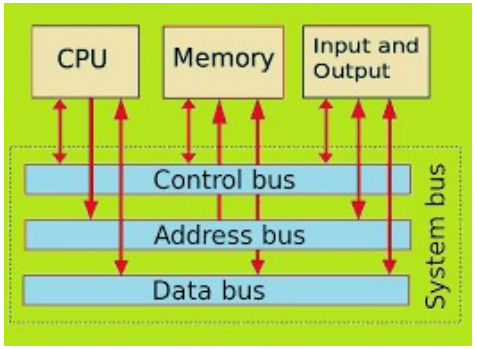
CSE 306 Operating Systems
Computer System Overview

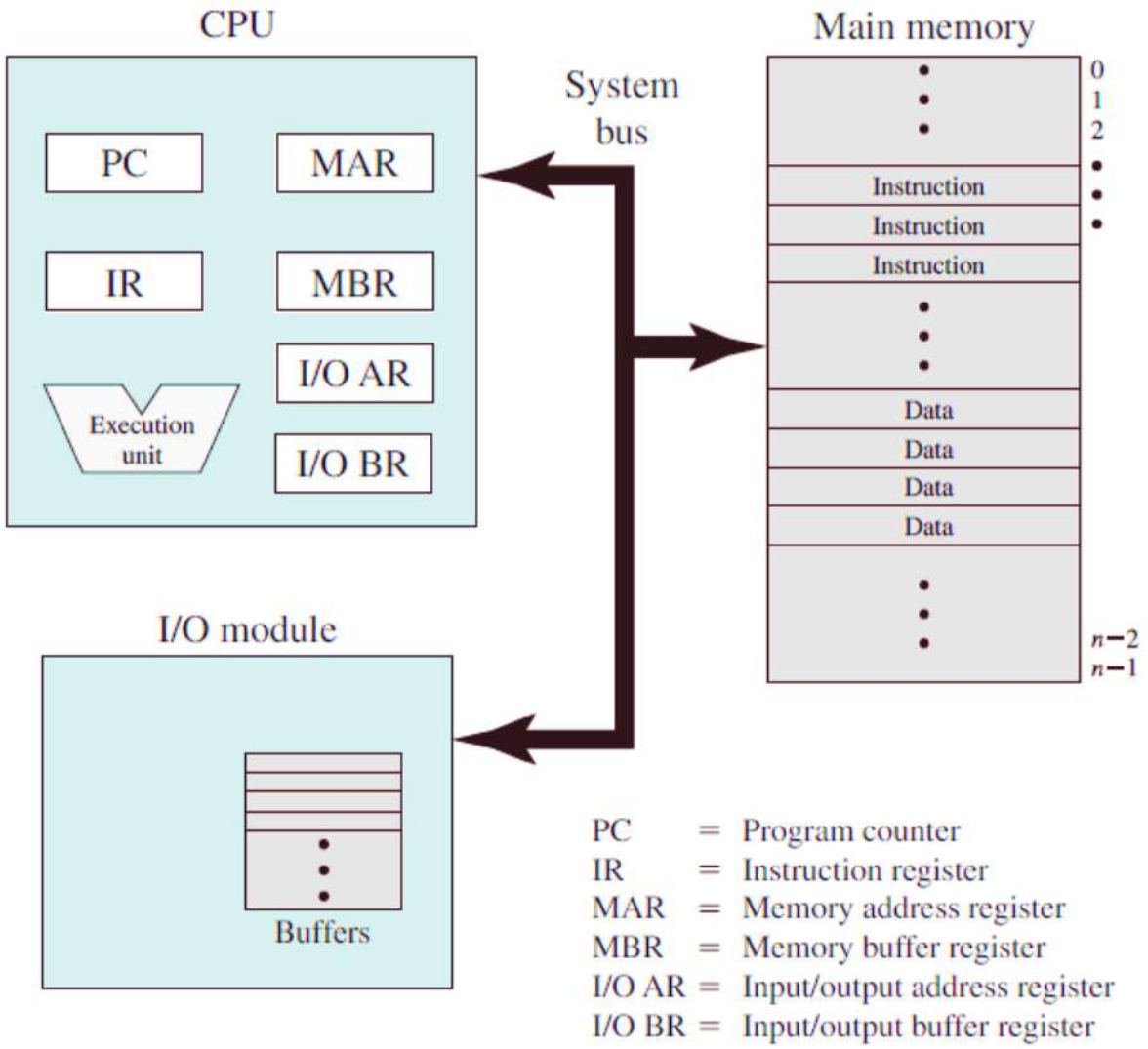
YoungMin Kwon

Basic Elements

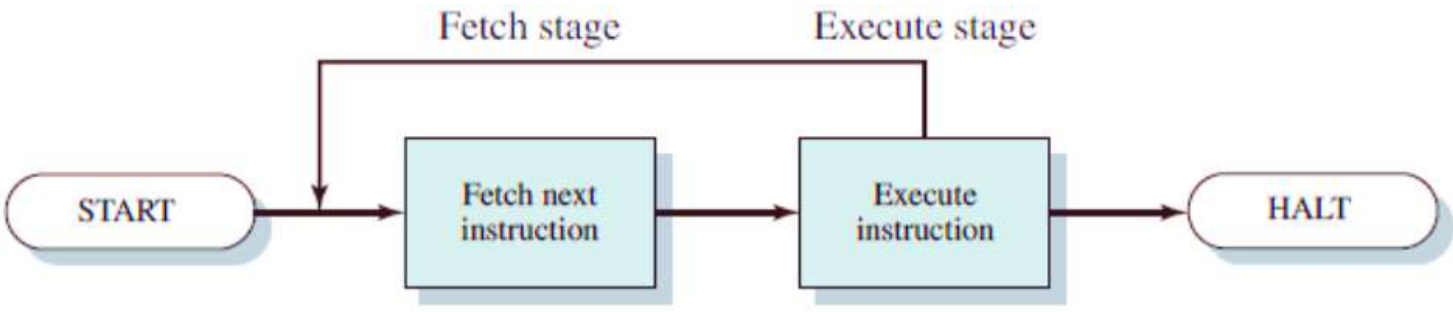


- A computer consists of
  - Processor: controls the operation of the computer and performs its data processing functions
  - Memory: stores data and program
  - I/O modules: move data between the computer and external environments
  - System bus: provides communication among processors, main memory and I/O modules

Computer Components: Top-Level View



Instruction Execution



Basic Instruction Cycle

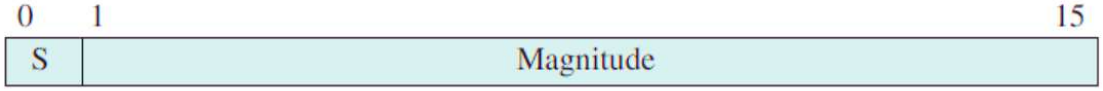
- 2 Steps of Instruction Processing
  - Read instructions (fetches) from memory
    - PC holds the address of the next instruction
    - PC will increase automatically unless instructed otherwise
  - Execute each instruction
    - IR (Instruction Register) holds the fetched instruction

Categories of Instructions

- Data transfer: processor ↔ memory
  - Data may be transferred between a processor and memory
- Data transfer: processor ↔ I/O
  - Data may be transferred between a processor and an I/O module
- Data processing
  - Processor may perform some arithmetic or logical operations on data
- Control
  - Change the sequence of instructions to execute

Program Execution Example

(a) Instruction format



(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

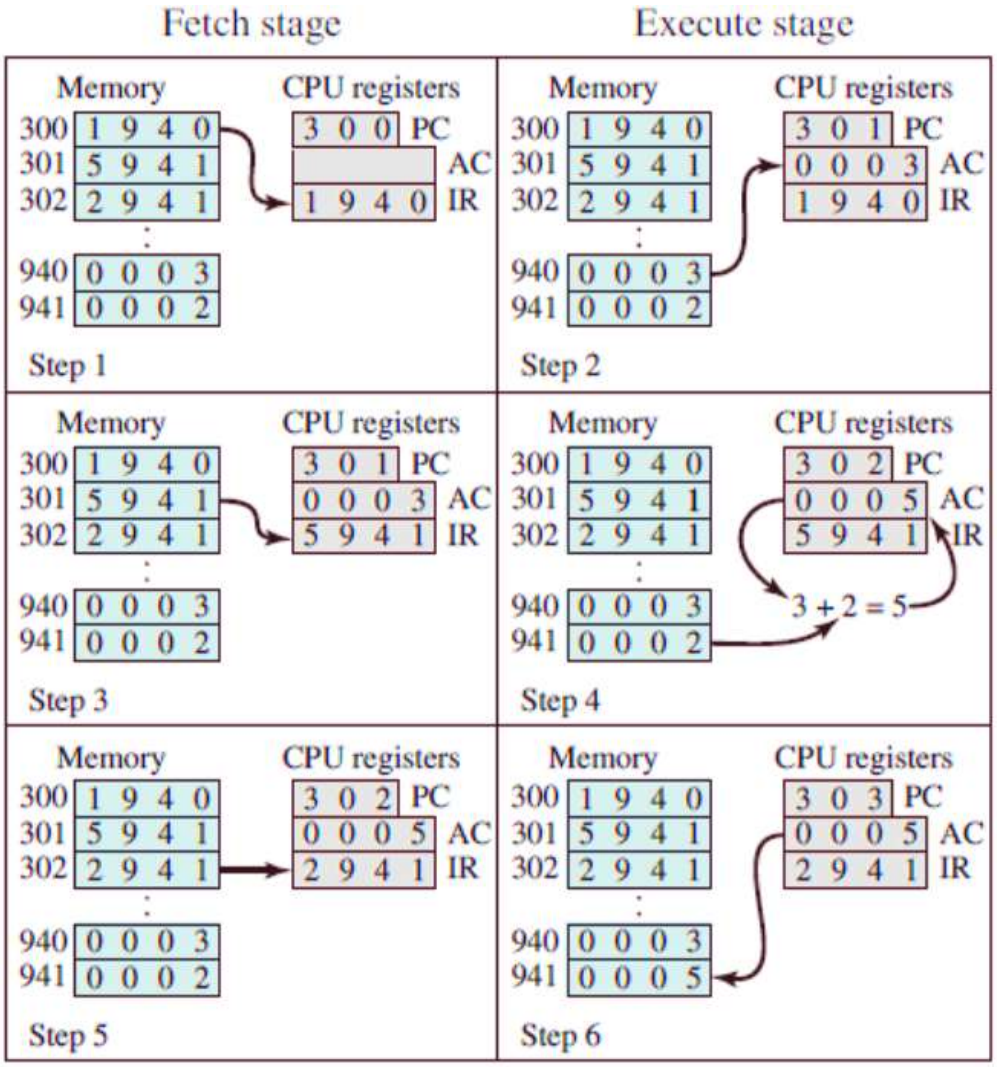
(c) Internal CPU registers

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

- A Hypothetical Machine
  - Has 1 data register AC: accumulator
  - Instructions (16 bit)
    - 4 bits for the opcode (16 different opcodes)
    - 12 bits for the memory address
  - Data (16 bit)

Program Execution Example



0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

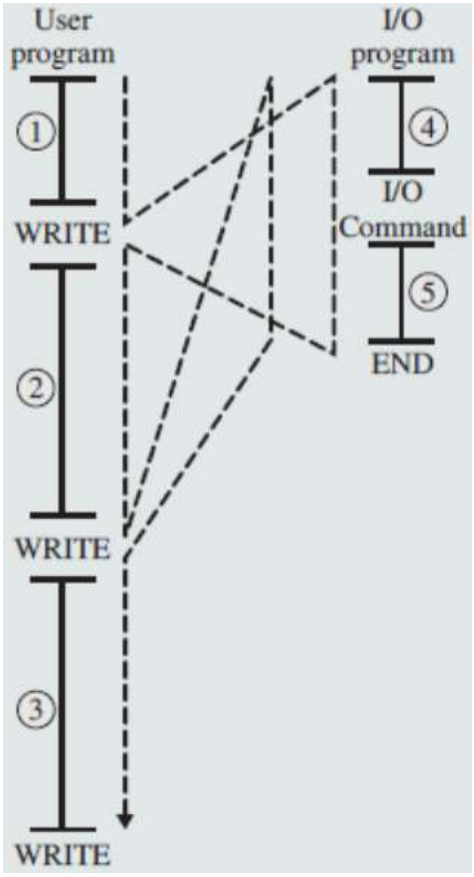
Interrupts

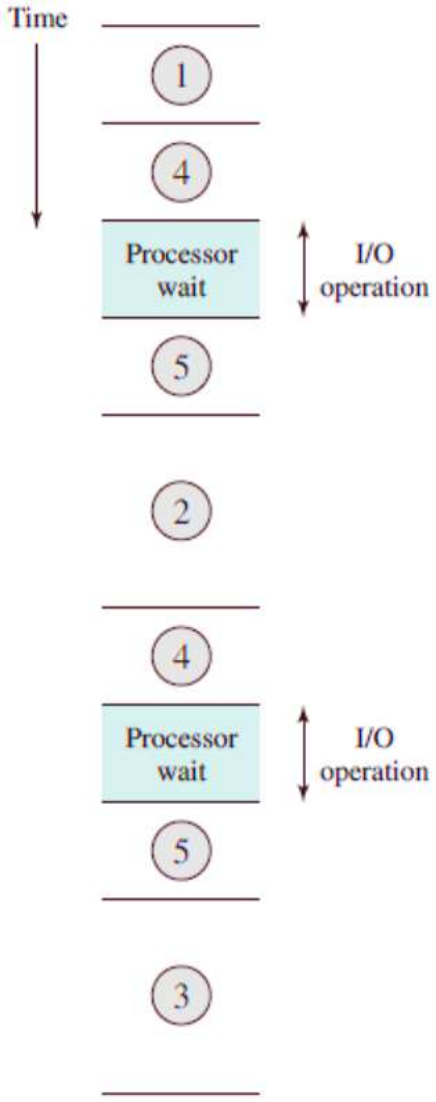
- Interrupts
  - A mechanism by which other modules (I/O, memory) may interrupt the normal sequence of the processor
- Commonly generated by
  - Program: division by 0, illegal memory access, illegal instruction
  - Timer: at a regular interval
  - I/O: to signal a completion of an operation, to signal a detection of an event
  - Hardware failure: power failure, parity error

Interrupts

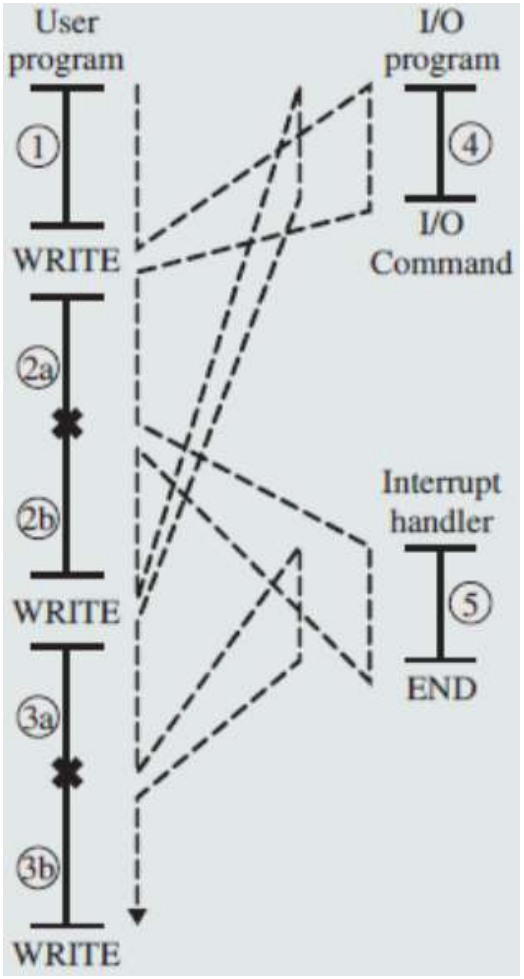
- Interrupt mechanisms can improve processor utilization
- Example
  - A PC operating at 1 GHz would execute about 10^9 instructions per second
  - For a typical HDD with 7200 RPM, it takes about 4 ms to write
  - Without interrupts
    - A processor has to wait 4 ms for every write operation
  - With interrupts,
    - A process makes a write request and proceed with the next instructions.
    - When the write is done an interrupt handler will be invoked

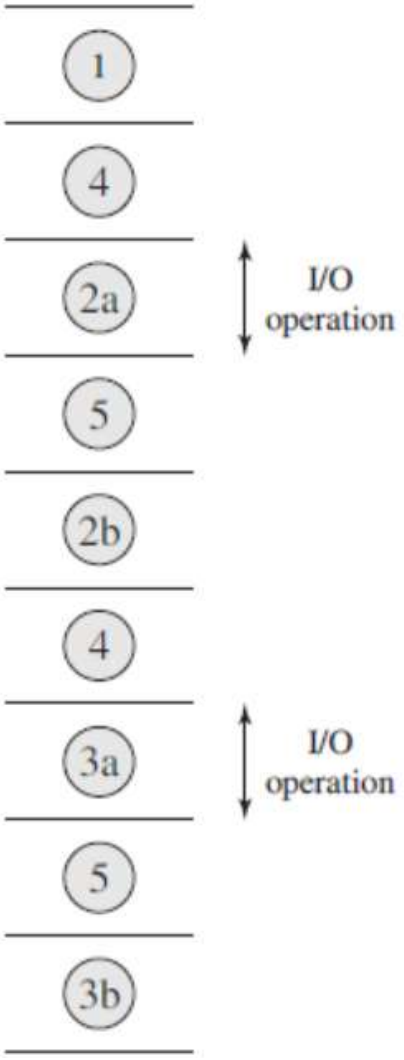
Flow of Control Without Interrupts



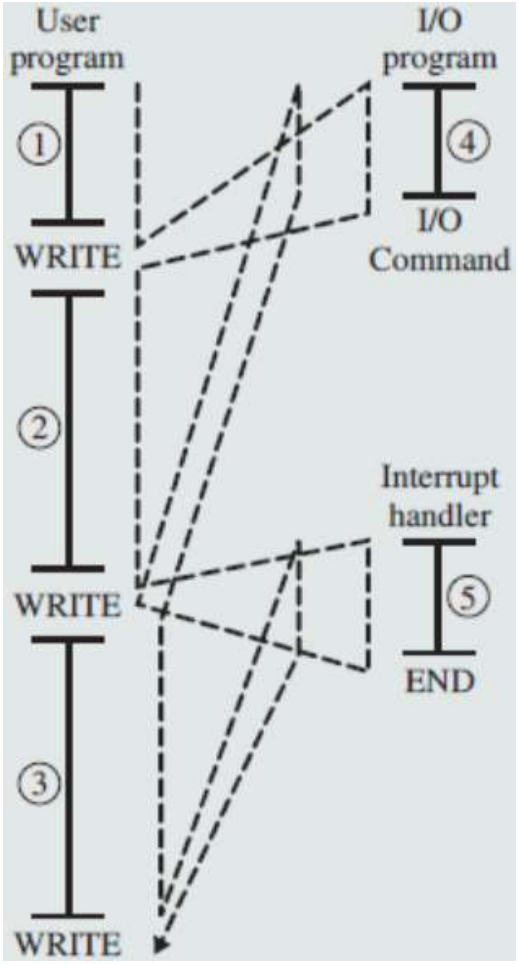


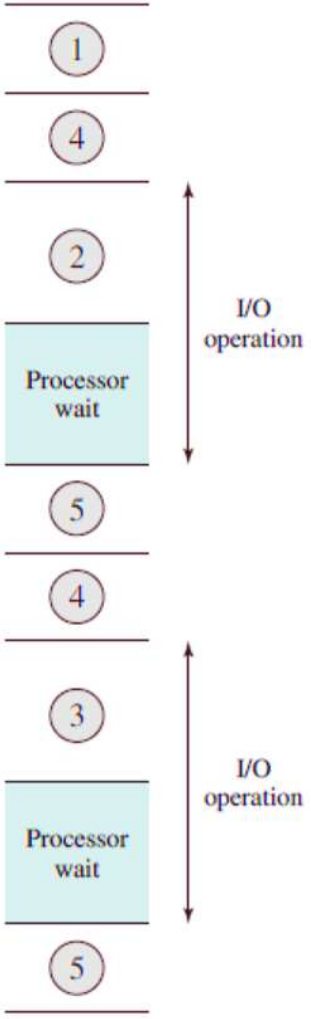
Flow of Control With Interrupts



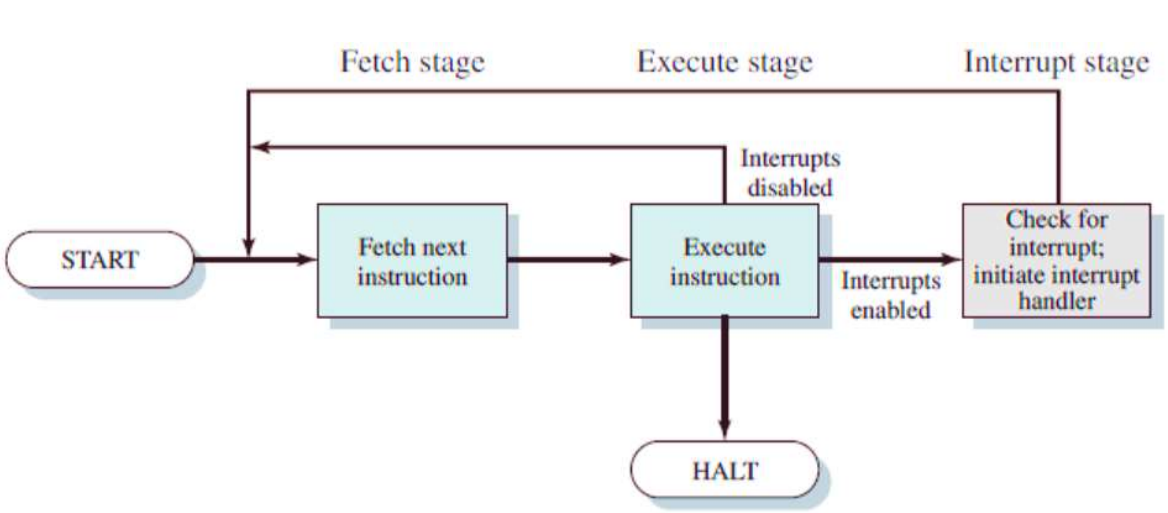


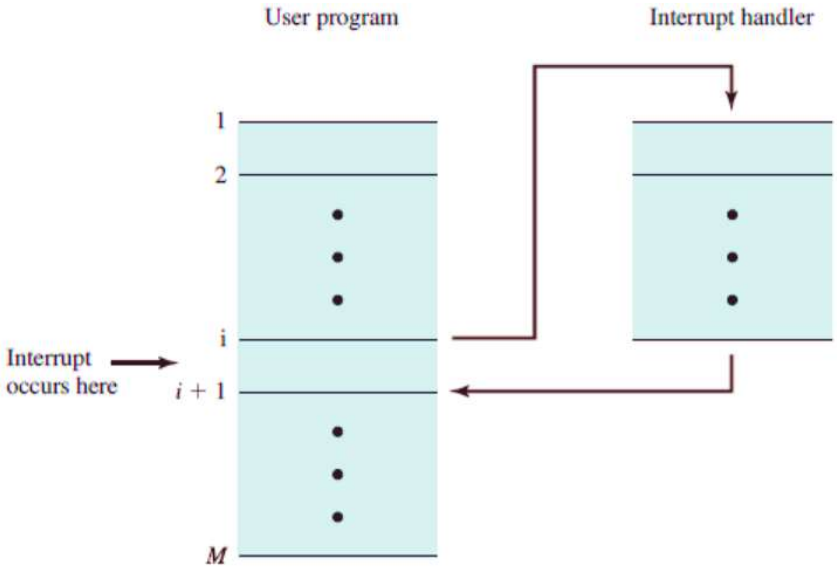
Flow of Control With Interrupts (long I/O wait)





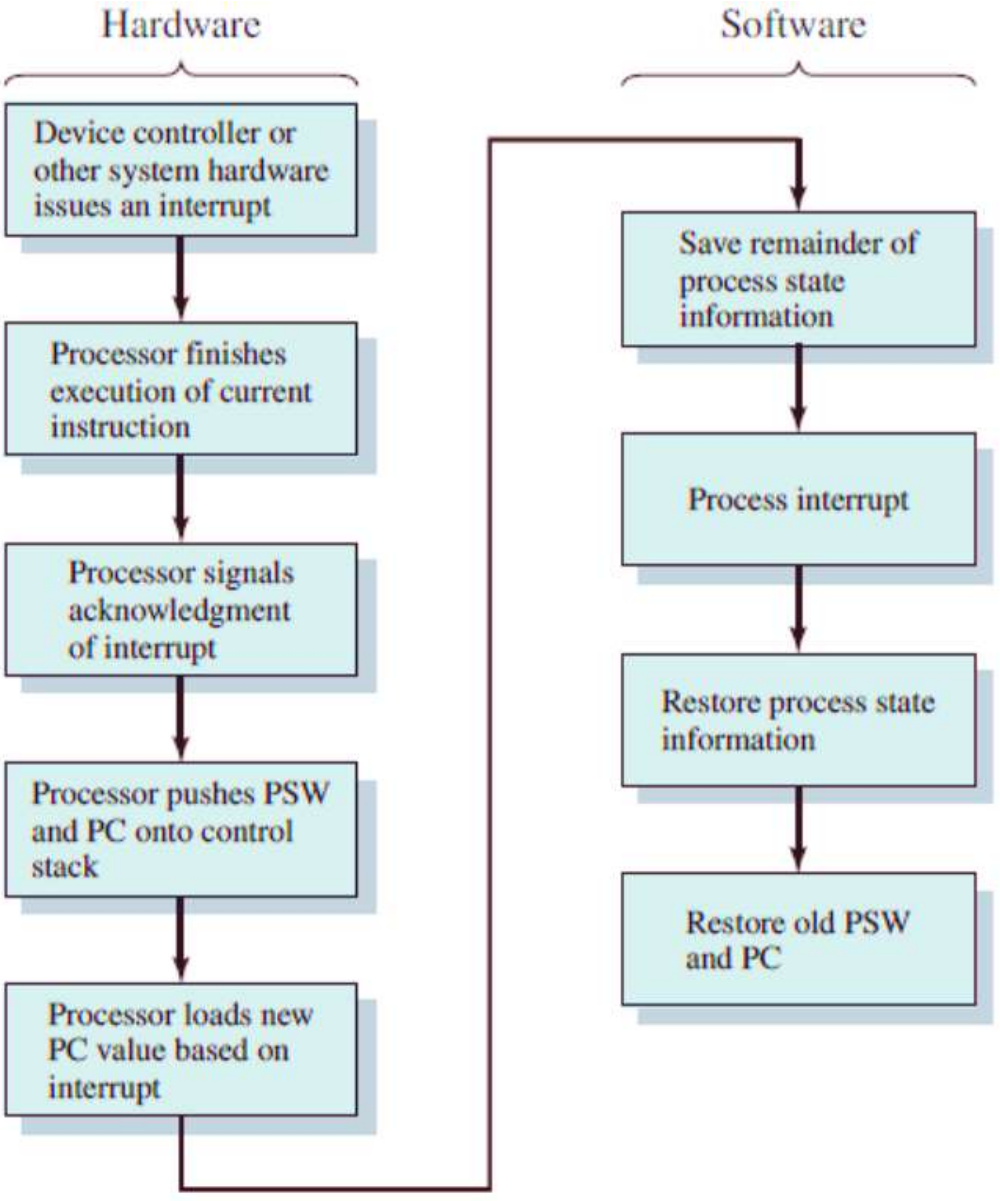
Instruction Cycle with Interrupts



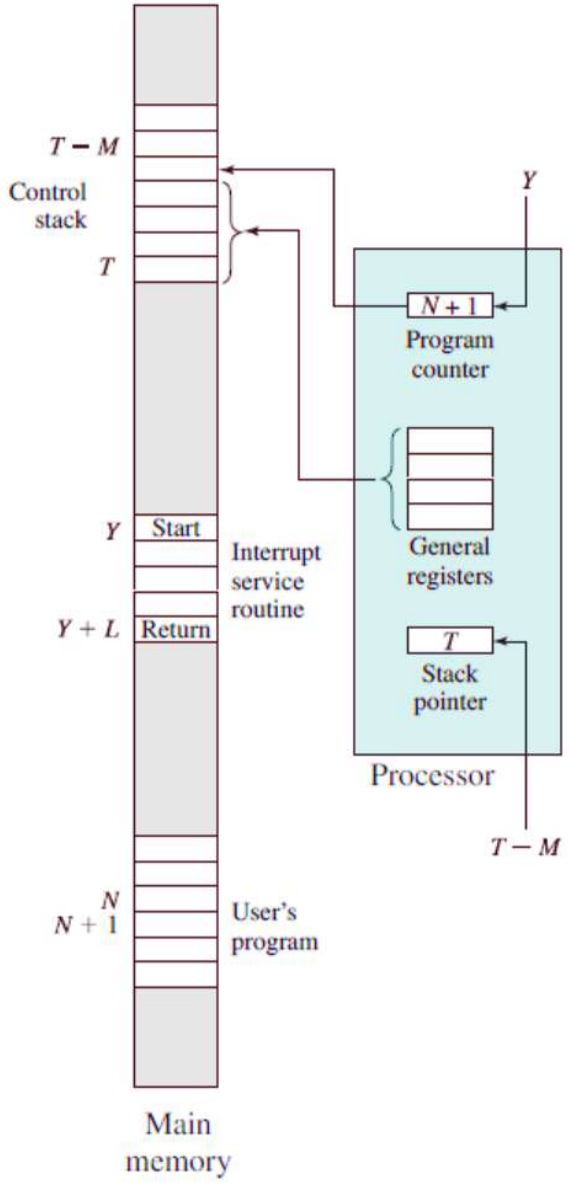


- Interrupt stage
  - For interrupts, an interrupt stage is added to the instruction cycle
  - Processor checks if there are any pending interrupts
    - No pending interrupt: fetches the next instruction
    - Otherwise: executes an interrupt-handler routine

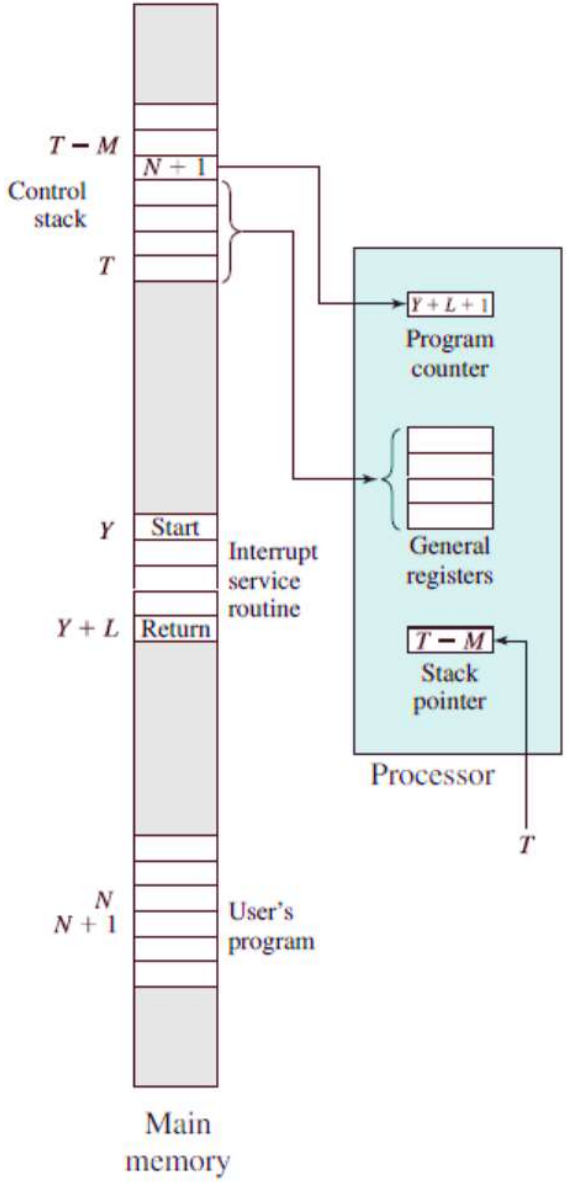
Interrupt Processing



Interrupt: Memory and Register Changes



(a) Interrupt occurs after instruction at location N

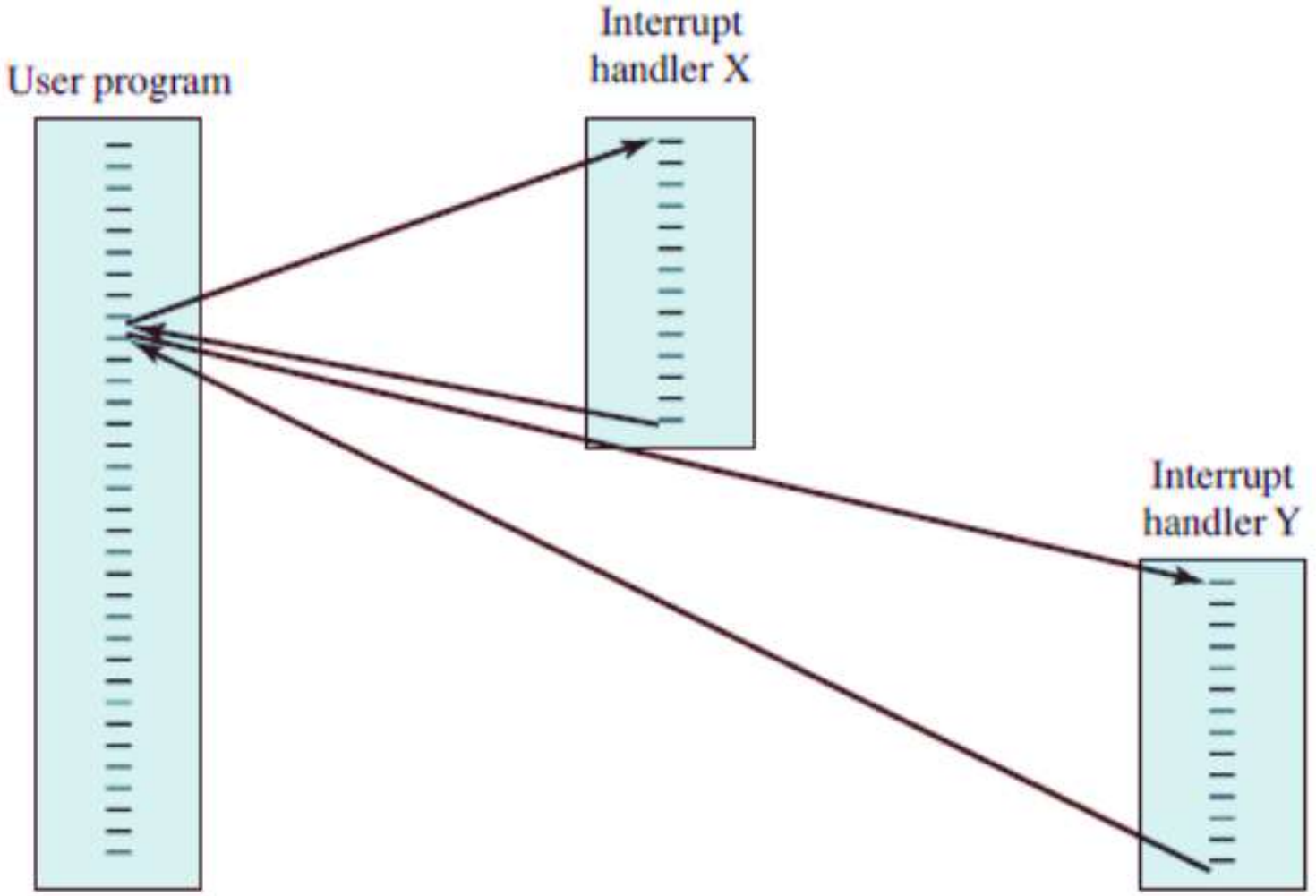


(b) Return from interrupt

Multiple Interrupts (1st approach)

- Disable interrupts while processing one
  - Interrupts occurred during the process remains pending
  - After served the current interrupt, re-enable the interrupt.
  - If there is a pending interrupt, handle it

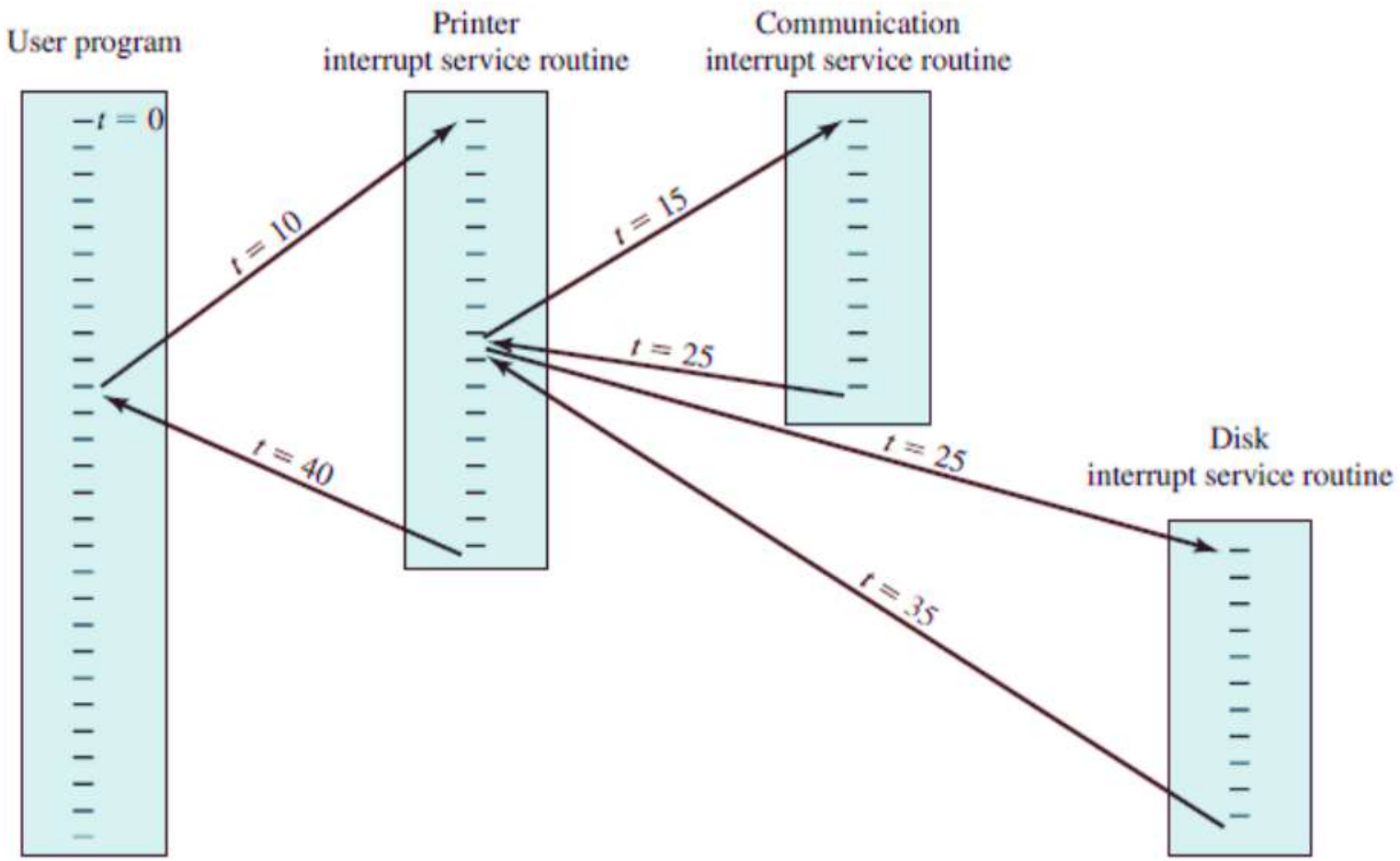
Multiple Interrupts (1st approach)



Multiple Interrupts (2nd approach)

- Define priorities for interrupts and allow an interrupt of higher priority
  - Lower priority interrupts occurred during this process remains pending
  - Higher priority interrupts will be handled
  - After processed the current interrupt, if there is a pending one handle it

Multiple Interrupts (2nd approach)



Designing Memory System

- 3 questions to consider
  - How much
    - If the capacity is there, apps will likely be developed to use it
  - How fast
    - Fast enough not to block processors
  - How expensive
    - Reasonable in relation to other components

- Memory technology
  - Faster access memory → greater cost
  - Larger memory → smaller cost, slower access speed

Storage Technology

- Access Time (to read 512 bytes, year 2015)
  - SRAM: 256 ns
  - DRAM: 4000 ns
  - HDD: 10 ms (40,000 times greater than SRAM, 2,500 times greater than DRAM)

- Cost (year 2015)
  - SRAM: 25 $/MB
  - DRAM: 0.02 $/MB
  - HDD: 0.03 $/GB

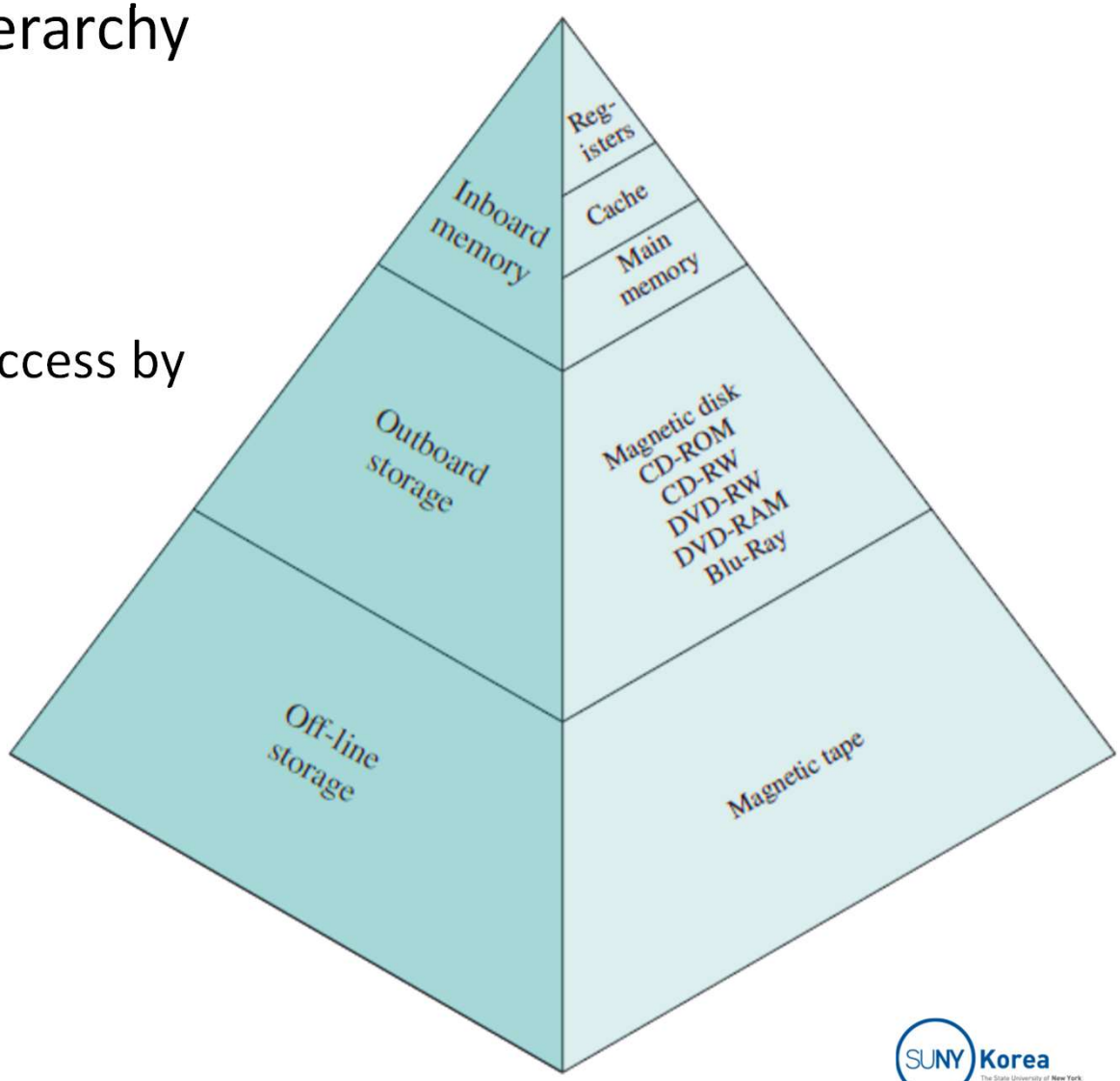
Storage Technology Trend

- 2015 technology compared to 1985

	MB/$	Access time
SRAM	116	115
DRAM	44,000	10
HDD	3,333,333	25
CPU		2,075 (effective cycle time)

Memory Hierarchy

- As one goes down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access by the processor



Locality

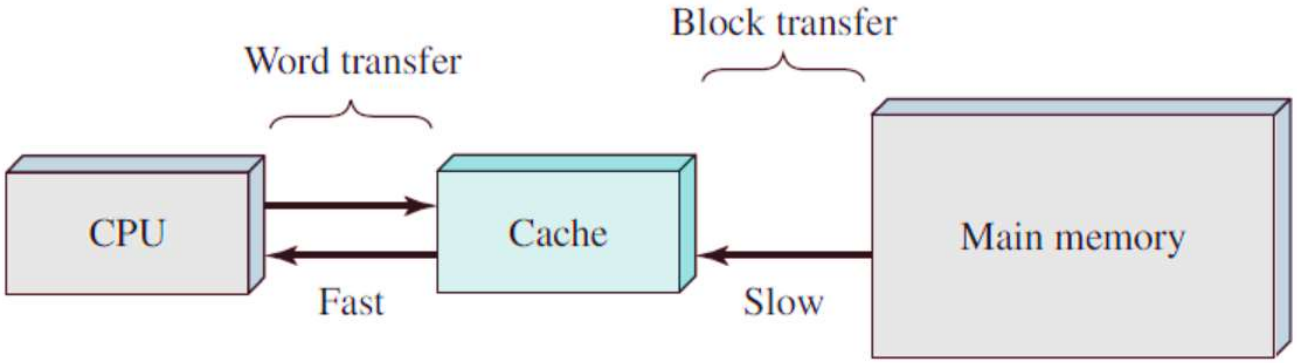
```
char a[N][M];
int sum = 0;
for (i = 0; i < N; i++)
    for (j = 0; j < M; j += K) //stride K
        sum += a[i][j];
```

- Temporal Locality
  - A memory referenced once is likely to be referenced again in the near future
- Spatial Locality
  - If a memory location is referenced, its nearby locations are likely to be referenced in the near future

Cache Memory

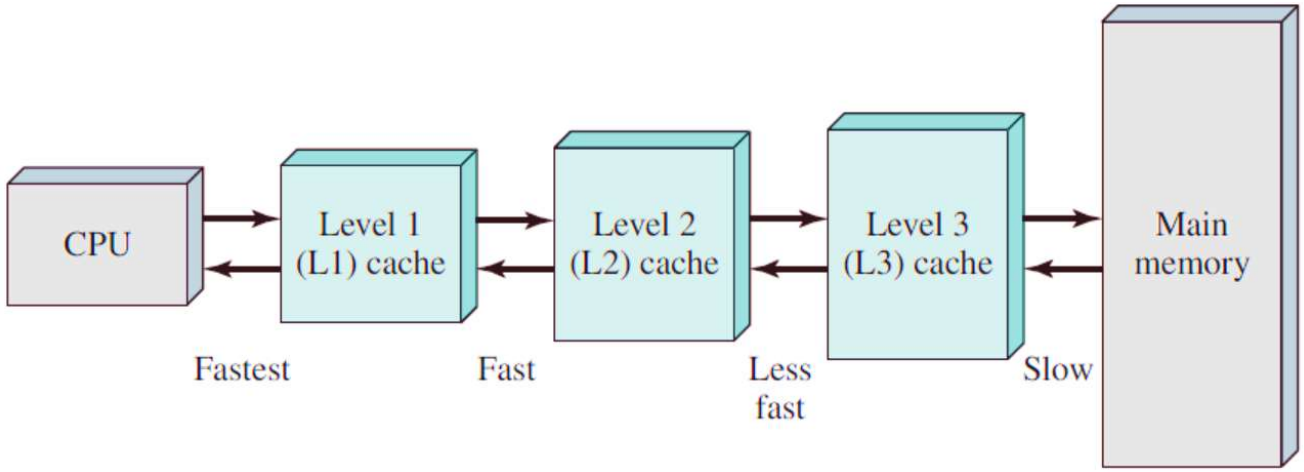
- Processors access memory frequently
  - To fetch instructions on every instruction cycle
  - To read/write data
- Processor speed is increased more rapidly than memory access speed
  - Need to trade-off among: speed, cost, and size
- Cache
  - Provides a small, fast memory between processor and main memory
  - Exploits the principle of locality

Cache Memory



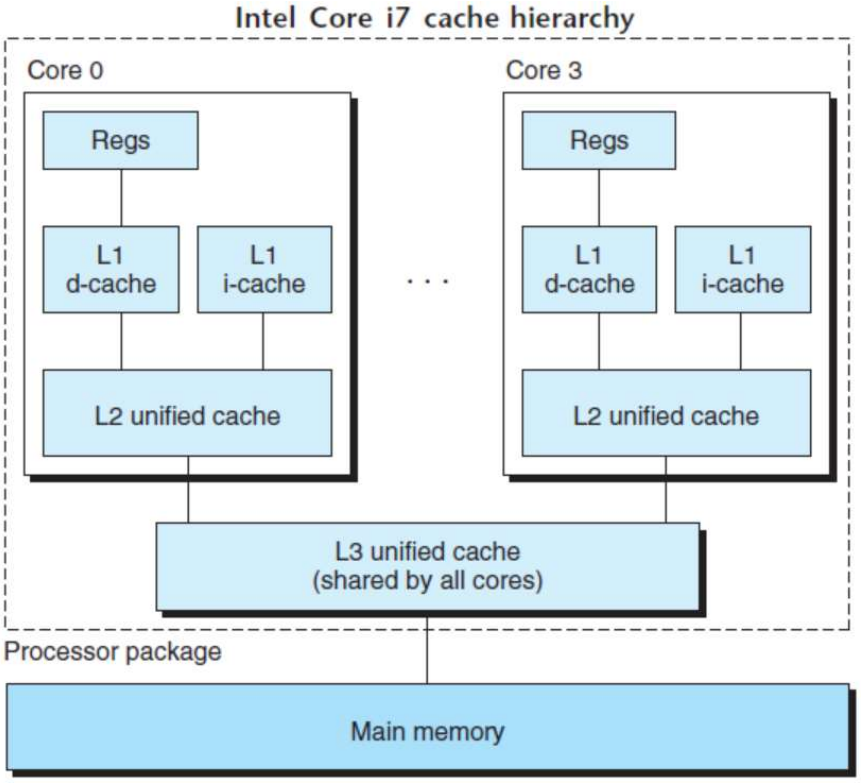
- Single cache
  - CPU reads a byte or a word
  - A block of memory is transferred from memory to cache
    - To exploit the spatial locality

Cache Memory



- Multi-level cache
  - Cost-effective way of utilizing the locality

Cache Hierarchy

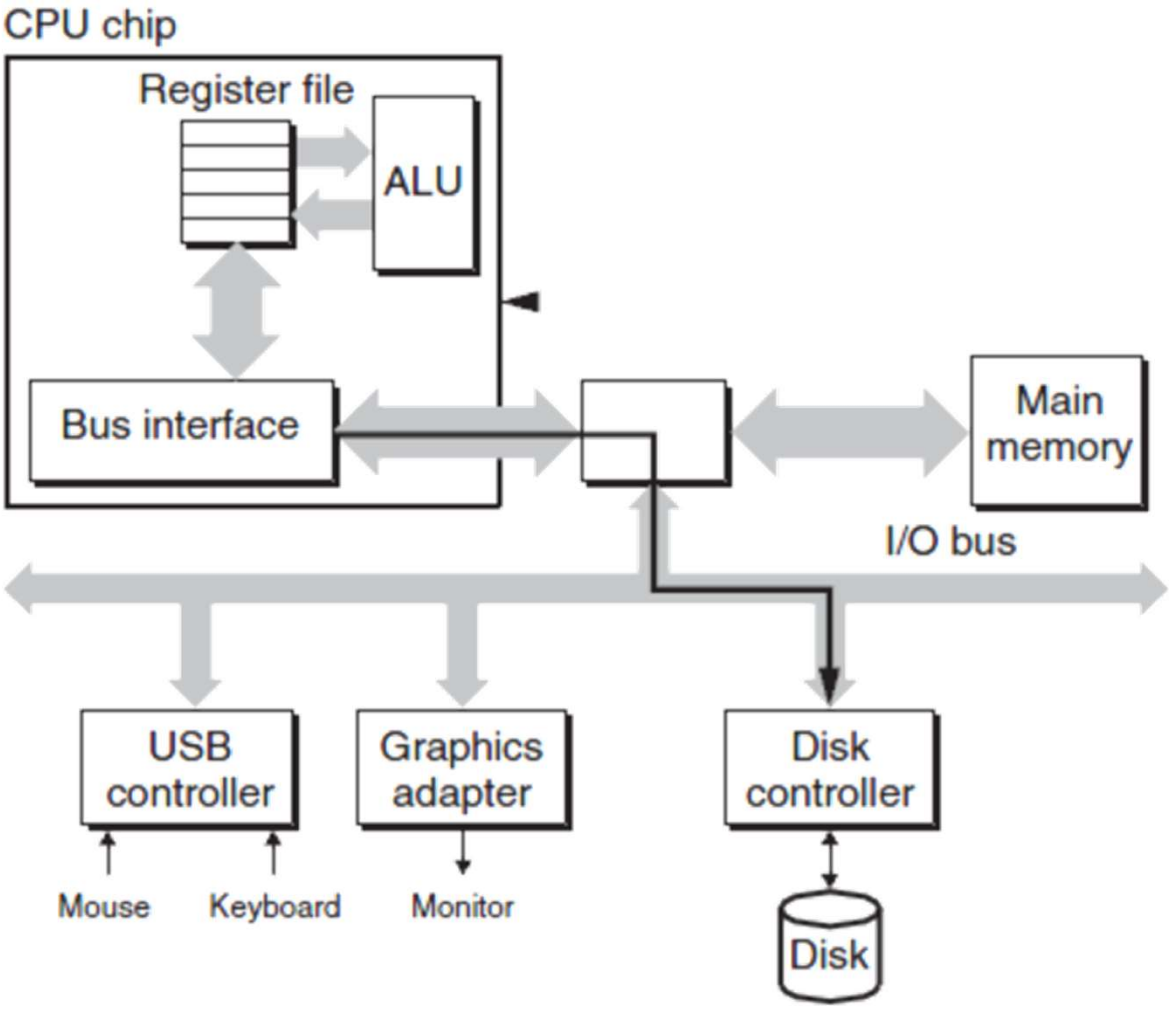


- i-cache: for instructions
- d-cache: for data
- unified-cache: for both instructions and data

Direct Memory Access

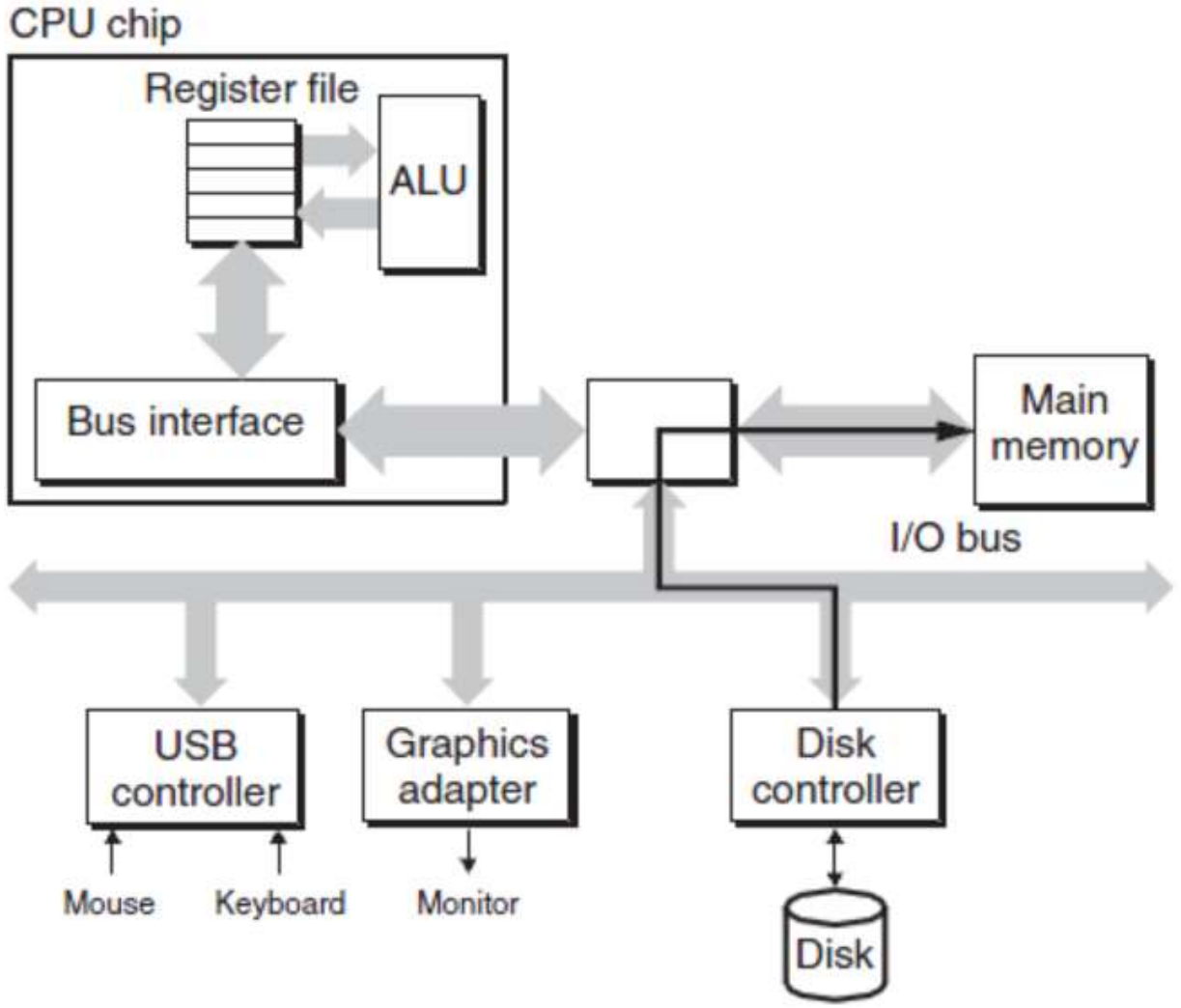
- 3 techniques for I/O operations
  - Programmed I/O
    - I/O module performs the requested action, but no more
    - Processor has to check whether I/O operation is done
  - Interrupt driven I/O
    - I/O module interrupts processor when the data is ready
  - Direct Memory Access (DMA)
    - Large volume of data is directly transferred between I/O module and memory

DMA Example: reading a disk sector



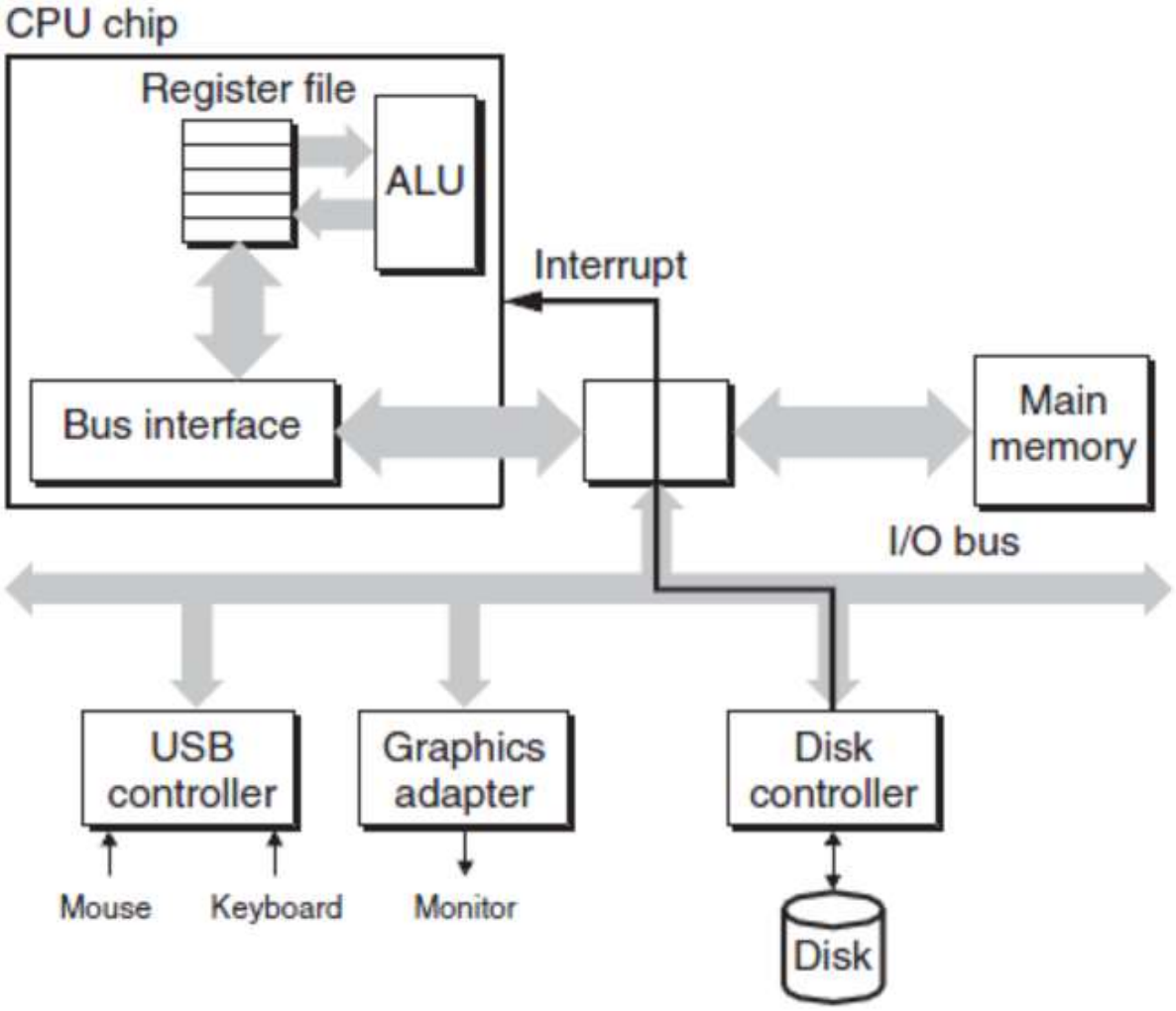
(a) The CPU initiates a disk read by writing a command, logical block number, and destination memory address to the memory-mapped address associated with the disk.

DMA Example: reading a disk sector



(b) The disk controller reads the sector and performs a DMA transfer into main memory.

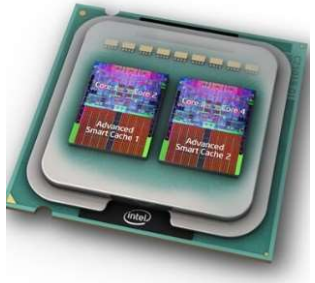
DMA Example: reading a disk sector



(c) When the DMA transfer is complete, the disk controller notifies the CPU with an interrupt.

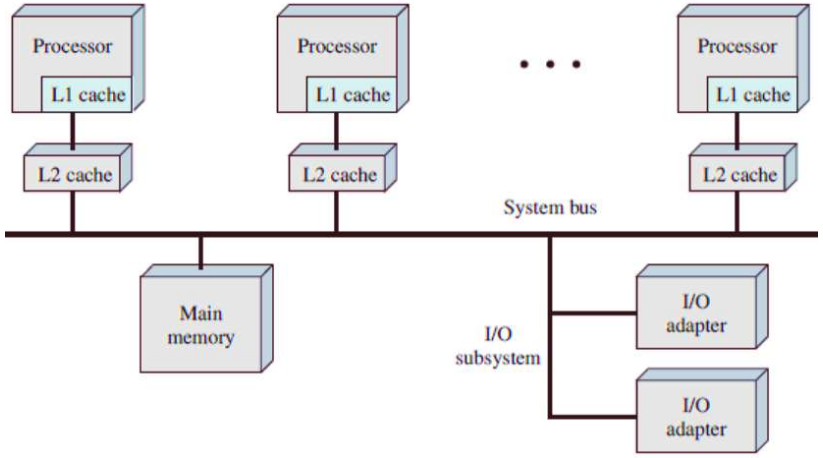
Multi-Processor and Multi-Core

- Traditional computers
  - Computers have been seen as a sequential machine



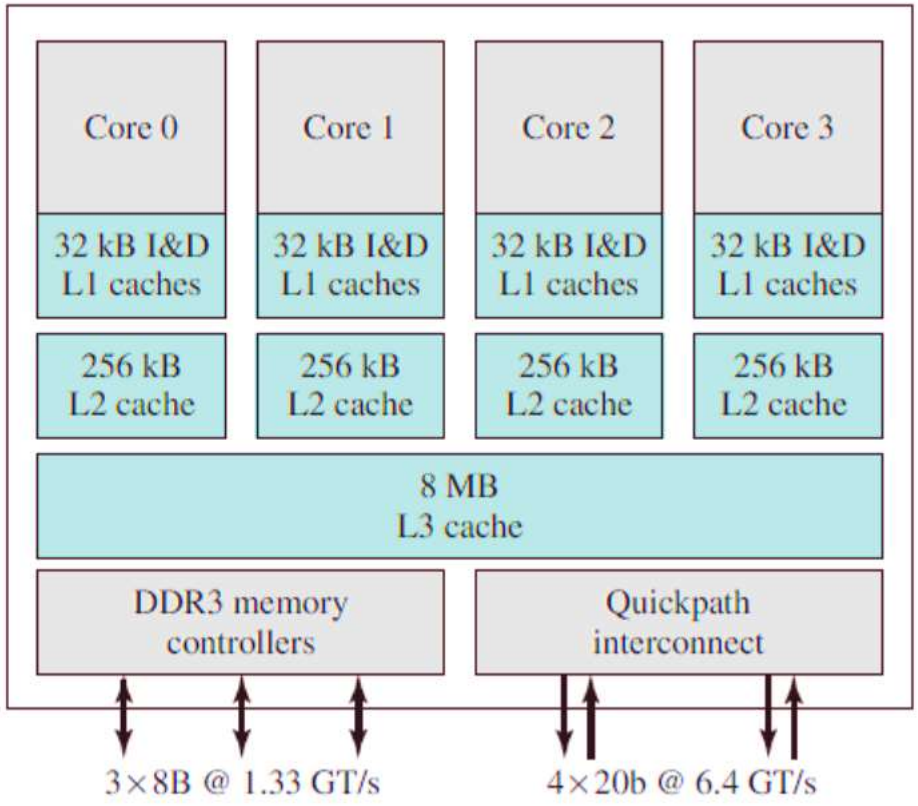
- Parallelism in computers
  - The cost of computer HW has dropped
  - To improve performance
  - To improve reliability
  - Examples: multicore computers, symmetric multiprocessors (SMP), and clusters

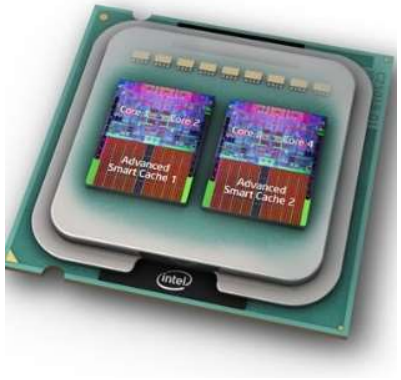
Symmetric Multiprocessors (SMP)



- Definition
  - Two or more comparable processors
  - All processors share the same memory and I/O facilities; interconnected by a bus
  - All processors share access to I/O devices
  - All processors can perform the same functions
  - The system is controlled by an integrated operating system (differentiates SMP from clusters)

Multicore Computers





Intel core i7 block diagram