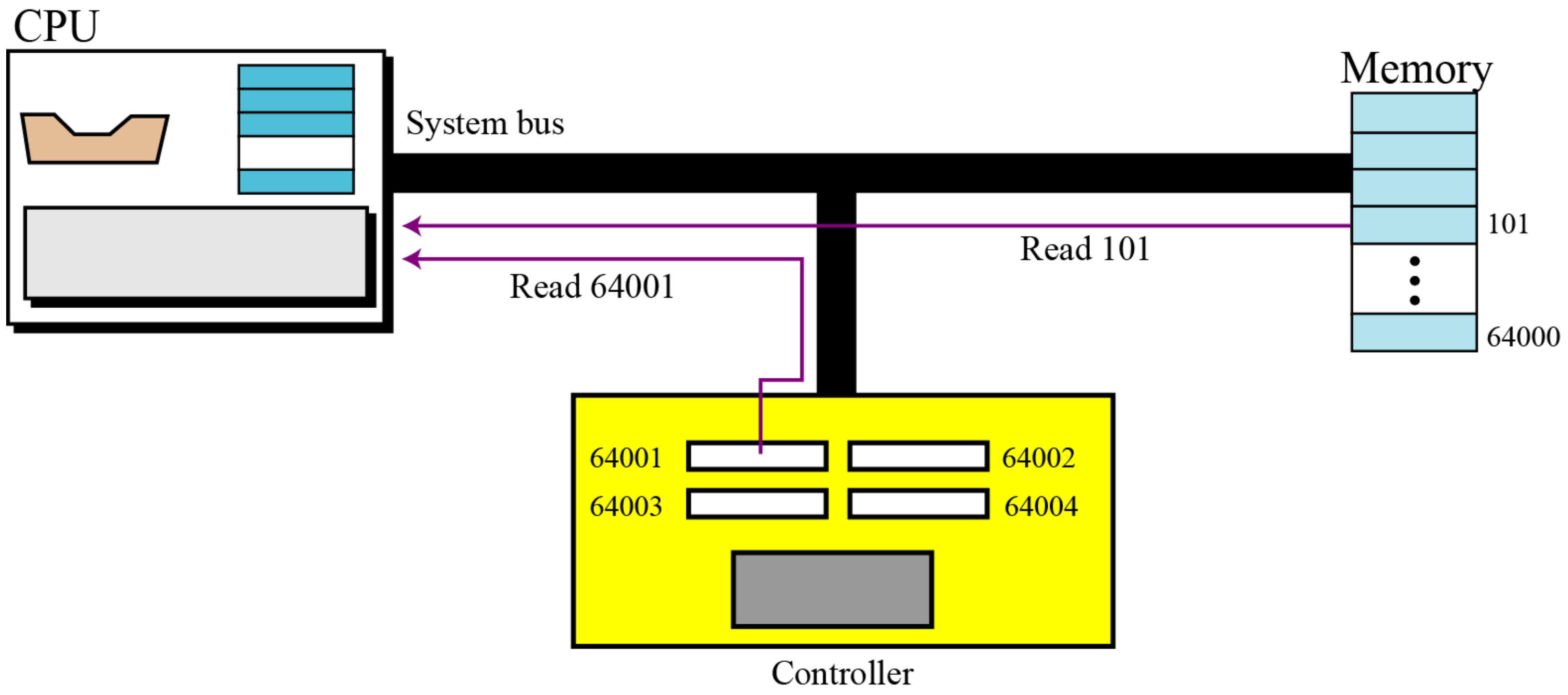
Subsystem interconnection - Isolated I/O

- The instructions used to read/write memory are totally different than the instructions used to read/write I/O devices
  - Each I/O device has its own address
  - The I/O addresses can thus overlap with memory addresses without any ambiguity



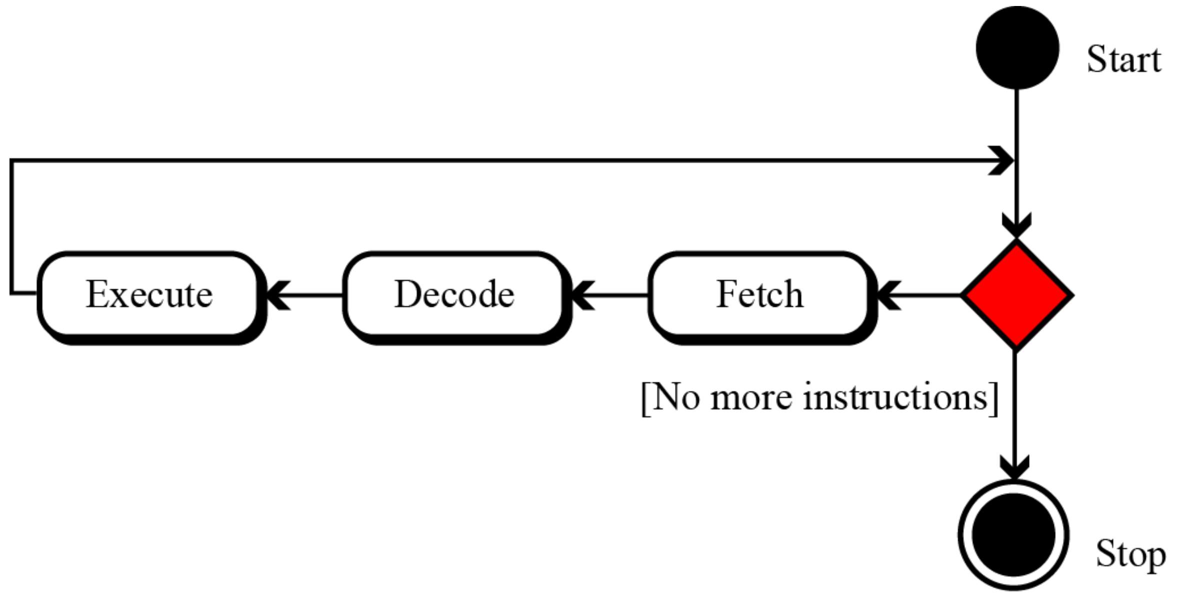
Subsystem interconnection - Memory-mapped I/O

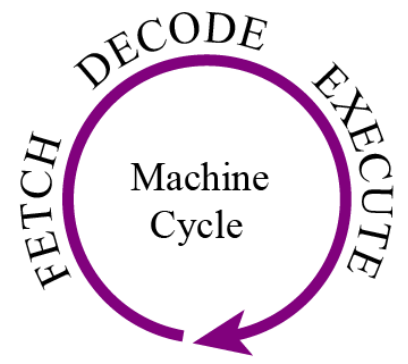
- The CPU treats each register in the I/O controller as a word in memory
  - The advantage of the memory-mapped configuration is a smaller number of instructions
  - The disadvantage is that part of the memory address space is allocated to registers in I/O controllers



Program Execution - Machine cycle

- The CPU uses repeating machine cycles to execute instructions in the program, one by one, from beginning to end
  - A simplified cycle can consist of three phases: *fetch (擷取)*, *decode (解碼)*, and *execute (執行)*





Program Execution - Machine cycle

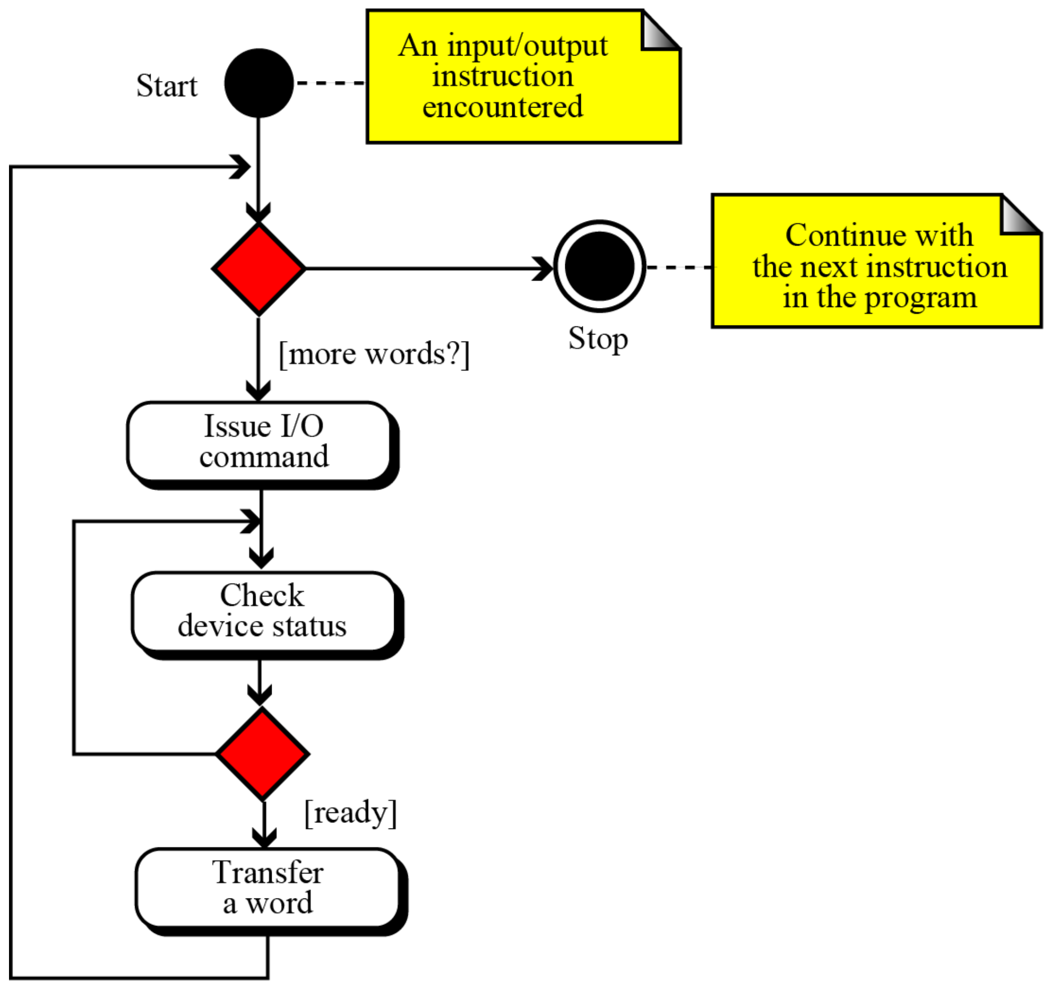
- Fetch
  - In this phase, the control unit copies the next instruction into the instruction register
  - The address of the instruction to be copied is held in the program counter register
  - After copying, the program counter is incremented to refer to the next instruction
- Decode
  - When the instruction is in the instruction register, it is decoded by the control unit. The result of this step is the binary code for some operation that the system will perform
- Execute
  - After the instruction is decoded, the control unit sends the task order to a component in the CPU

Program Execution - Input/output operation

- Because I/O devices operate at much slower speeds than the CPU, the operation of the CPU must be somehow synchronized with the I/O devices
  - Three methods have been devised for this synchronization: *programmed I/O (程式化)*, *interrupt-driven I/O (中斷驅動)*, and *direct memory access (DMA) (直接記憶體存取)*

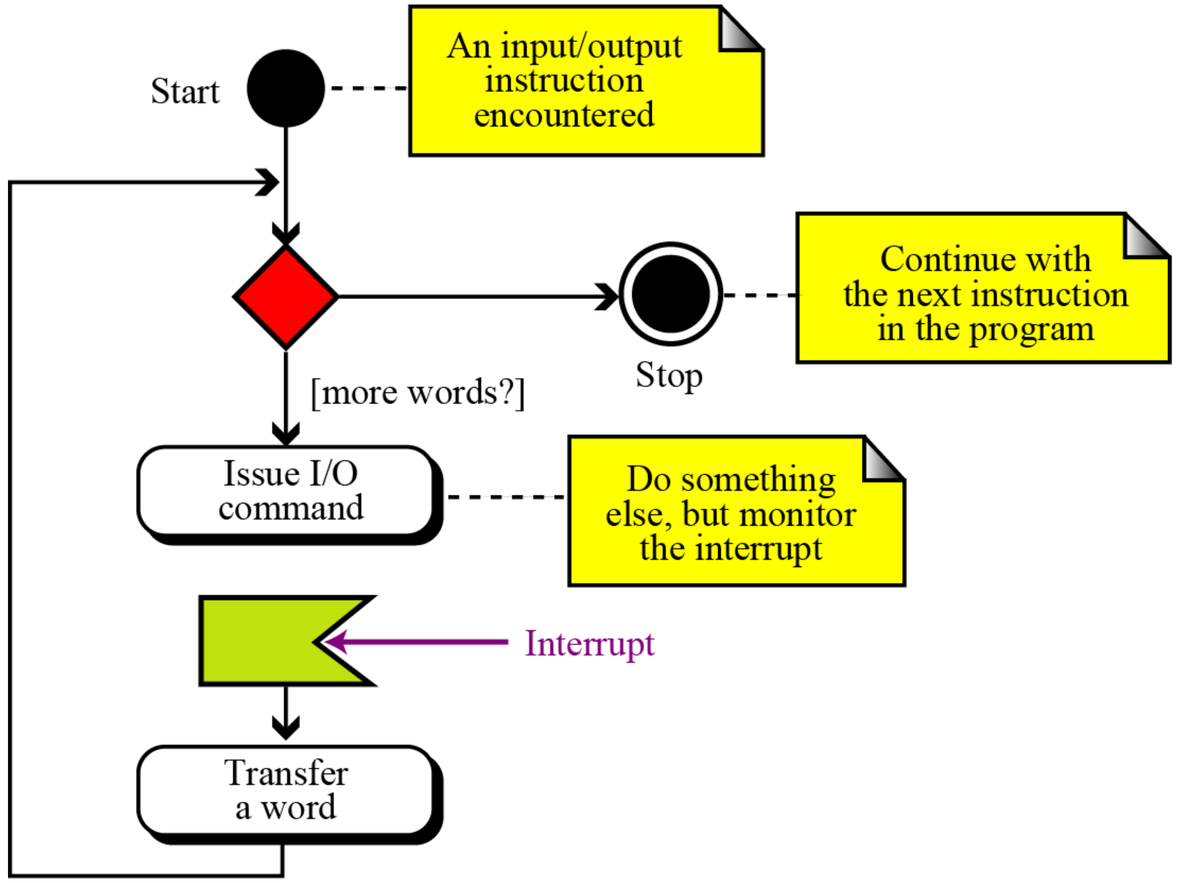
Program Execution - Programmed I/O

- When the CPU encounters an I/O instruction, it does nothing else until the data transfer is complete
  - The big issue here is that CPU time is wasted by checking the status of the I/O device for each unit of data to be transferred


Start
An input/output instruction encountered
Stop
Continue with the next instruction in the program
[more words?]
Issue I/O command
Check device status
[ready]
Transfer a word

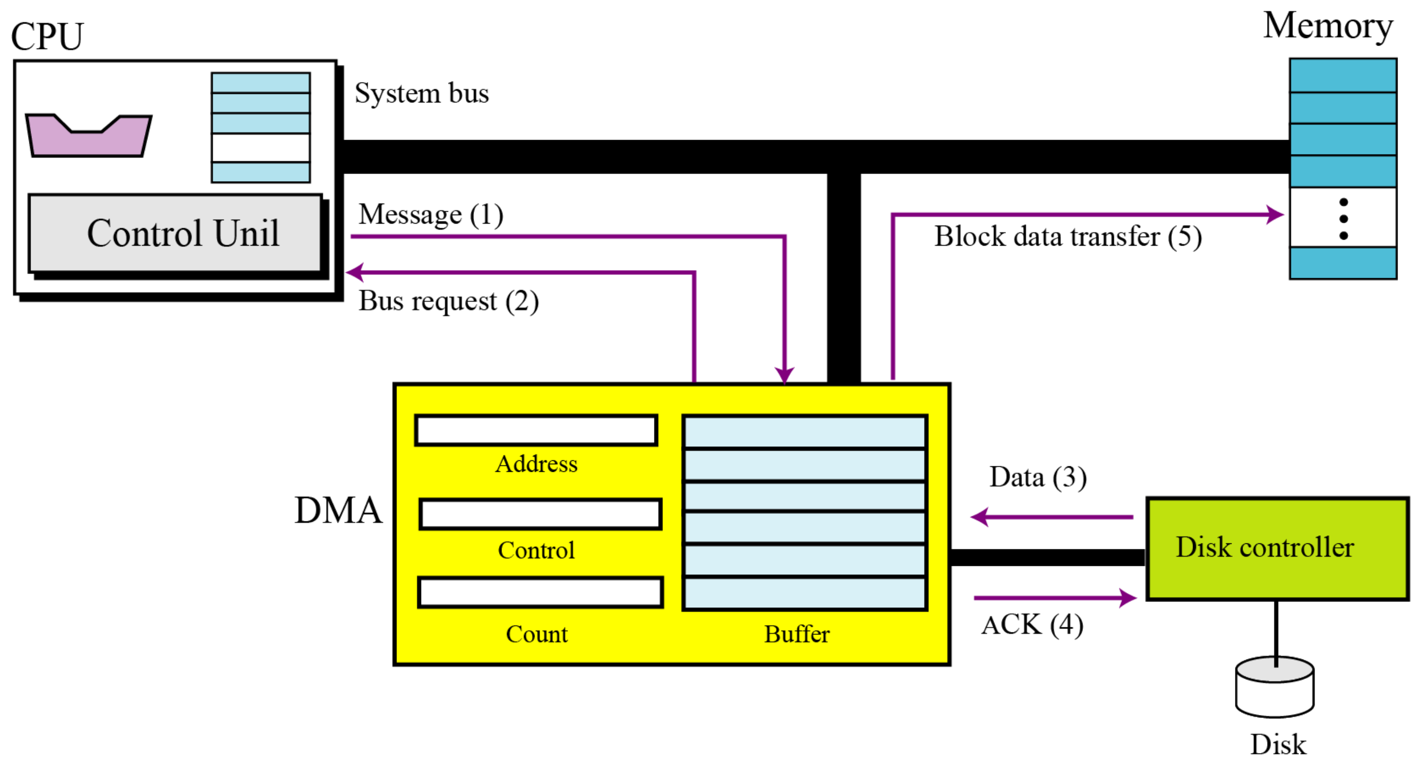
Program Execution - Interrupt-driven I/O

- The CPU informs the I/O device that a transfer is going to happen
  - The I/O device informs (interrupts) the CPU when it is ready
  - In this method, CPU time is not wasted—the CPU can do something else while the slow I/O device is finishing a task


Start
An input/output instruction encountered
Continue with the next instruction in the program
Stop
[more words?]
Issue I/O command
Do something else, but monitor the interrupt
Interrupt
Transfer a word

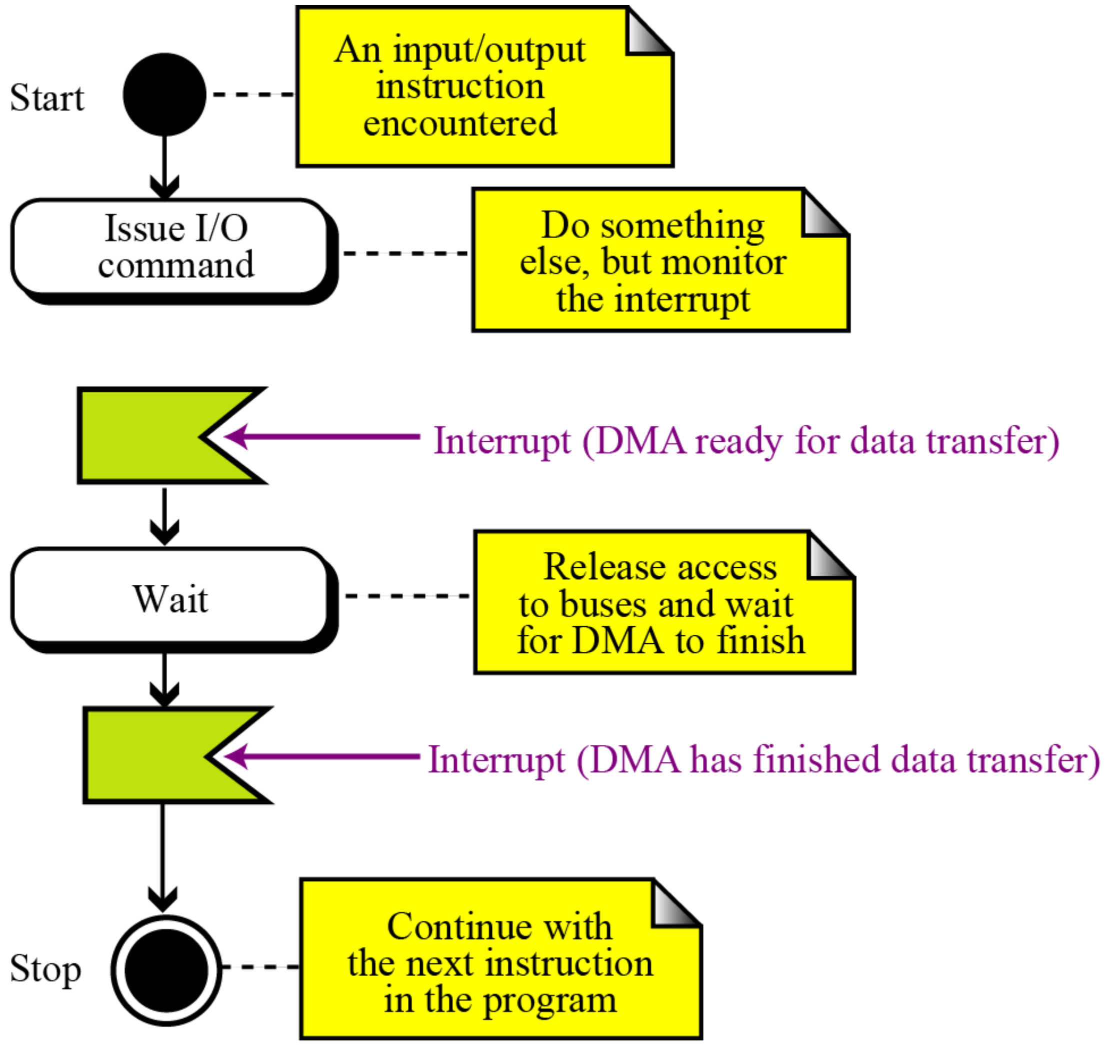
Program Execution - Direct memory access (DMA)

- This method transfers a large block of data between a high-speed I/O device and memory directly without passing it through the CPU
  - The DMA controller has registers to hold a block of data before and after memory transfer



Program Execution - Direct memory access (DMA)

- CPU sends a message to the DMA
  - The message contains the type of transfer (input or output), the beginning address of the memory location, and the number of bytes to be transferred
  - When ready to transfer data, the DMA controller informs the CPU that it needs to take control of the buses
  - The CPU is idle only during the data transfer between the DMA and memory, not while the device prepares the data


Start
An input/output instruction encountered
Issue I/O command
Do something else, but monitor the interrupt
Interrupt (DMA ready for data transfer)
Wait
Release access to buses and wait for DMA to finish
Interrupt (DMA has finished data transfer)
Stop
Continue with the next instruction in the program

Different Architectures - complex instruction set computer (CISC)

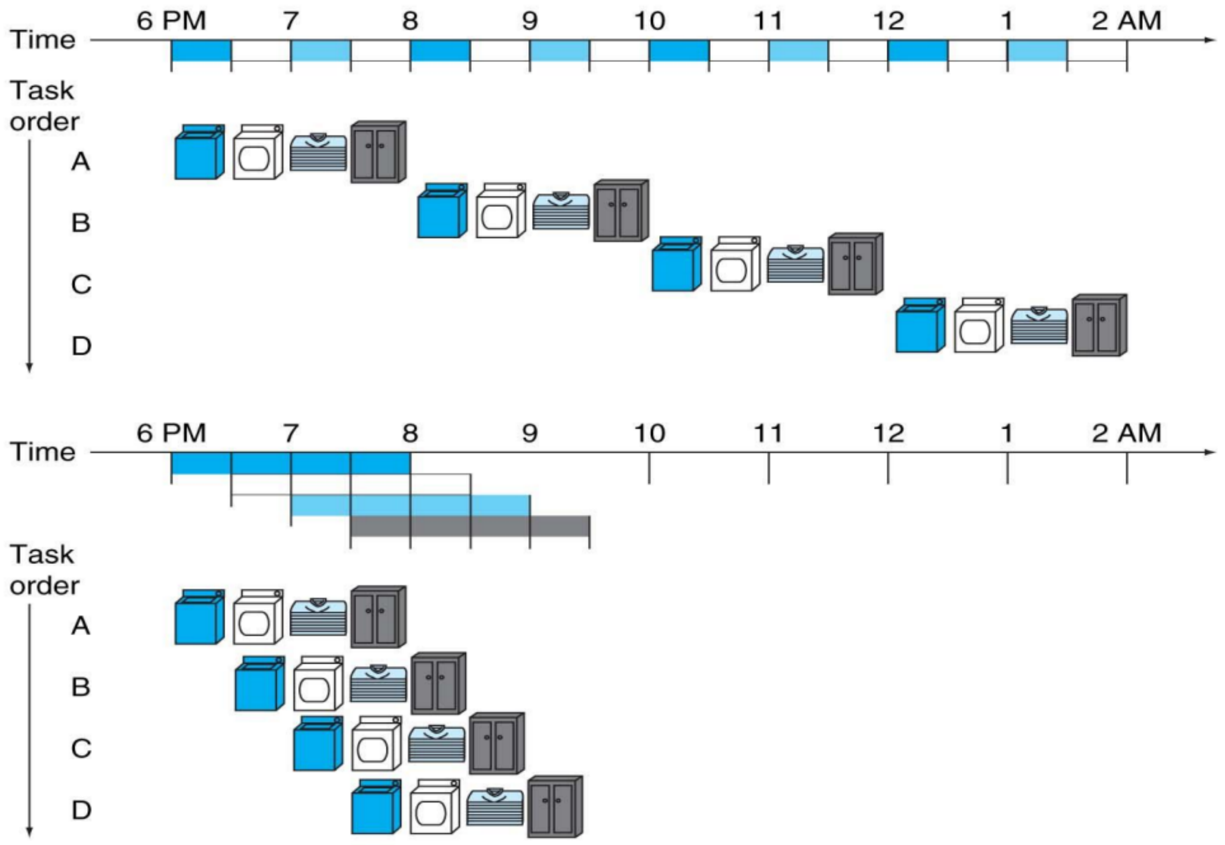
- The strategy behind CISC (複雜指令電腦) architectures is to have a large set of instructions
  - Programming CISC-based computers is easier than in other designs because there is a single instruction for both simple and complex tasks
- Programming is done on two levels
  - The CPU performs only simple operations, called *microoperations*
  - A complex instruction is transformed into a set of these simple operations and then executed by the CPU. This necessitates the addition of a special memory called *micromemory* that holds the set of operations for each complex instruction in the instruction set
  - The type of programming that uses microoperations is called *microprogramming*
- An example of CISC architecture can be seen in the Pentium series of processors developed by Intel

Reduced instruction set computer (RISC)

- The strategy behind RISC (精簡指令電腦) architecture is to have a small set of instructions that do a minimum number of simple operations
- Complex instructions are simulated using a subset of simple instructions
  - Programming in RISC is more difficult and time-consuming than in the other design because most of the complex instructions are simulated using simple instructions
  - This architecture uses less chip space due to a reduced instruction set
  - RISC processors can be designed more quickly than CISC processors due to their simple architecture
- An example of RISC architecture can be seen in the ARM

Pipelining (管線處理)

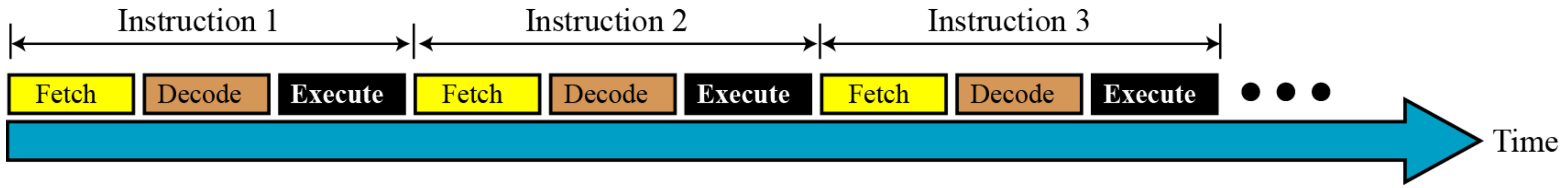
- Modern computers use a technique called *pipelining* to improve the throughput (the total number of instructions performed in each period of time)



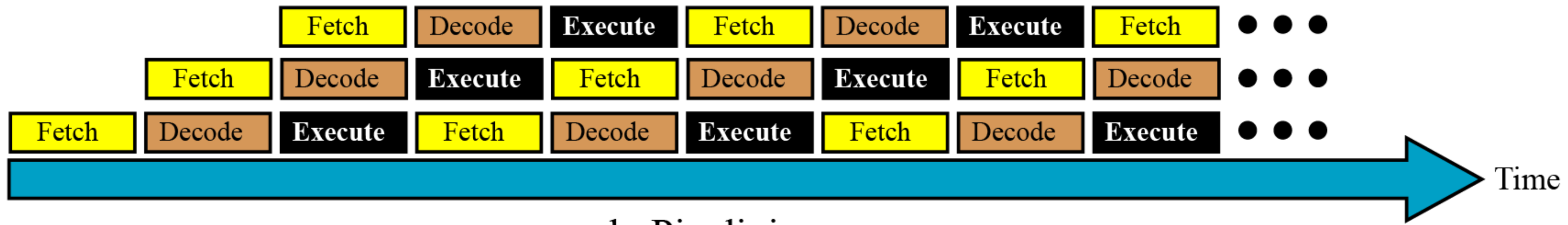
https://www.cpp.edu/~gsyoung/CS370/14Sp/pipeline_computations_kla%20Danny.pdf

Pipelining

- When the CPU is performing the decode phase of the first instruction, it can also perform the fetch phase of the second instruction
  - The first computer can perform on average 9 phases in the specific period of time, while the pipelined computer can perform 24 phases in the same period of time
  - If we assume that each phase uses the same amount of time, the throughput is increased by 266 percent



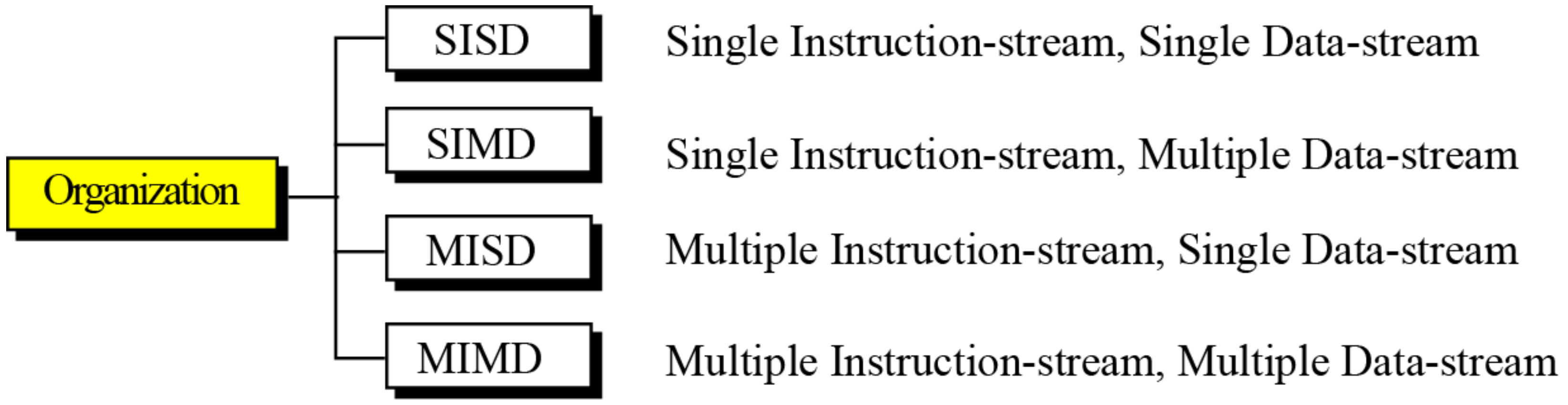
a. No pipelining



b. Pipelining

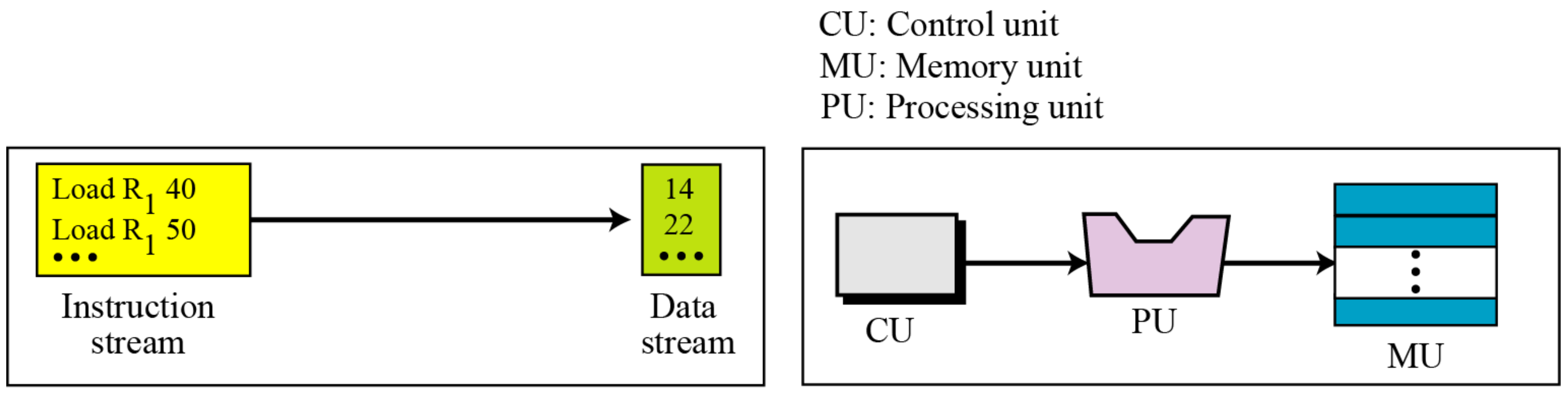
Parallel processing

- With the evolution in technology and the drop in the cost of computer hardware, today we can have a single computer with multiple control units, multiple arithmetic logic units and multiple memory units
  - The idea of *parallel processing* emerges. Like pipelining, parallel processing can improve throughput
  - Parallel processing may occur in the data stream, the instruction stream, or both


Organization
SISD
Single Instruction-stream, Single Data-stream
SIMD
Single Instruction-stream, Multiple Data-stream
MISD
Multiple Instruction-stream, Single Data-stream
MIMD
Multiple Instruction-stream, Multiple Data-stream

Single instruction-stream, single data-stream (SISD)

- It represents a computer that has one control unit, one arithmetic logic unit, and one memory units
  - The instructions are executed sequentially and each instruction may access one or more data items in the data stream. Our simple computer introduced earlier in the chapter is an example of SISD organization

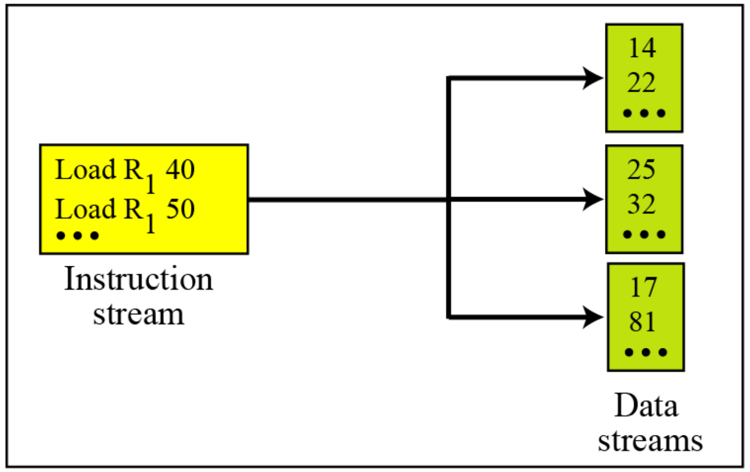


a. Concept b. Configuration

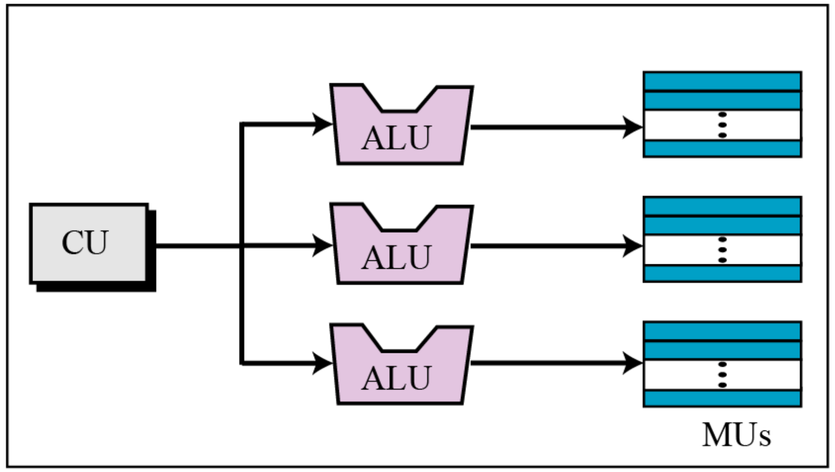
Single instruction-stream, multiple data-stream (SIMD)

- It represents a computer that has one control unit, multiple processing units, and multiple memory units
  - All processor units receive the same instruction from the control unit, but operate on different items of data. An array processor that simultaneously operates on an array of data belongs to this category

ALU: Arithmetic Logic Unit
CU: Control Unit
MU: Memory Unit



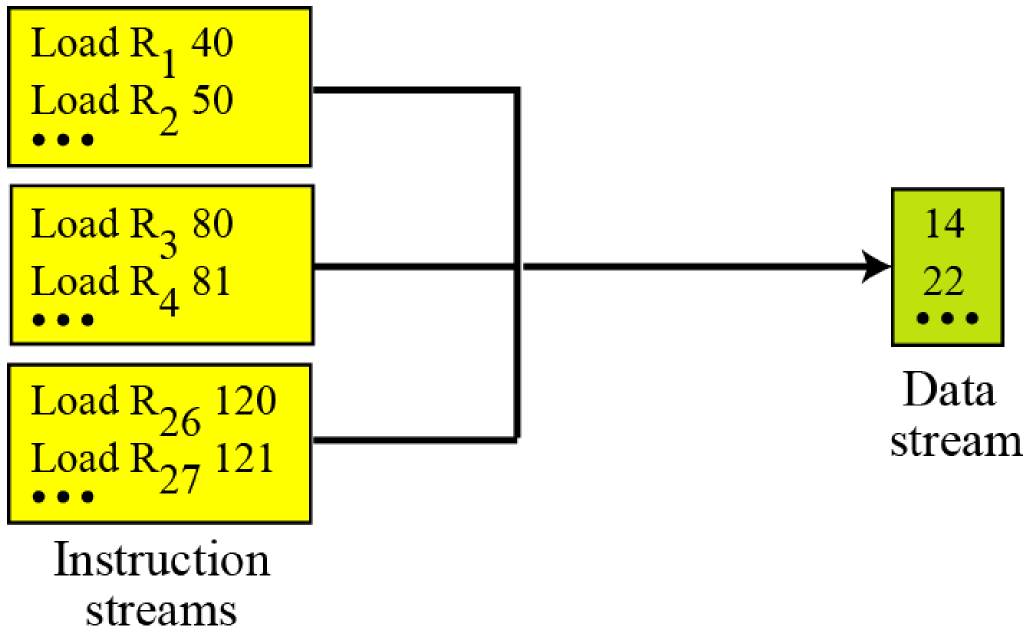
a. Concept



b. Implementation

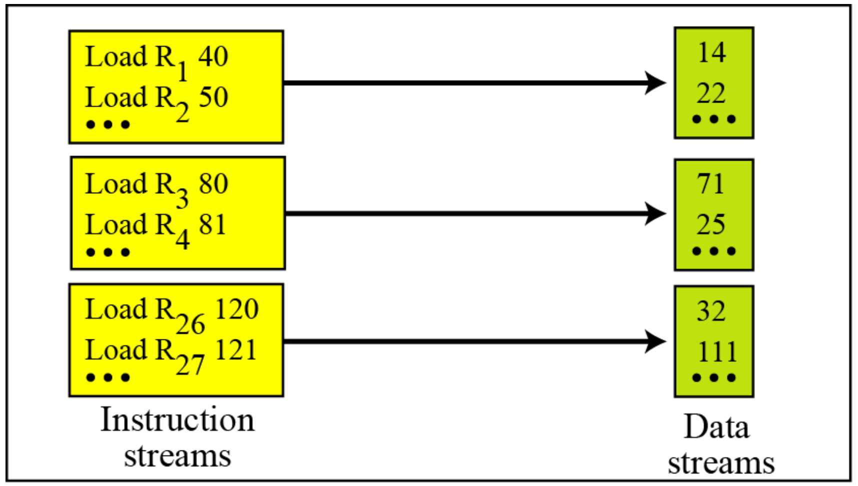
Multiple instruction-stream, single data-stream (MISD)

- The architecture is one in which several instructions belonging to several instruction streams simultaneously operate on the same data stream
  - It has never been implemented

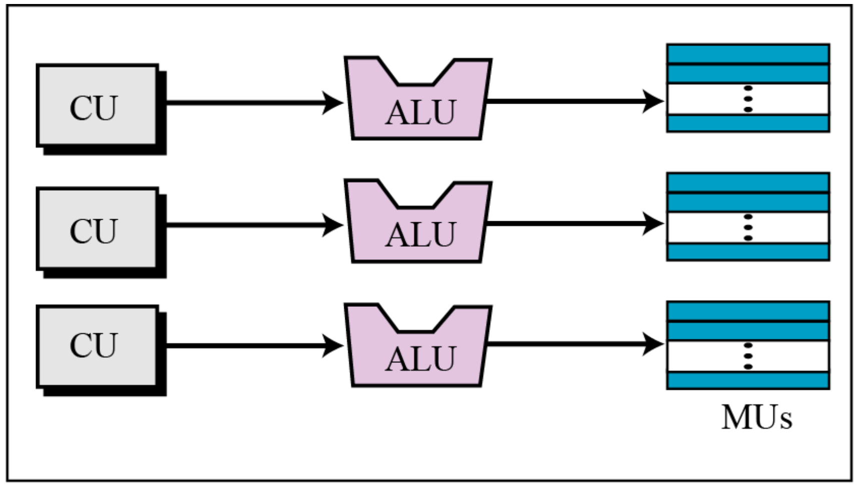


Multiple instruction-stream, multiple data-stream (MIMD)

- The architecture is one in which several instructions belonging to several instruction streams simultaneously operate on several data streams
  - In this architecture several tasks can be performed simultaneously. The architecture can use a single shared memory or multiple memory sections



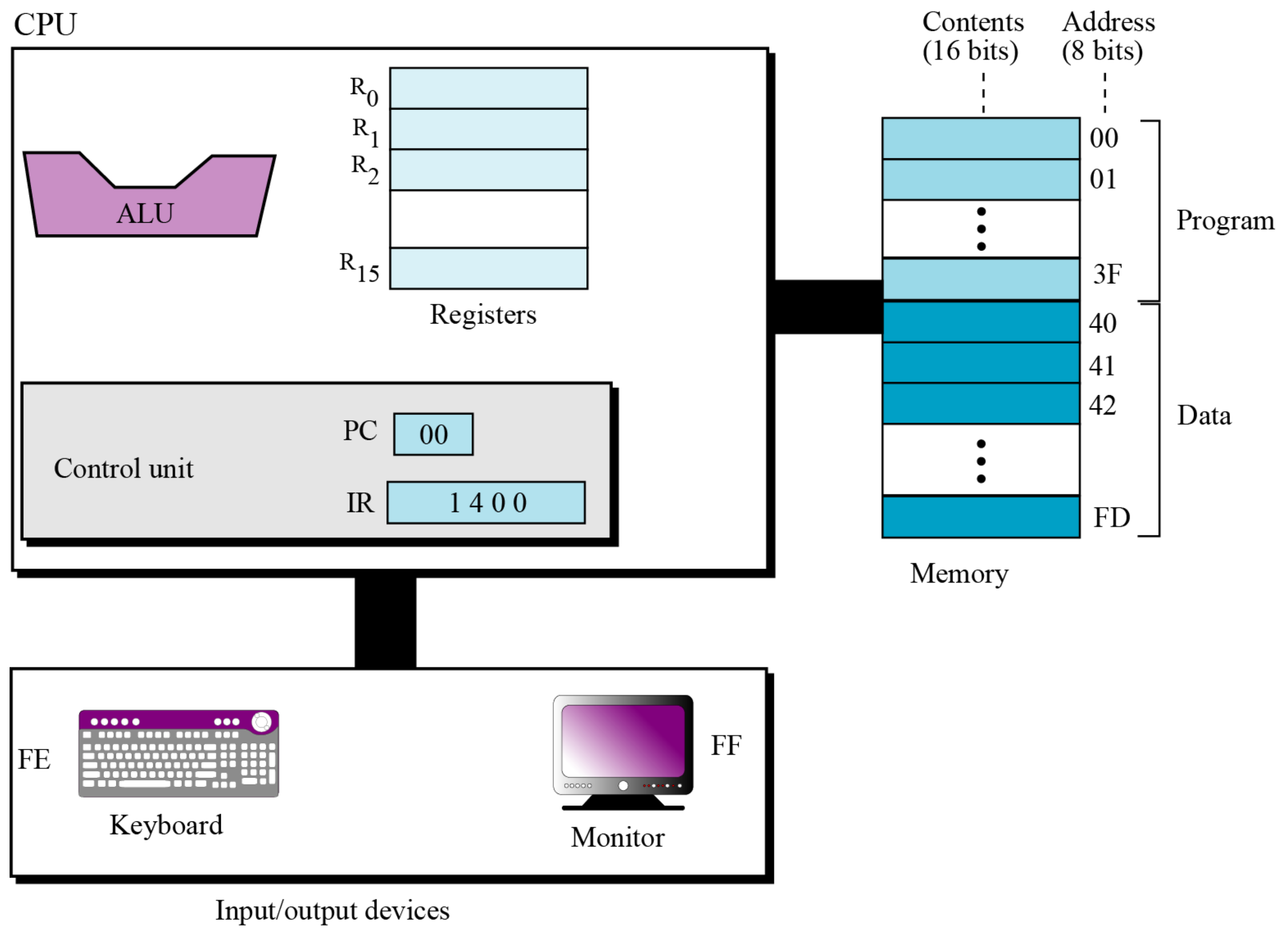
a. Concept



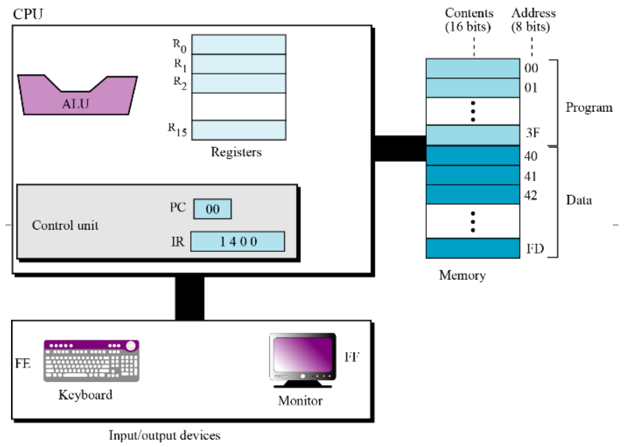
b. Implementation

A simple computer

- To explain the architecture of computers as well as their instruction processing, we introduce a simple (unrealistic) computer
  - Our simple computer has three components: CPU, memory, and an input/output subsystem

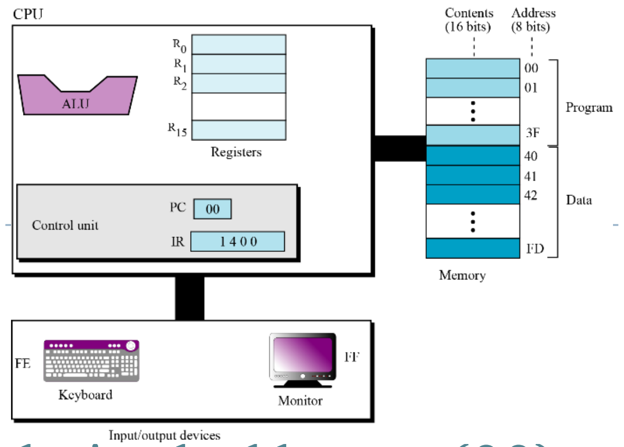


A simple computer



- CPU
  - Data registers
    - There are sixteen 16-bit data registers with hexadecimal addresses $(0, 1, 2, \ldots, F)_{16}$, but we refer to them as R_0 to R_{15}
  - Control unit
    - The control unit has the circuitry to control the operations of the ALU, access to memory, and access to the I/O subsystem. It has two dedicated registers
    - Program counter (PC): hold only eight bits, and keeps track of which instruction is to be executed next. The contents of the PC points to the address of the memory location of the main memory that holds the next program instruction. After each machine cycle the program counter is incremented by one to point to the next program instruction
    - Instruction register (IR): holds a 16-bit value which is the encoded instruction for the current cycle

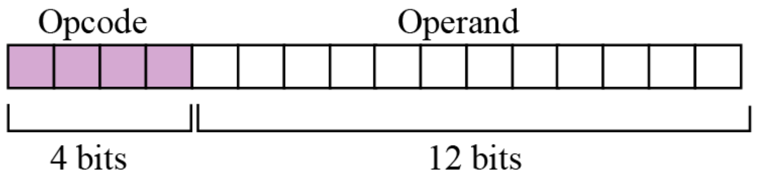
A simple computer



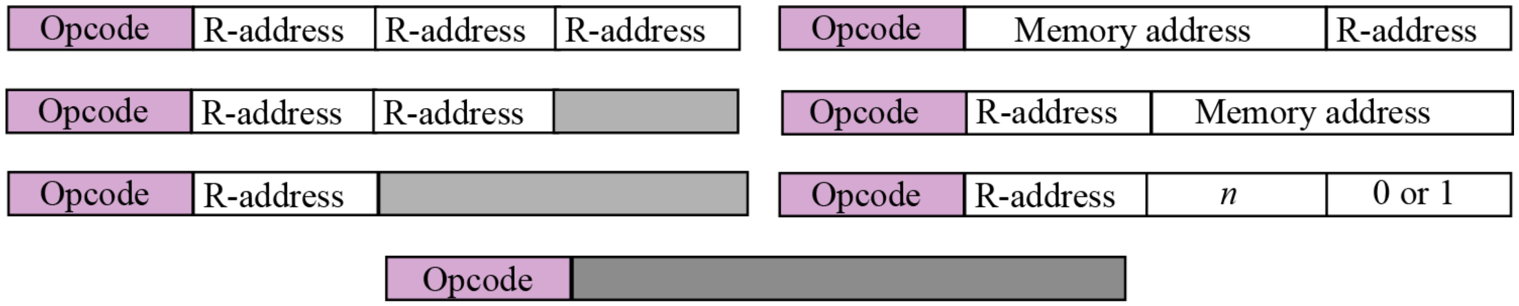
- Main memory
  - The main memory has 256 16-bit memory locations with hexadecimal addresses $(00)_{16}$ to $(FD)_{16}$
  - The main memory holds both data and program instruction
    - The first 64 locations ($(00)_{16}$ to $(3F)_{16}$) are dedicated to program instructions. Program instructions for any program are stored in consecutive memory locations
    - Memory locations $(40)_{16}$ to $(FD)_{16}$ are used for storing data
- Input/output subsystem
  - The subsystem consists of a keyboard and a monitor
  - These devices have memory-mapped addresses and we assume that the keyboard (as the input device) and monitor (as the only output device) acts like memory locations with addresses $(FE)_{16}$ and $(FF)_{16}$ respectively

Instruction set (指令集)

- Our simple computer is capable of having a set of sixteen instructions, although we are using only fourteen of these instructions
  - Each instruction consists of the operation code (*opcode*)(*運算碼*) and the *operands* and the opcode specifies the type of operation to be performed
  - Each instruction consists of sixteen bits divided into four 4-bit fields. The leftmost field contains the opcode and the other three fields contains the operand or address of operands



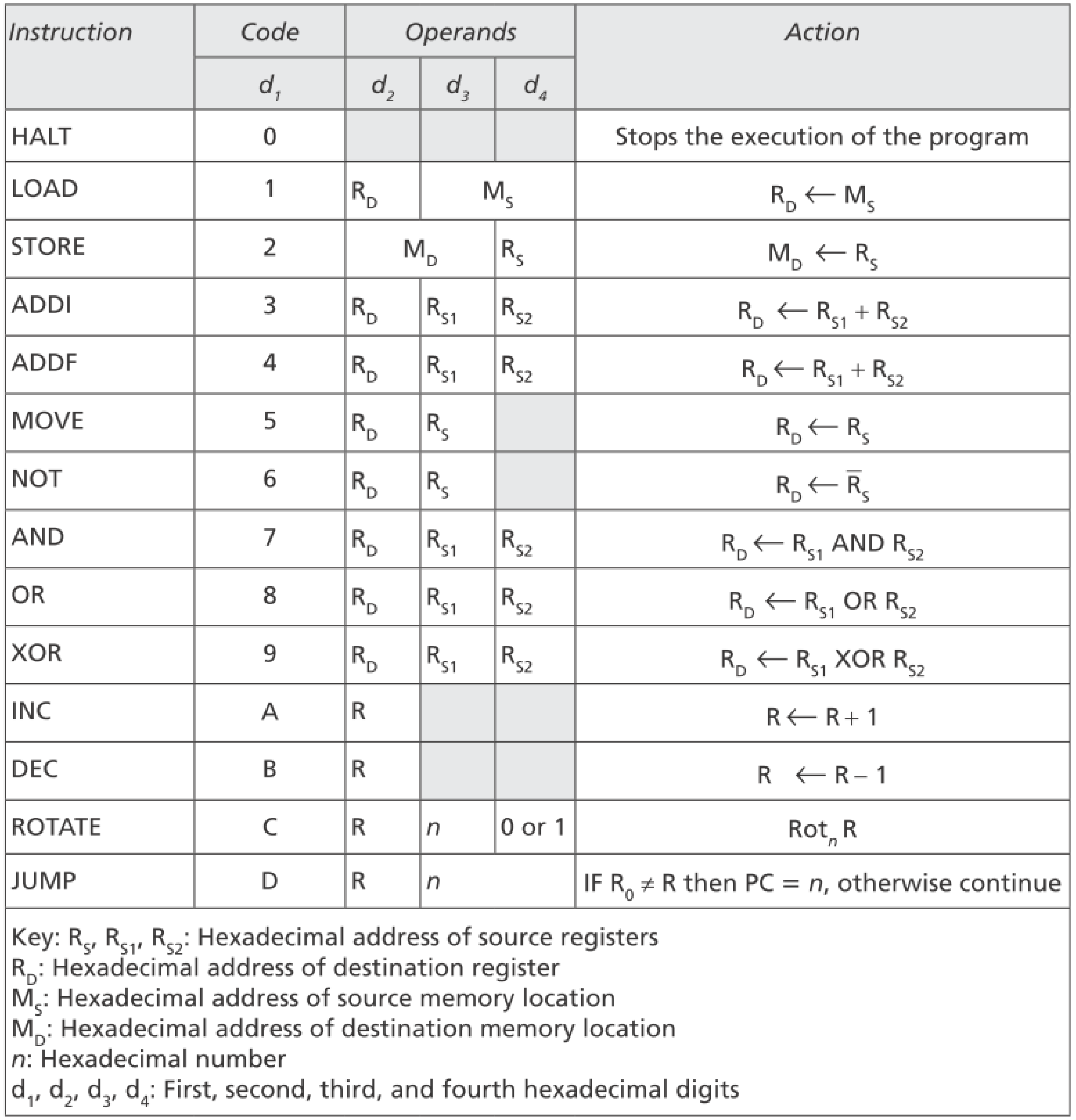
a. Instruction format



b. Instruction types

Instruction set

- Note that not every instruction requires three operands. Any operand field not needed is filled with $(0)_{16}$
  - If the third operand of the ROTATE instruction is 0, the instruction circularly rotates the bit pattern to the right *n* places
- A machine cycle is made of three phases: fetch, decode, and execute
  - The process continues until the CPU reaches a HALT instruction



Instruction	*Code*	*Operands*			*Action*
	d_1	d_2	d_3	d_4	
HALT	0				Stops the execution of the program
LOAD	1	R_D	M_S		$R_D \leftarrow M_S$
STORE	2	M_D		R_S	$M_D \leftarrow R_S$
ADDI	3	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
ADDF	4	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
MOVE	5	R_D	R_S		$R_D \leftarrow R_S$
NOT	6	R_D	R_S		$R_D \leftarrow \overline{R_S}$
AND	7	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ AND R_{S2}
OR	8	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ OR R_{S2}
XOR	9	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ XOR R_{S2}
INC	A	R			$R \leftarrow R + 1$
DEC	B	R			$R \leftarrow R - 1$
ROTATE	C	R	*n*	0 or 1	$\text{Rot}_n R$
JUMP	D	R	*n*		IF $R_0 \neq R$ then PC = *n*, otherwise continue

Key: R_S, R_{S1}, R_{S2}: Hexadecimal address of source registers
R_D: Hexadecimal address of destination register
M_S: Hexadecimal address of source memory location
M_D: Hexadecimal address of destination memory location
n: Hexadecimal number
d_1, d_2, d_3, d_4: First, second, third, and fourth hexadecimal digits

An example

- Let us show how our simple computer can add two integers A and B and create the result as C. We assume that integers are in two's complement format
  - We assume that the first two integers are stored in memory locations $(40)_{16}$ and $(41)_{16}$ and the result should be stored in memory location $(42)_{16}$. To do the simple addition needs five instructions, as shown next:

1. Load the contents of M_{40} into register R_0 ($R_0 \leftarrow M_{40}$).
2. Load the contents of M_{41} into register R_1 ($R_1 \leftarrow M_{41}$).
3. Add the contents of R_0 and R_1 and place the result in R_2 ($R_2 \leftarrow R_0 + R_1$).
4. Store the contents R_2 in M_{42} ($M_{42} \leftarrow R2$).
5. Halt.

An example

- In the language of our simple computer, these five instructions are encoded as:

1. Load the contents of M_{40} into register R_0 ($R_0 \leftarrow M_{40}$).
2. Load the contents of M_{41} into register R_1 ($R_1 \leftarrow M_{41}$).
3. Add the contents of R_0 and R_1 and place the result in R_2 ($R_2 \leftarrow R_0 + R_1$).
4. Store the contents R_2 in M_{42} ($M_{42} \leftarrow$ R2).
5. Halt.

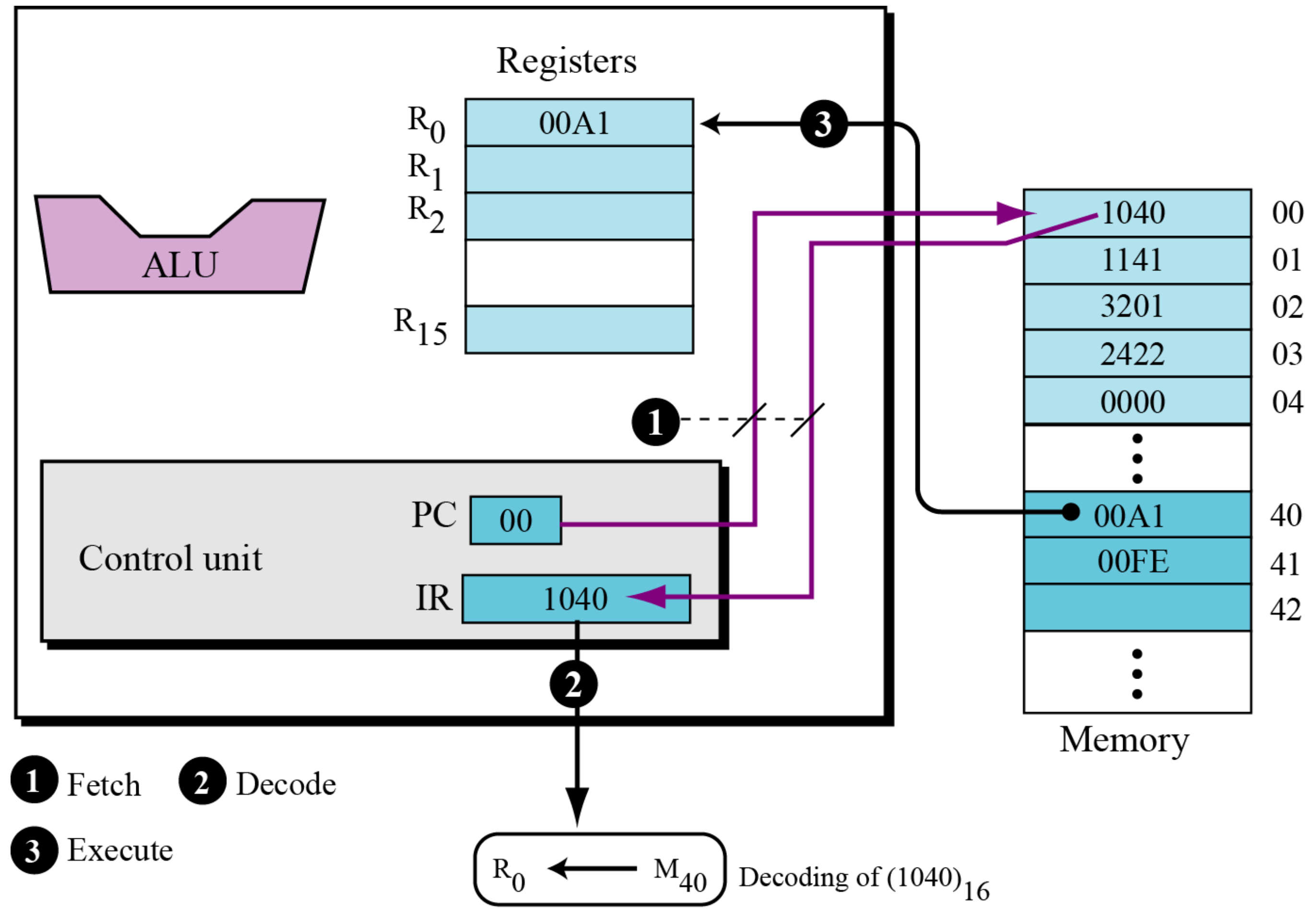
Code	*Interpretation*			
$(1040)_{16}$	1: LOAD	0: R_0	40: M_{40}	
$(1141)_{16}$	1: LOAD	1: R_1	41: M_{41}	
$(3201)_{16}$	3: ADDI	2: R_2	0: R_0	1: R_1
$(2422)_{16}$	2: STORE	42: M_{42}		2: R_2
$(0000)_{16}$	0: HALT			

An example - Storing program and data

- We can store the program in memory starting from location $(00)_{16}$ to $(04)_{16}$
  - We already know that the data needs to be stored in memory locations $(40)_{16}$, $(41)_{16}$, and $(42)_{16}$
- Our computer uses one cycle per instruction. If we have a small program with five instructions, we need five cycles
  - We also know that each cycle is normally made up of three steps: *fetch, decode, execute*. Assume for the moment that we need to add $161 + 254 = 415$. The numbers shown in memory in hexadecimal are, $(00A1)_{16}$, $(00FE)_{16}$, and $(019F)_{16}$

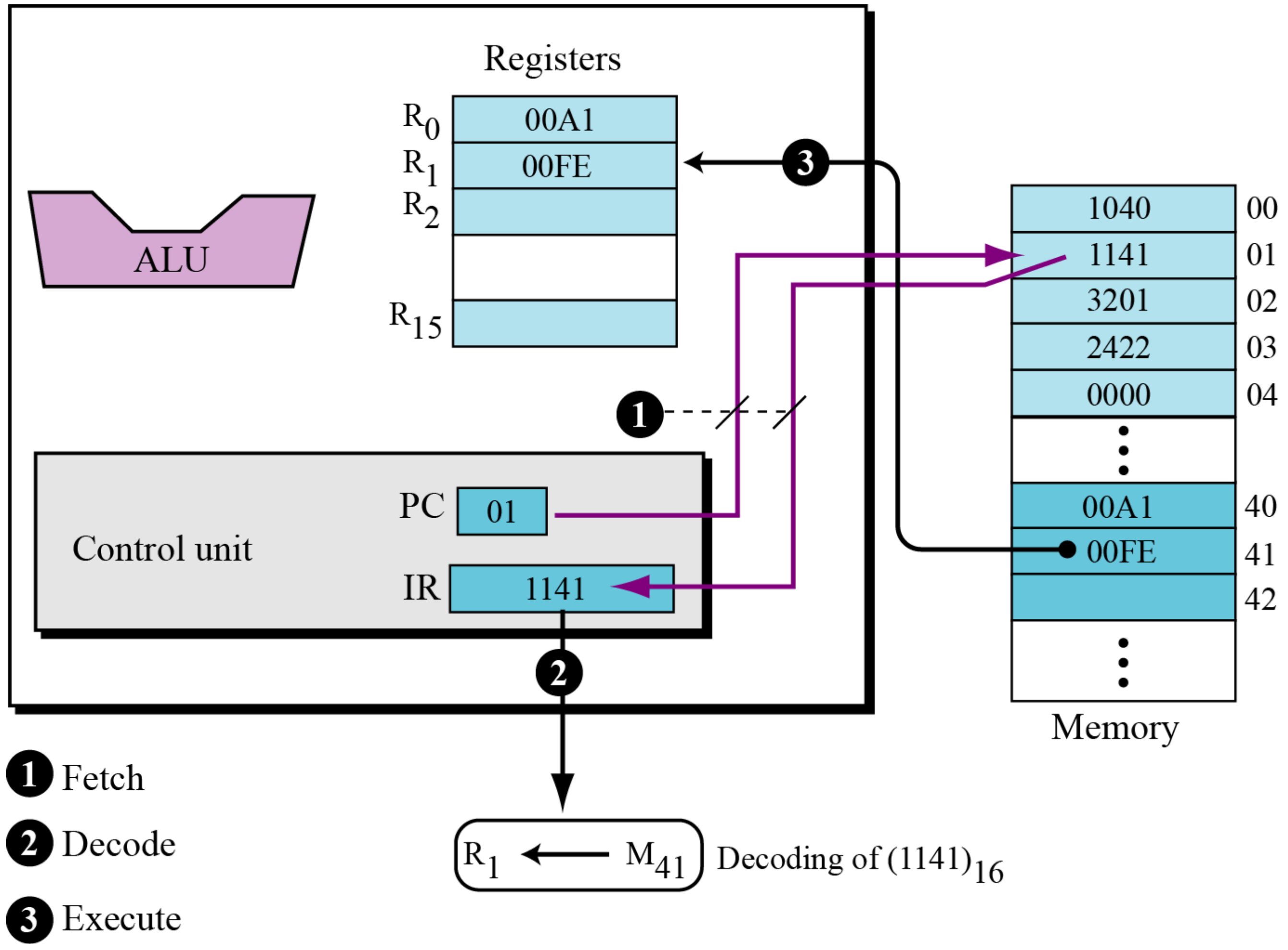
An example - Cycle 1

1. The control unit *fetches* the instruction stored in memory location $(00)_{16}$ and puts it in the IR. After this step, the value of the PC is incremented
2. The control unit *decodes* the instruction $(1040)_{16}$ as $R_0 \leftarrow M_{40}$
3. The control unit *executes* the instruction, which means that a copy of the integer stored in memory location $(40)_{16}$ is loaded into register R_0



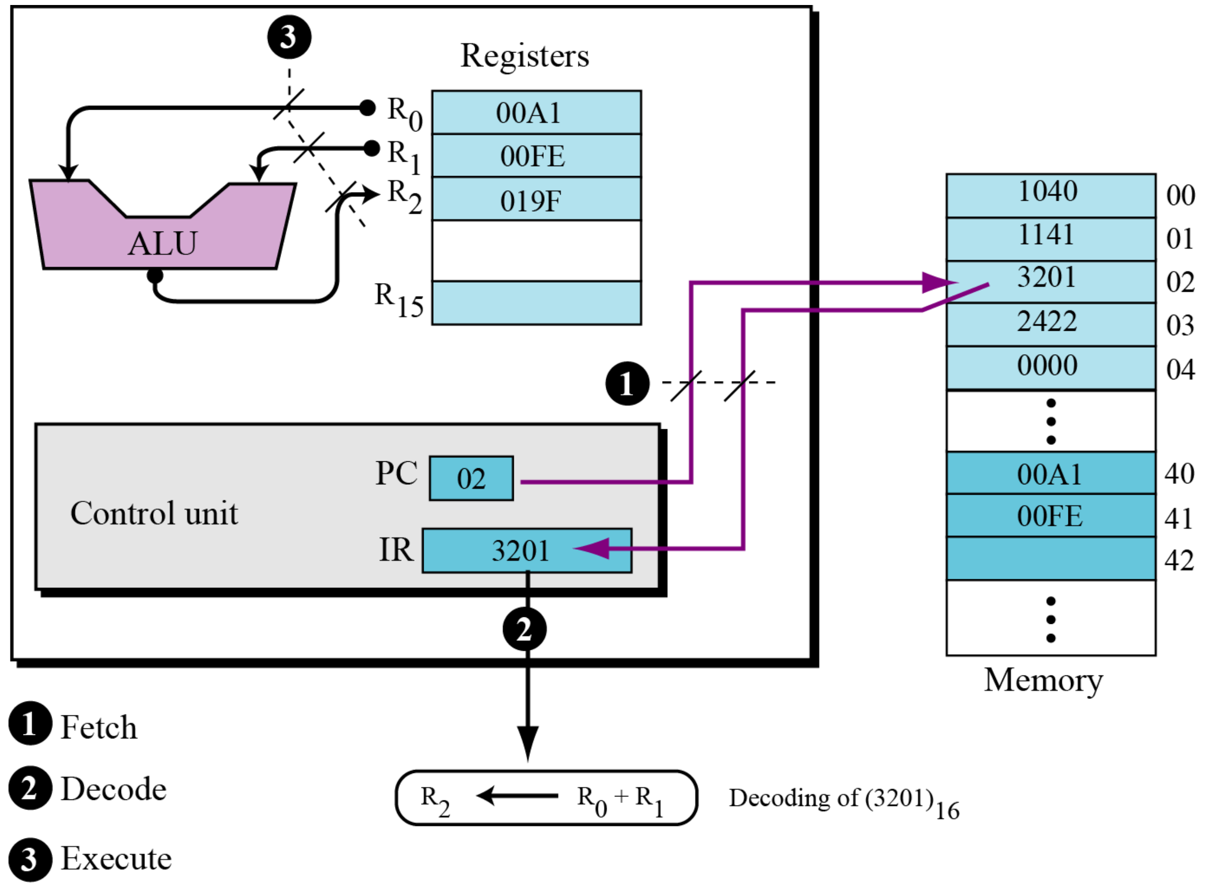
An example - Cycle 2

1. The control unit *fetches* the instruction stored in memory location $(01)_{16}$ and puts it in the IR. After this step, the value of the PC is incremented
2. The control unit *decodes* the instruction $(1141)_{16}$ as $R_1 \leftarrow M_{41}$
3. The control unit *executes* the instruction, which means that a copy of the integer stored in memory location $(41)_{16}$ is loaded into register R_1



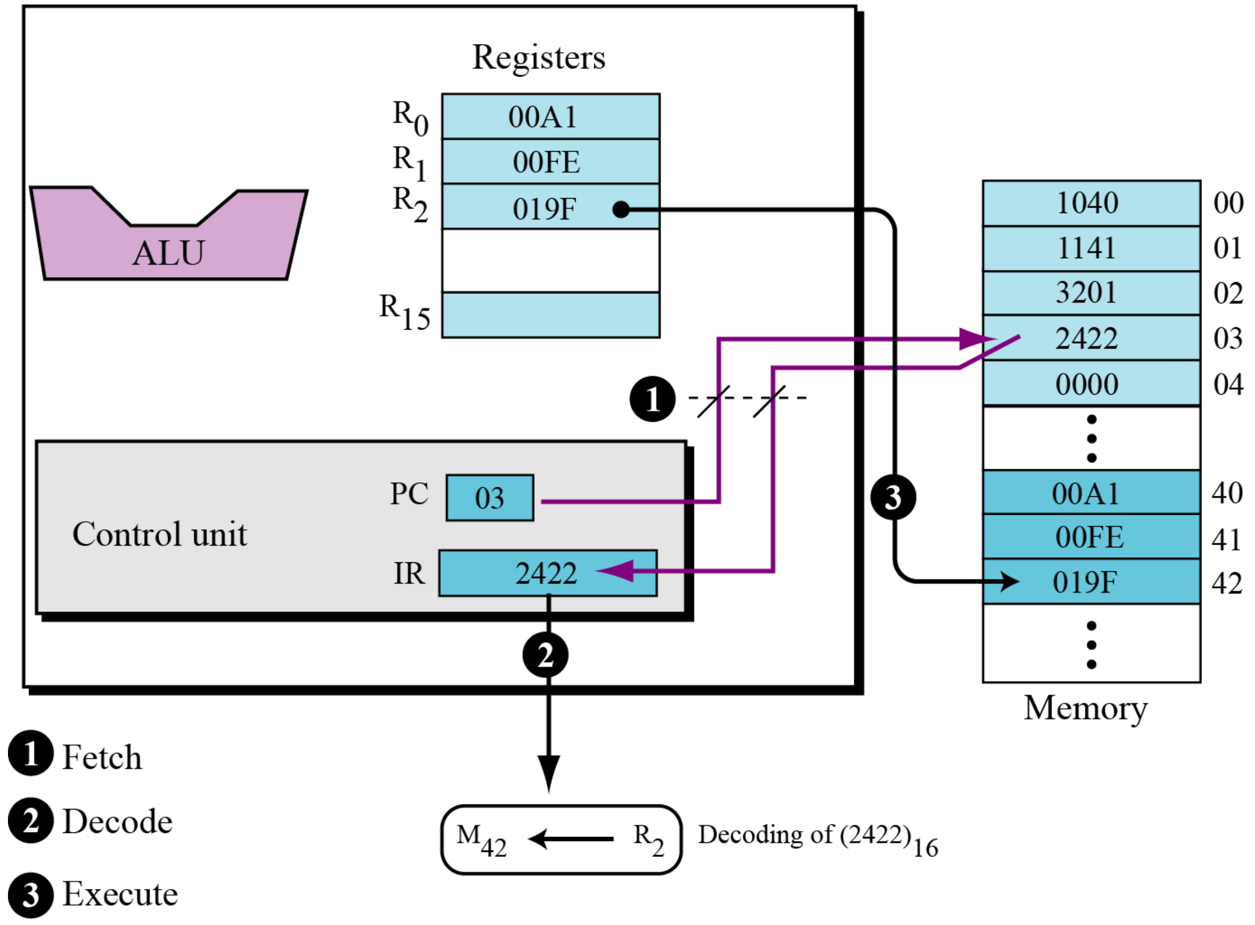
An example - Cycle 3

1. The control unit *fetches* the instruction stored in memory location $(02)_{16}$ and puts it in the IR. After this step, the value of the PC is incremented
2. The control unit *decodes* the instruction $(3201)_{16}$ as $R_2 \leftarrow R_0 + R_1$
3. The control unit *executes* the instruction, which means that the contents R_0 is added to R_1 and the result is put in R_2



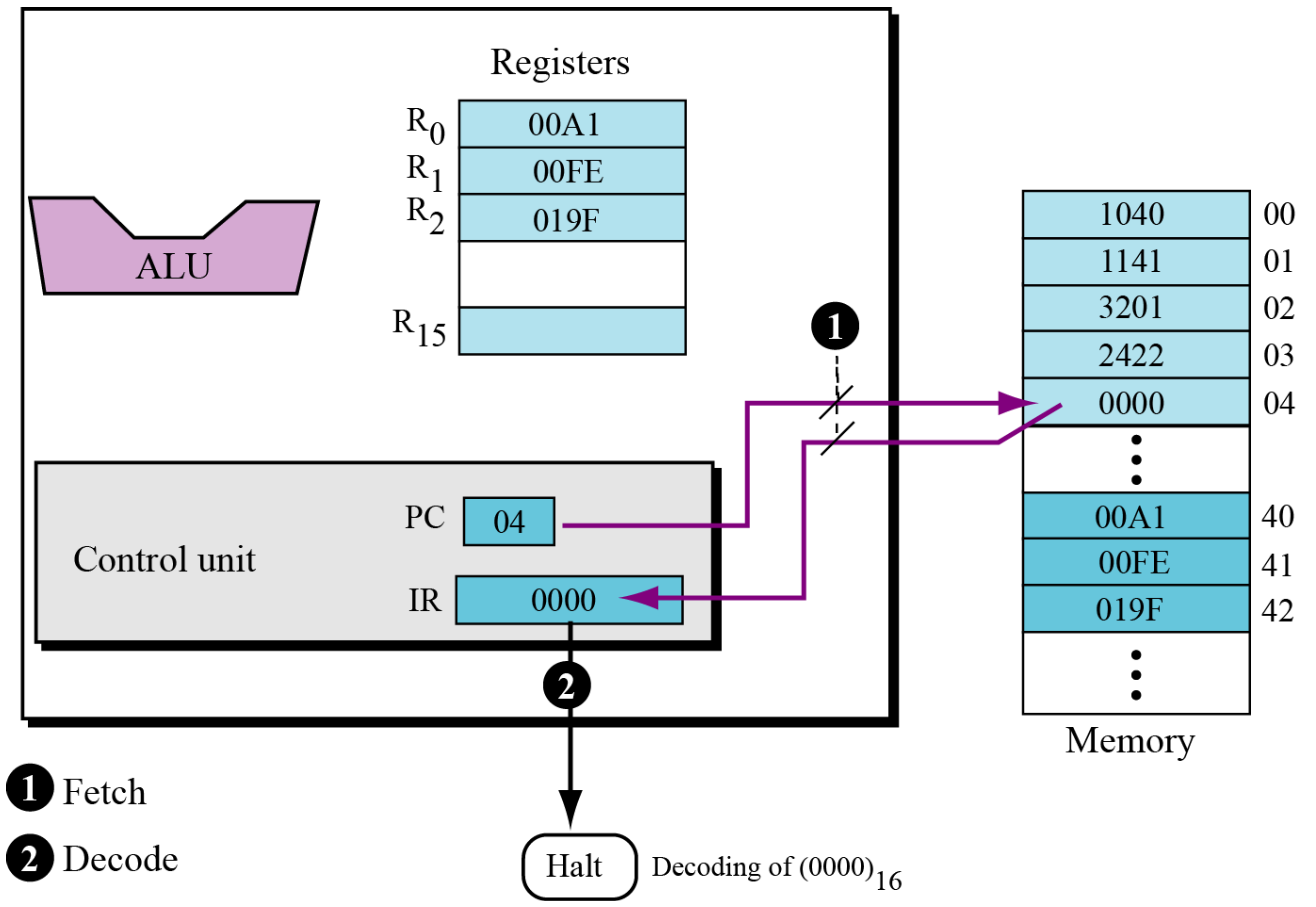
An example - Cycle 4

1. The control unit *fetches* the instruction stored in memory location $(03)_{16}$ and puts it in the IR. After this step, the value of the PC is incremented
2. The control unit *decodes* the instruction $(2422)_{16}$ as $M_{42} \leftarrow R_2$
3. The control unit *executes* the instruction, which means a copy of integer in R_2 is stored in memory location $(42)_{16}$



An example - Cycle 5

1. The control unit *fetches* the instruction stored in memory location $(04)_{16}$ and puts it in the IR. After this step, the value of the PC is incremented
2. The control unit *decodes* the instruction $(0000)_{16}$ as Halt
3. The control unit *executes* the instruction, which means that the computer stops

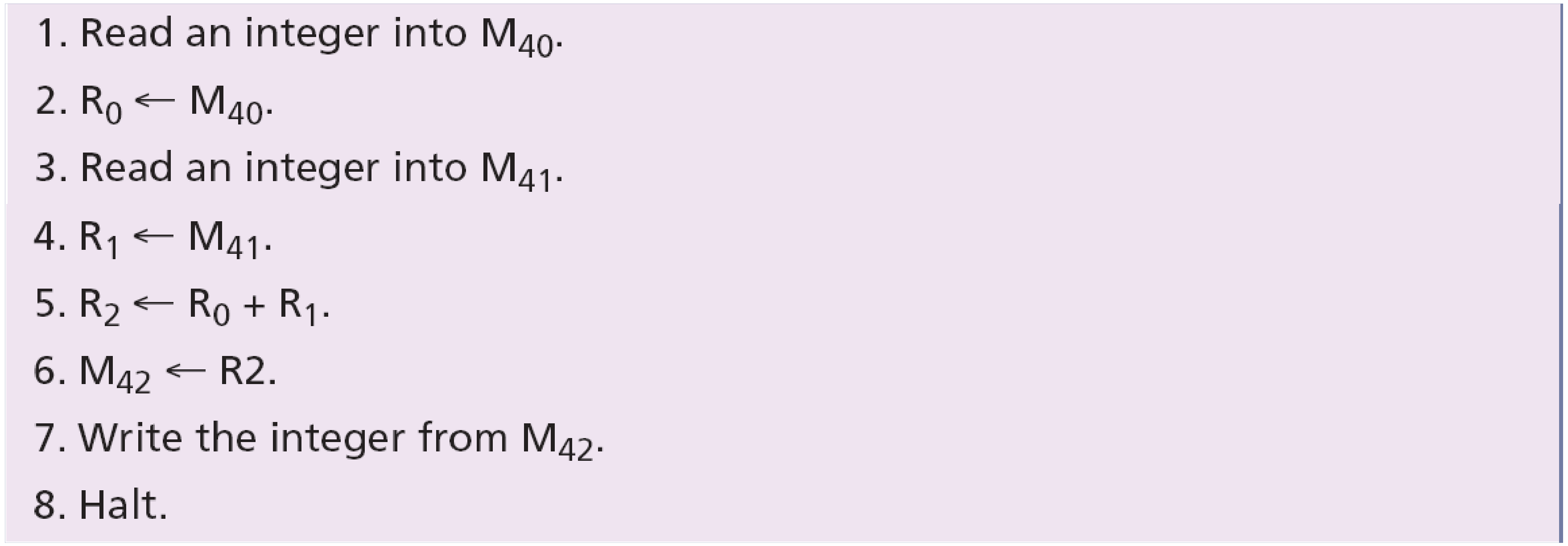


Another example

- In the previous example we assumed that the two integers to be added were already in memory. We also assume that the result of addition will be held in memory
  - In a real situation, we enter the first two integers into memory using an input device such as keyboard, and we display the third integer through an output device such as a monitor
  - Getting data via an input device is normally called a *read* operation, while sending data to an output device is normally called a *write* operation

Another example

- To make our previous program more practical, we need modify it as follows



1. Read an integer into M_{40}.
2. $R_0 \leftarrow M_{40}$.
3. Read an integer into M_{41}.
4. $R_1 \leftarrow M_{41}$.
5. $R_2 \leftarrow R_0 + R_1$.
6. $M_{42} \leftarrow R2$.
7. Write the integer from M_{42}.
8. Halt.

- In our computer we can simulate read and write operations using the LOAD and STORE instruction

Another example

- LOAD and STORE read data input to the CPU and write data from the CPU. We need two instructions to read data into memory or write data out of memory

$R \leftarrow M_{FE}$ Because the keyboard is assumed to be memory location $(FE)_{16}$

$M \leftarrow R$

$R \leftarrow M$

$M_{FF} \leftarrow R$ Because the monitor is assumed to be memory location $(FF)_{16}$

- The input operation must always read data from an input device into memory and the output operation must always write data from memory to an output device

Another example

- With this in mind, the program is coded as:

1	$(1FFE)_{16}$	5	$(1040)_{16}$	9	$(1F42)_{16}$
2	$(240F)_{16}$	6	$(1141)_{16}$	10	$(2FFF)_{16}$
3	$(1FFE)_{16}$	7	$(3201)_{16}$	11	$(0000)_{16}$
4	$(241F)_{16}$	8	$(2422)_{16}$		

- Operations 1 to 4 are for input and operations 9 and 10 are for output. When we run this program, it waits for the user to input two integers on the keyboard and press the enter key. The program then calculates the sum and displays the result on the monitor

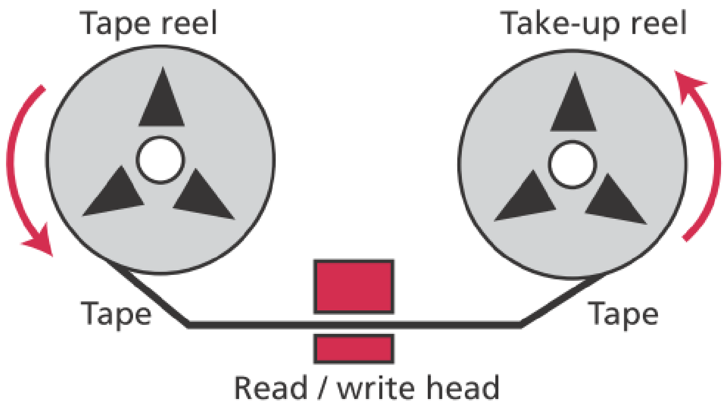
Reusability

- One of the advantages of a computer over a non-programmable calculator is that we can use the same program over and over
  - We can run the program several times and each time enter different inputs and obtain a different output

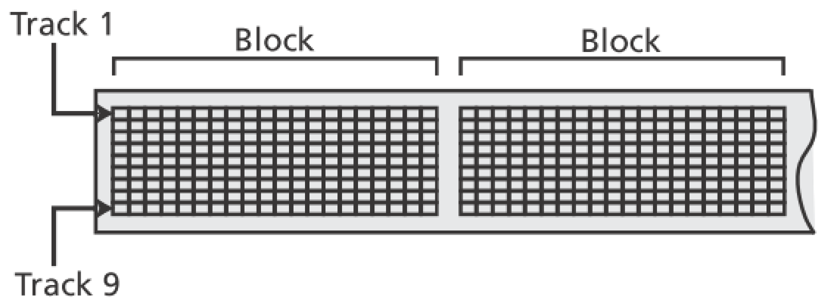
Appendix

Input/output subsystem - Magnetic storage devices

- Magnetic tape (磁帶)
  - One common type is half-inch plastic tape coated with a thick magnetic film
  - The tape is mounted on two reels and uses a read/write head that reads or writes information when the tape is passed through it
  - Surface organization
    - The width of the tape is divided into nine tracks, each location on a track storing 1 bit of information. Nine vertical locations can store 8 bits of information plus a bit for error detection



a. Tape drive



b. Surface orgnization

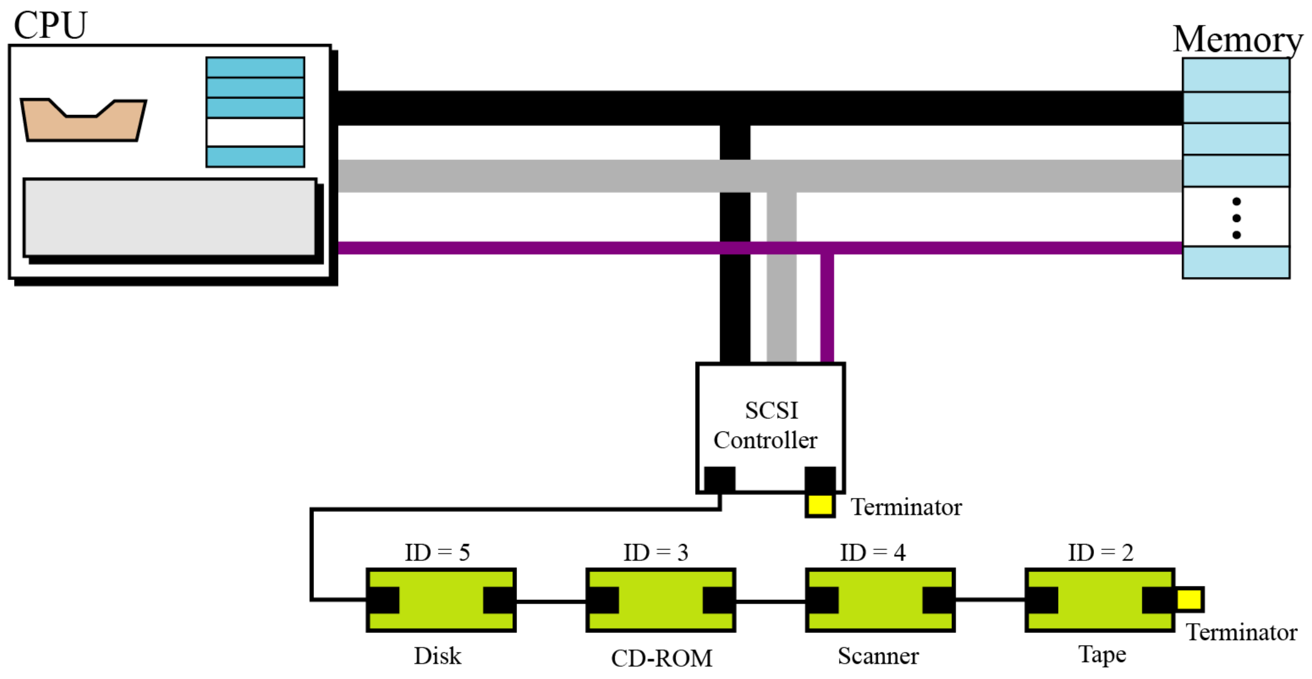
Input/output subsystem - Magnetic storage devices

- Magnetic tape
  - Data access
    - A magnetic tape is considered a sequential access device. Although the surface may be divided into blocks, there is no addressing mechanism to access each block
    - To retrieve a specific block on the tape, we need to pass through all the previous blocks
  - Performance
    - Although magnetic tape is slower than a magnetic disk, it is cheaper. Today, people use magnetic tape to back up large amounts of data

https://en.wikipedia.org/wiki/Magnetic_tape#/media/File:Kaseta_magnetofonowa_ubt.jpeg

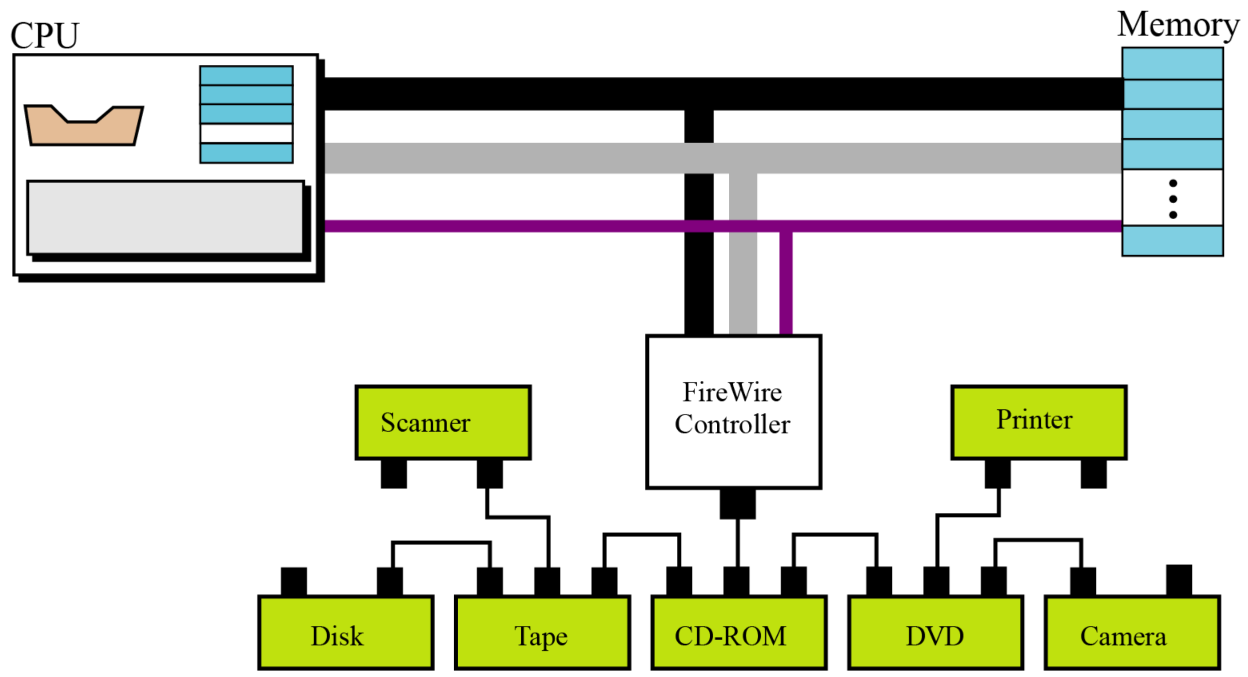
Subsystem interconnection

- A controller can be a serial or parallel device
  - A serial controller has only one data wire, while a parallel controller has several data connections so that several bits can be transferred at a time
- Small computer system interface (SCSI) (小型電腦系統介面)
  - It is used in many systems. It has a *parallel interface* with 8, 16, or 32 connections
  - It provides a daisy-chained connection. Both ends of the chain must be connected to a special device called a *terminator*, and each device must have a unique address (target ID)



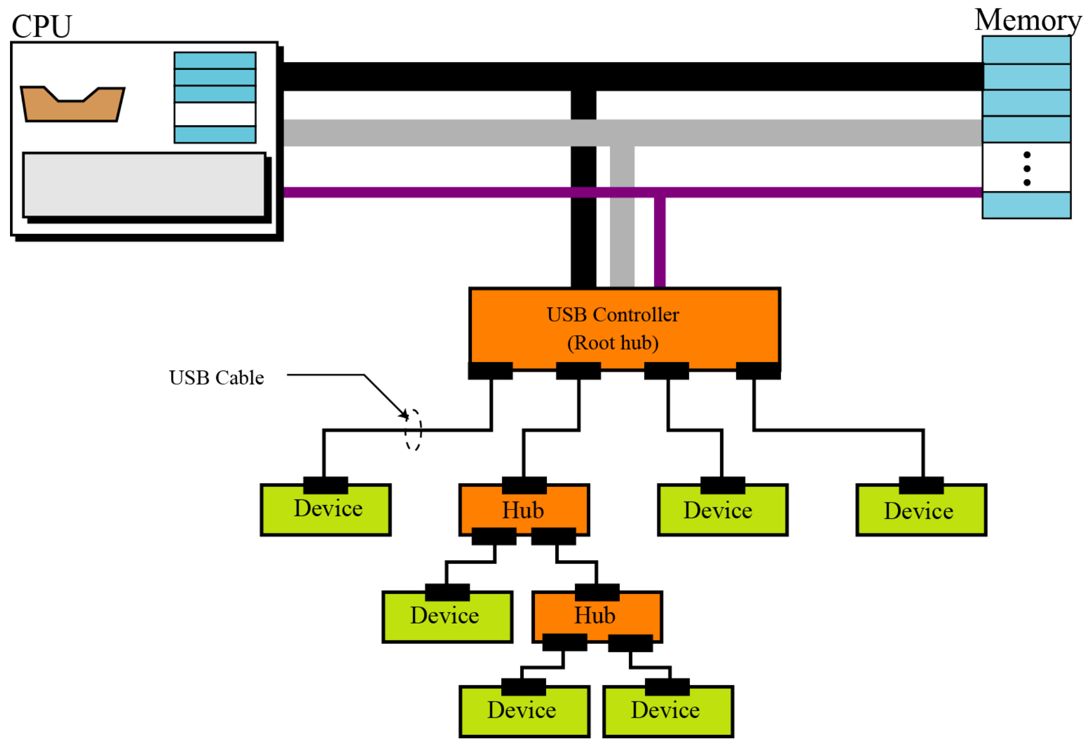
Subsystem interconnection

- FireWire
  - It is a high-speed *serial interface* that transfers data in packets, achieving a transfer rate of up to 50 MB/sec, or double that in the most recent version
  - It can be used to connect up to 63 devices in a daisy chain or a tree connection (using only one connection)



Subsystem interconnection

- Universal Serial Bus (USB)
  - USB is a *serial controller* that connects both low- and high-speed devices to the computer bus
  - Multiple devices can be connected to a USB controller, which is also referred to as a *root hub*
  - Devices can easily be removed or attached to the tree without powering down the computer. This is referred to as *hot-swappable*
  - USB 3.0 adds a new transfer mode called SuperSpeed capable of transferring data at up to 4.8 Gbit/s. It is promised to update USB 3.0 to 10 Gbit/s



Subsystem interconnection

- HDMI (High-Definition Multimedia Interface)
  - It is a digital replacement for existing analog video standards
  - It can be used for transferring video data digital audio data from a source to a compatible computer monitor, video projector, digital television, or digital audio