Chapter 1
Computer System Overview

Contents

Operating System

- Exploits the hardware resources
  - ✓ one or more processors
  - ✓ Manages secondary memory and I/O devices
- Provides a set of services to system users

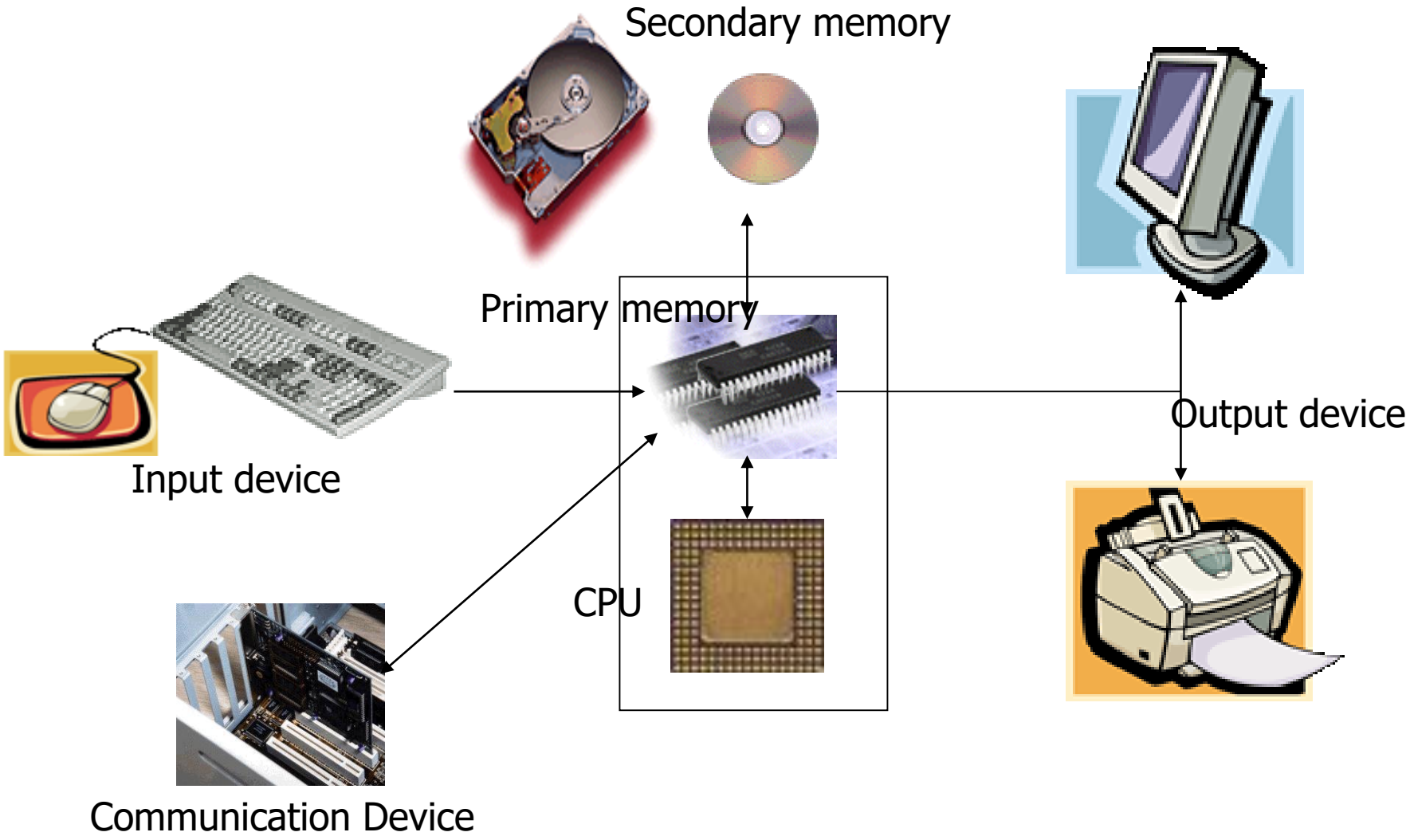
☞ need to understand the underlying computer hardware

1.1 컴퓨터 구성 요소(하드웨어)

- 처리기(중앙 처리 장치: central processing unit)
  - 컴퓨터의 두뇌 : 데이터 연산, 논리 연산 (ALU), 제어(control Unit)
  - 레지스터 (register)
  - x86, ARM, PPC, Sparc, Alpha, MIPS, Intel Core i7-8700K
- 주기억장치 (main memory) : 휘발성(volatile)
  - 메모리 셀 : 메모리 내의 개별적인 저장 공간
  - 데이터와 프로그램 저장 (실 메모리라고도 불림)
- 저장 장치 (storage device) : 비휘발성(non-volatile)
  - 디스크, CD-ROM, 플로피, Flash Memory(NOR, NAND 등)
- 입출력 장치
  - 입력 장치 (input device) : 키보드, 마우스, Key Pad, Touch Screen
  - 출력 장치 (output device) : 모니터, 프린터, LCD
- 통신 장치
  - 모뎀(modem), 이더넷(Ethernet), IrDA, CDMA, Bluetooth, WiFi

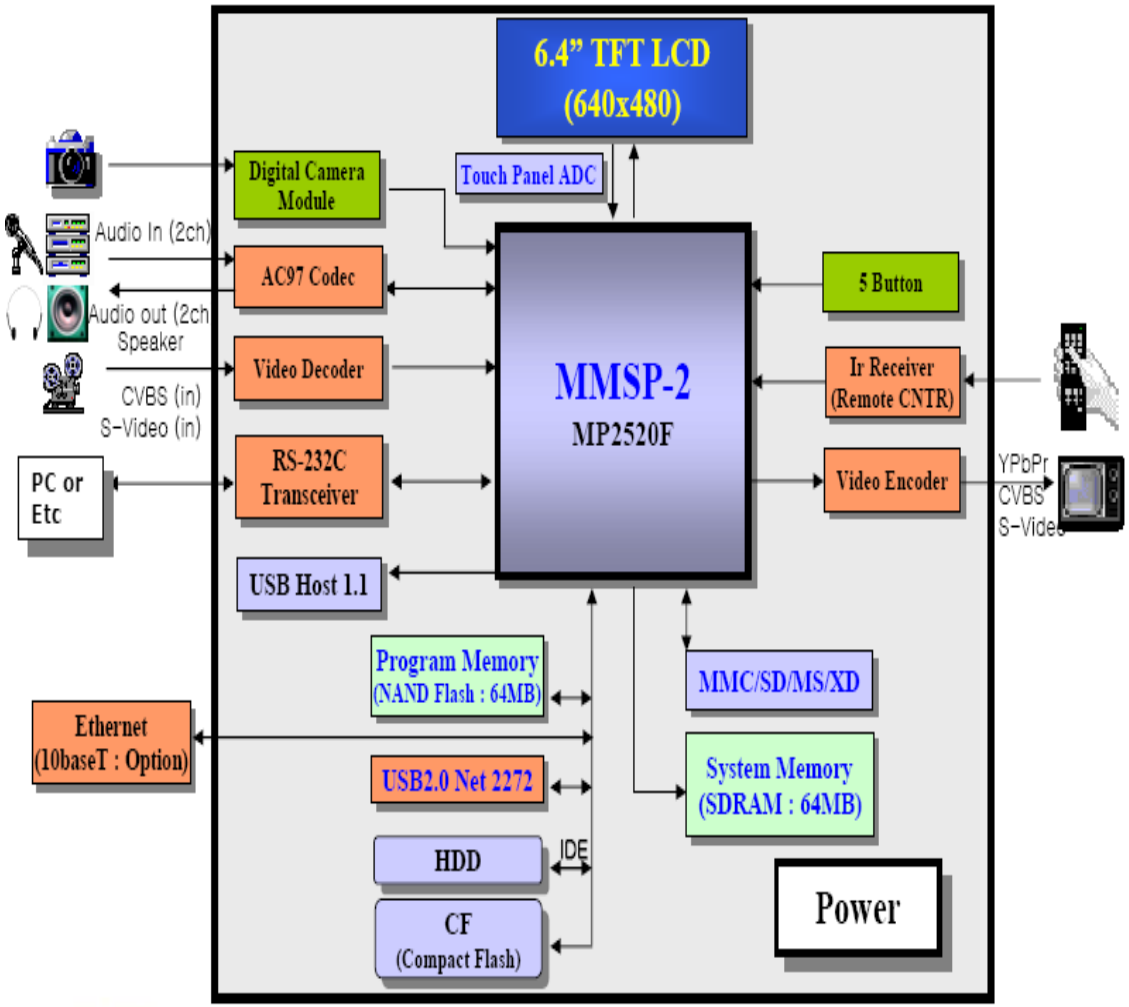
Basic Elements

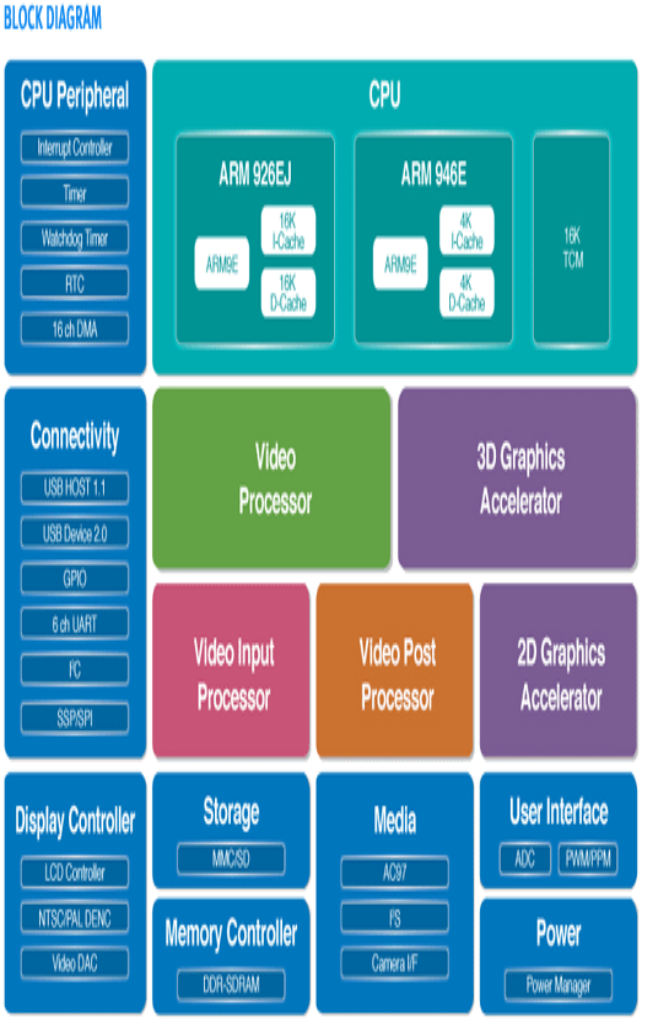
- Computer Organization: PC



Basic Elements

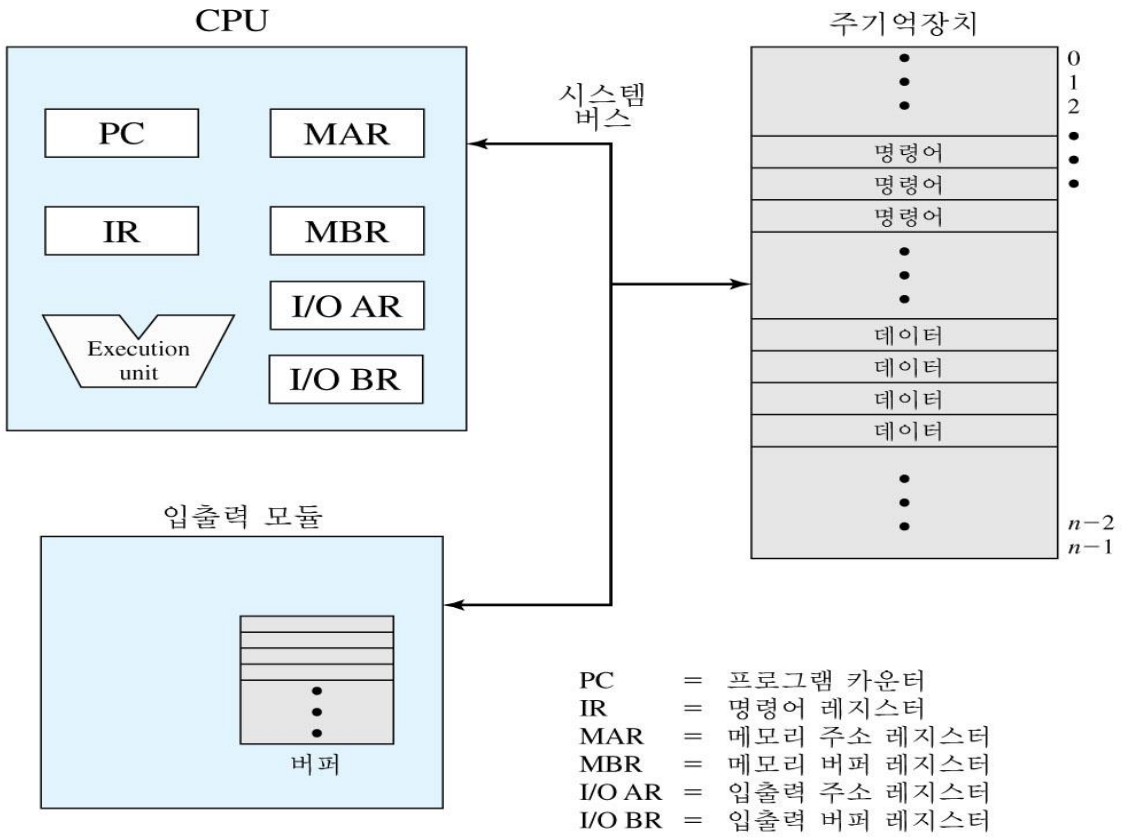
- Computer Organization: PMP examples (from MagicEye)
  - ✓ 또는 MP3 player, mobile TV 단말기





Top-Level Components

- Processor: 최상위수준 관점



Top-Level Components

- 레지스터
  - ✓ Memory address register (MAR)
    - Specifies the address for the next read or write
  - ✓ Memory buffer register (MBR)
    - Contains data written into memory or receives data read from memory
  - ✓ I/O address register (I/O AR)
  - ✓ I/O buffer register (I/O BR)
- Memory module
  - ✓ locations defined by sequentially numbered addresses
  - ✓ Each location contains a bit pattern: instruction or data
- I/O module
  - ✓ exchange data between devices and processors (memory)
  - ✓ contain internal buffers

주요 처리기 레지스터

- **제어 및 상태 레지스터 (Control and status registers)**
  - 처리기의 작동을 제어하기 위해 사용
  - 프로그램의 실행을 제어하기 위한 특권 모드의 운영체제 루틴에 의해 사용됨
  - 프로그램 계수기 (Program Counter (PC))
    - 반입할 명령어의 주소 포함
  - 명령어 레지스터 (Instruction Register (IR))
    - 최근에 반입된 명령어 포함
  - 프로그램 상태 워드 (Program Status Word (PSW))
    - 인터럽트 가능화(enable)/불능화(disable)
    - 슈퍼바이저/사용자 모드
    - 프로그램에 관련된 시스템 상태를 유지하는데 사용되는 레지스터
    - 조건 코드(condition codes) 또는 플래그
      - Positive result, Negative result, , Zero, Overflow

Processor Registers

- IA-32 Processor

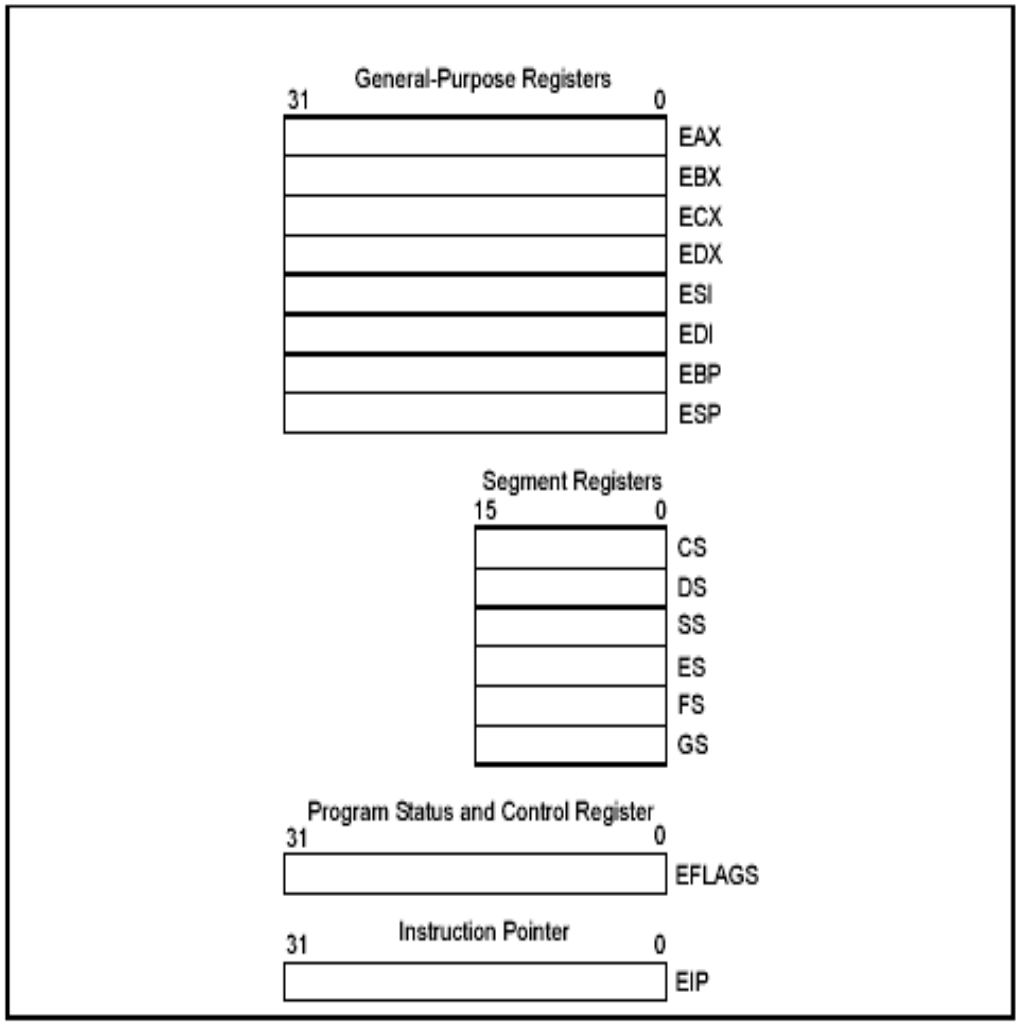
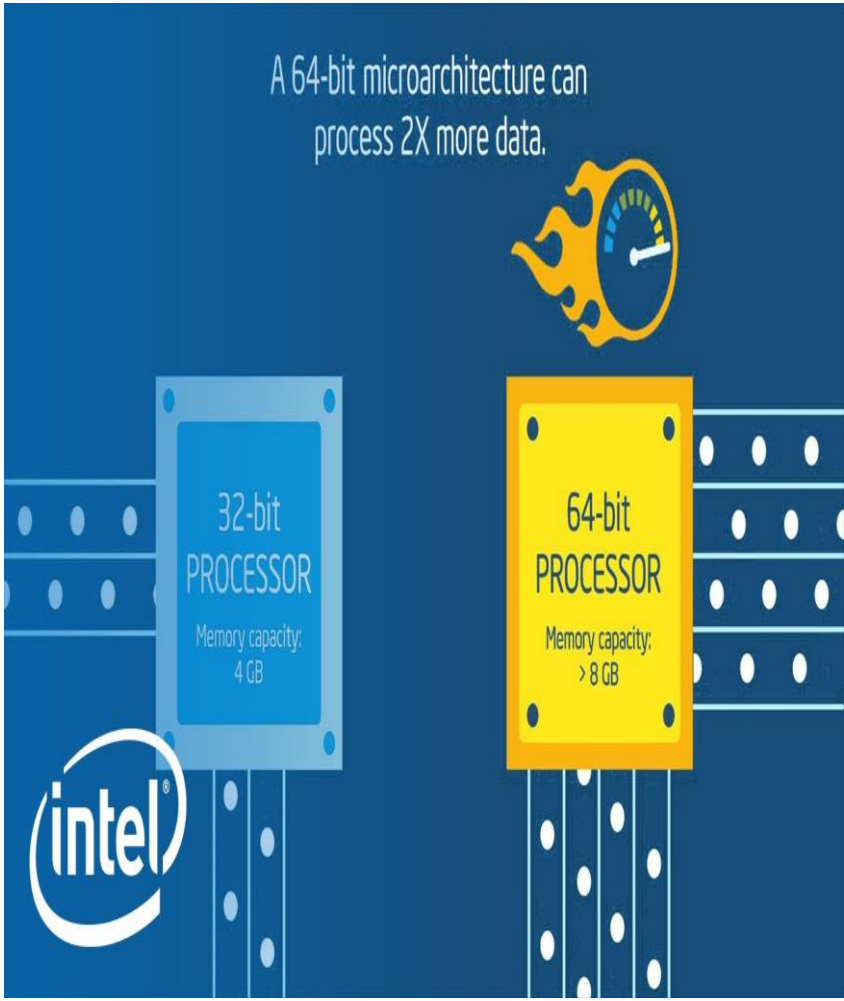


Figure 3-3. General System and Application Programming Registers

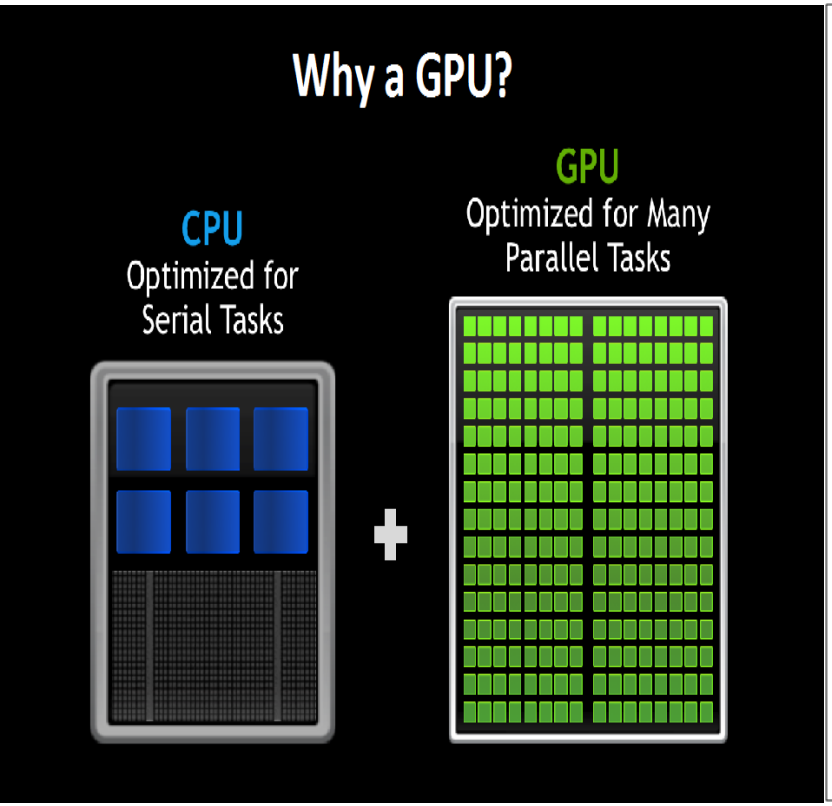


1.2 마이크로 프로세서의 발전

- **마이크로 프로세서**
  - 단일 칩에 하나의 프로세서를 집적
  - 가장 빠른 범용 프로세서
  - 멀티프로세서로 발전: 하나의 칩에 코어라고 불리는 여러 개의 프로세서 포함
- **GPU(Graphical Processing Unit)**
  - 단일명령 다중데이터(Single Instruction Multiple Data) 처리
  - 고급 그래픽 렌더링 뿐만 아니라 범용 수치처리에서도 사용
- **DSP(Digital Signal Processing)**
  - 오디오와 비디오 같은 스트림형 시그널 처리를 위한 프로세서
- **SoC(System on a Chip)**
  - CPU와 캐시뿐 아니라 DSP, GPU, I/O 디바이스, 주기억장치 등이 하나의 칩에 내장

1.2 마이크로 프로세서의 발전

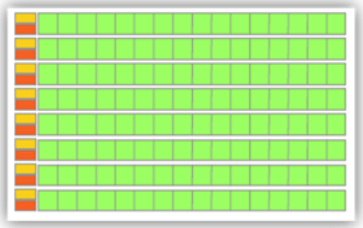
- **CPU vs GPU**



CPU

* Low compute density
* Complex control logic
* Large caches (L1$/L2$, etc.)
* Optimized for serial operations
  - Fewer execution units (ALUs)
  - Higher clock speeds
* Shallow pipelines (<30 stages)
* Low Latency Tolerance
* Newer CPUs have more parallelism

GPU



* High compute density
* High Computations per Memory Access
* Built for parallel operations
  - Many parallel execution units (ALUs)
  - Graphics is the best known case of parallelism
* Deep pipelines (hundreds of stages)
* High Throughput
* High Latency Tolerance
* Newer GPUs:
  - Better flow control logic (becoming more CPU-like)
  - Scatter/Gather Memory Access
  - Don't have one-way pipelines anymore

1.3 Instruction Execution

- Instruction
  - ✓ A program consists of a set of instructions
  - ✓ Each instruction consists of opcode, operands, and label
  - ✓ In its simplest form, instruction processing consists of 2 steps: fetch, execution
    - Fetch: Processor reads instructions from memory
    - Execution: Processor executes each instruction
- Instruction Cycle

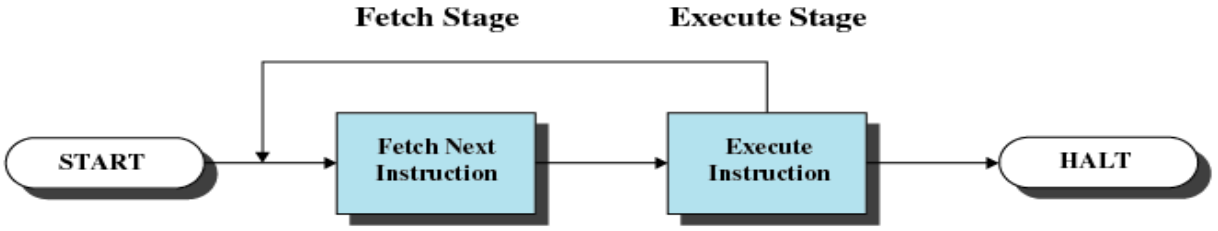


Figure 1.2 Basic Instruction Cycle

Instruction Fetch and Execute

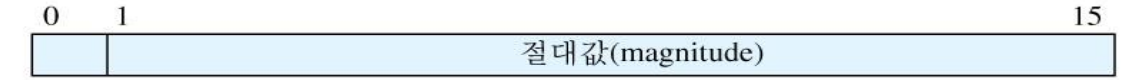
- Fetch stage
  - ✓ Program counter (PC) holds address of the instruction to be fetched next
  - ✓ Fetched instruction is placed in the Instruction register
  - ✓ Program counter is incremented after each fetch

- 명령어 범주
  - 처리기-메모리
    - 처리기와 메모리간 데이터 전달
  - 처리기-I/O
    - 주변장치로 (또는 주변장치로부터) 데이터 전달
  - 데이터 처리
    - 데이터에 산술 또는 논리 연산
  - 제어
    - 실행 순서의 변경

가상적 기계의 특성

- Execution stage: virtual 16-bit CPU

(a) 명령어 형식



(b) 정수 형식

프로그램 카운터 (PC) = 명령어 주소
명령어 레지스터 (IR) = 수행될 명령어
누산기 (AC) = 임시 저장소

(c) 내부 CPU 레지스터

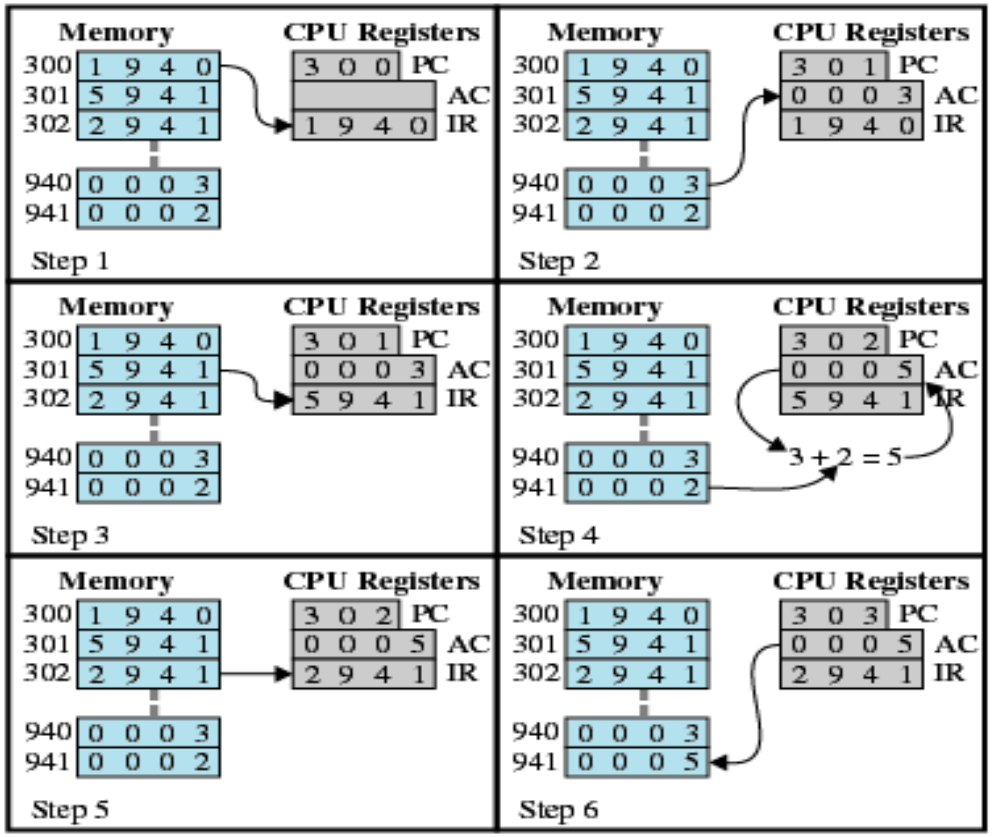
0001 = 메모리에 저장된 값을 AC에 적재
0010 = AC에 저장된 값을 메모리에 저장
0101 = 메모리에 저장된 값을 AC에 더함

(d) 연산코드 목록의 일부

Example of Program Execution

- Instruction cycle examples

  반입과 수행 단계로 구성된 명령어 사이클이 3번 필요



☞ pipeline?

Figure 1.4 Example of Program Execution
(contents of memory and registers in hexadecimal)

1.4 Interrupts

- A mechanism by which other modules may interrupt the normal sequencing of the processor
  - ✓ 인터럽트는 처리기의 정상적인 흐름을 방해
- 대부분의 I/O 디바이스는 처리기보다 느림 => I/O Bottleneck Problem
  - ✓ 처리기는 디바이스가 작업을 완료 할 때까지 기다려야 함
  - ✓ Blocking I/O의 문제점
- Classes of Interrupt

표 1.1 | 인터럽트 부류

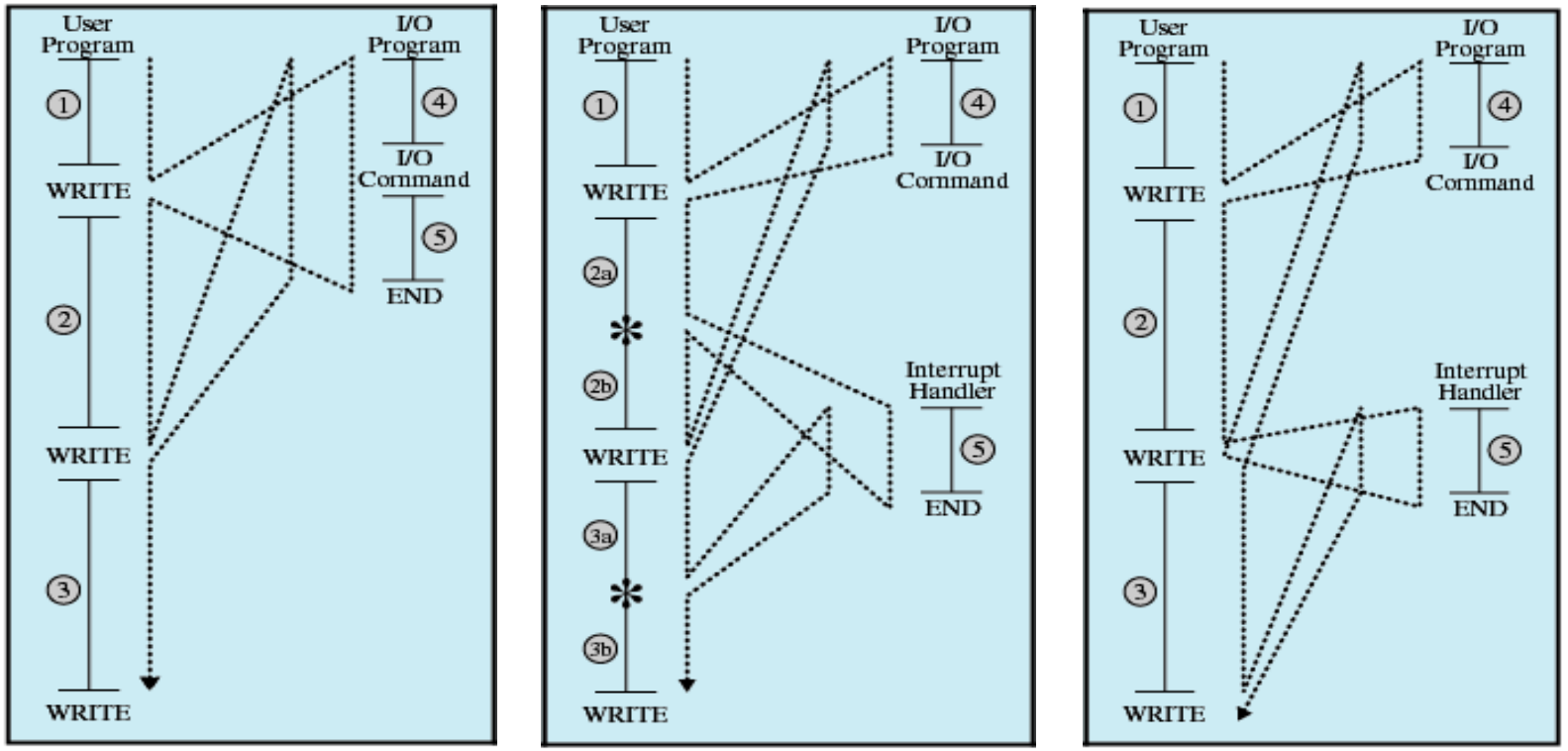
프로그램	산술연산 오버플로, 0으로 나누기, 불법적인 기계 명령어 수행 시도, 그리고 사용자에게 허용된 메모리 공간 밖의 참조 등과 같은 명령어 수행 결과로 발생하는 어떤 조건에 의해 생성된다.
타이머	처리기 내의 타이머에 의해 생성된다. 이것은 운영체제로 하여금 특정 기능을 정규적으로 수행할 수 있도록 허용한다.
입출력	입출력 제어기에 의해 생성되며, 연산의 정상적인 종료를 알리거나 다양한 에러 조건을 알려준다.
하드웨어 실패	전원 결함 또는 메모리 패리티 에러와 같은 결함에 의해 생성된다.

☞ interrupt, trap, exception

인터럽트와 프로그램 제어 흐름

- To improve processor utilization
  - ✓ Most I/O devices are slower than the processor
  - ✓ Should Processor pause to wait for device?

    -speed gap 예 ➔ CPU: 1GHz, Disk: 7200rpm (4ms), 4백만배 차이

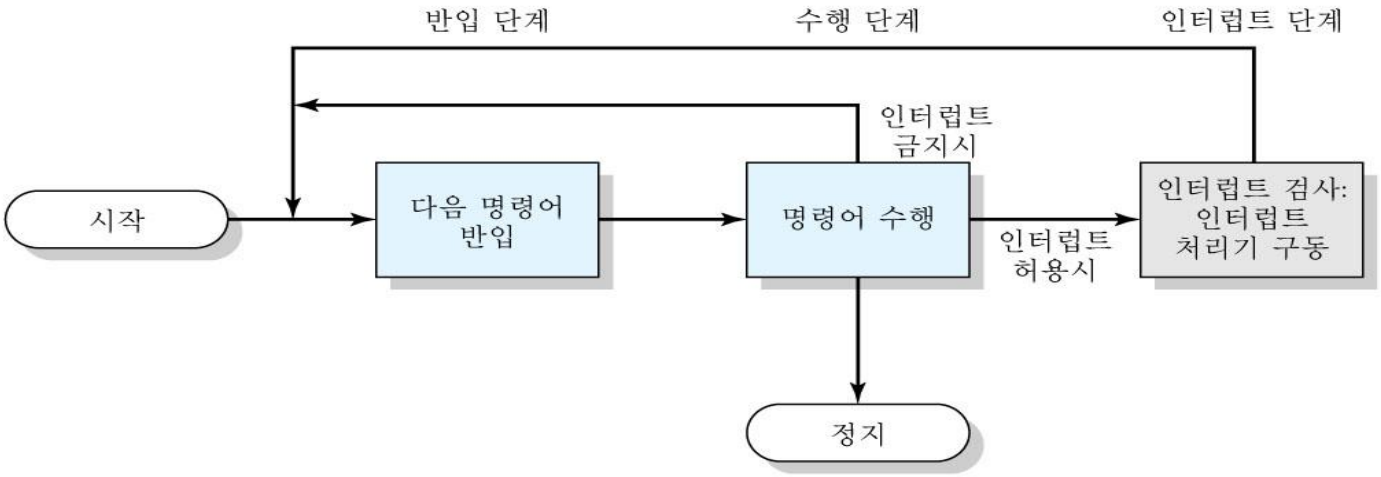


(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

☞ multiprogramming

인터럽트가 포함된 명령어 사이클

- Interrupt Stage : **처리기는 인터럽트 발생 여부 검사**
  - ✓ **인터럽트가 발생하지 않았으면 지금 수행 중인 프로그램의 다음 명령어 반입**
  - ✓ **인터럽트가 있으면, 현재 프로그램의 실행을 멈추고, 인터럽트 처리 루틴을 실행시킴**



Interrupts

- Interrupt Occur
  - ✓ suspending operation of the current program
  - ✓ branching off to a routine known as interrupt handler (contexts are saved by operating systems or hardware)

- Interrupt Handler
  - ✓ Program to service a particular I/O device
  - ✓ Generally part of the operating system

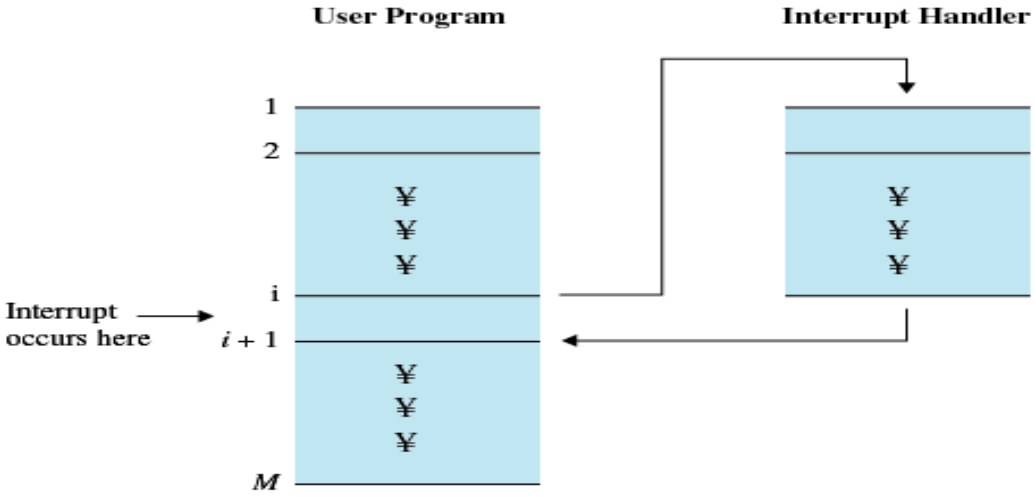


Figure 1.6 Transfer of Control via Interrupts

Timing Diagram Based on Short I/O Wait

- Performance Gain: fast devices

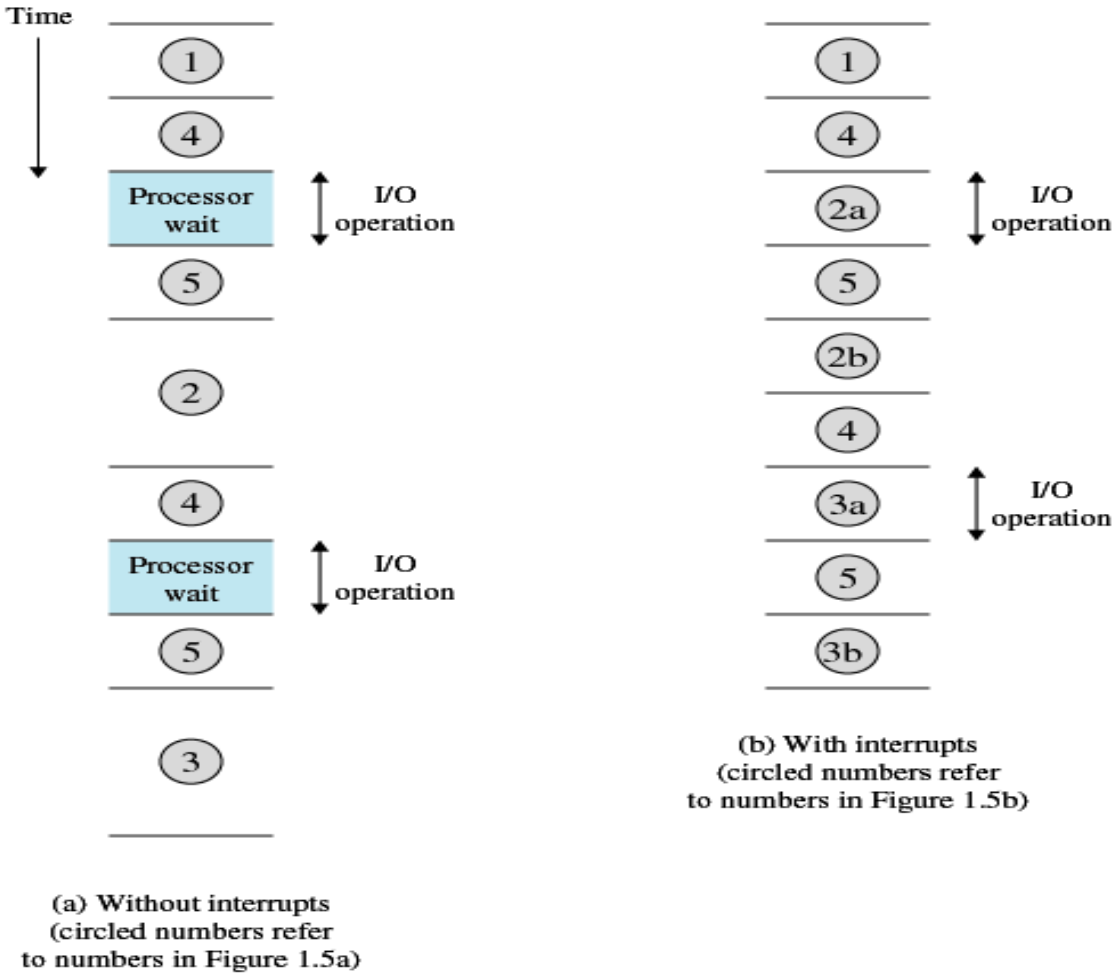


Figure 1.8 Program Timing: Short I/O Wait

Timing Diagram Based on Short I/O Wait

- Performance Gain: slow devices

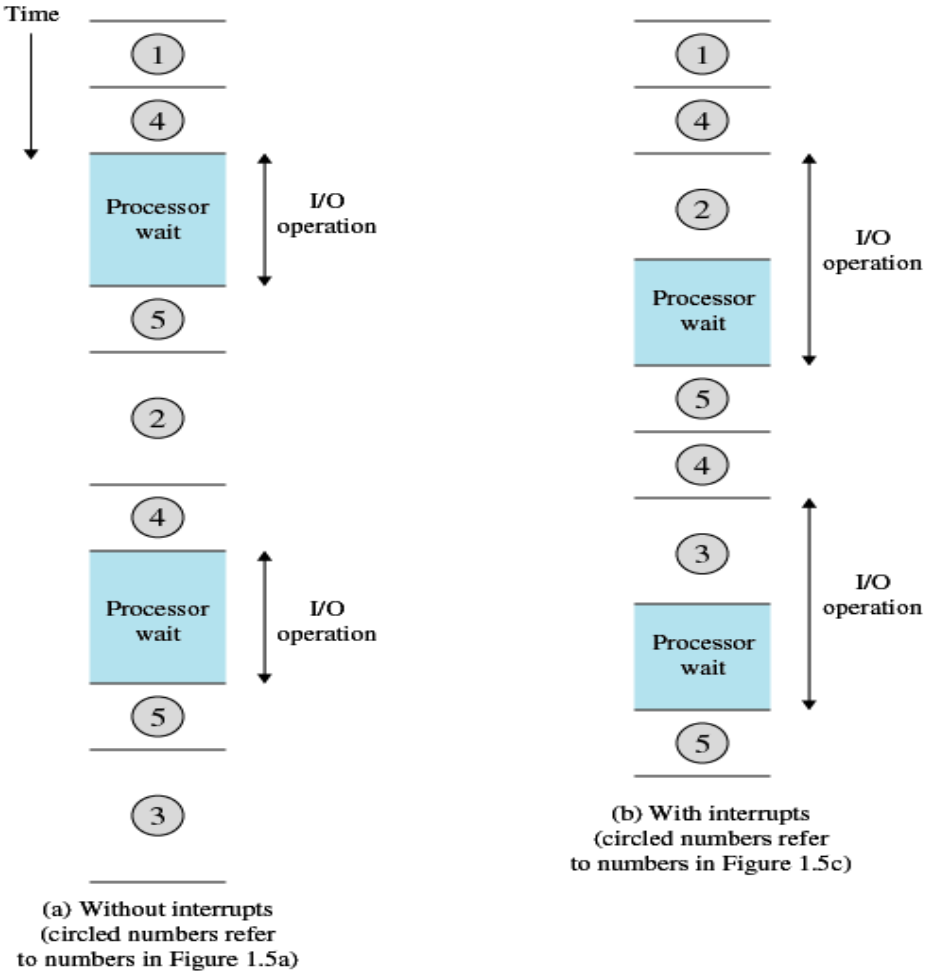
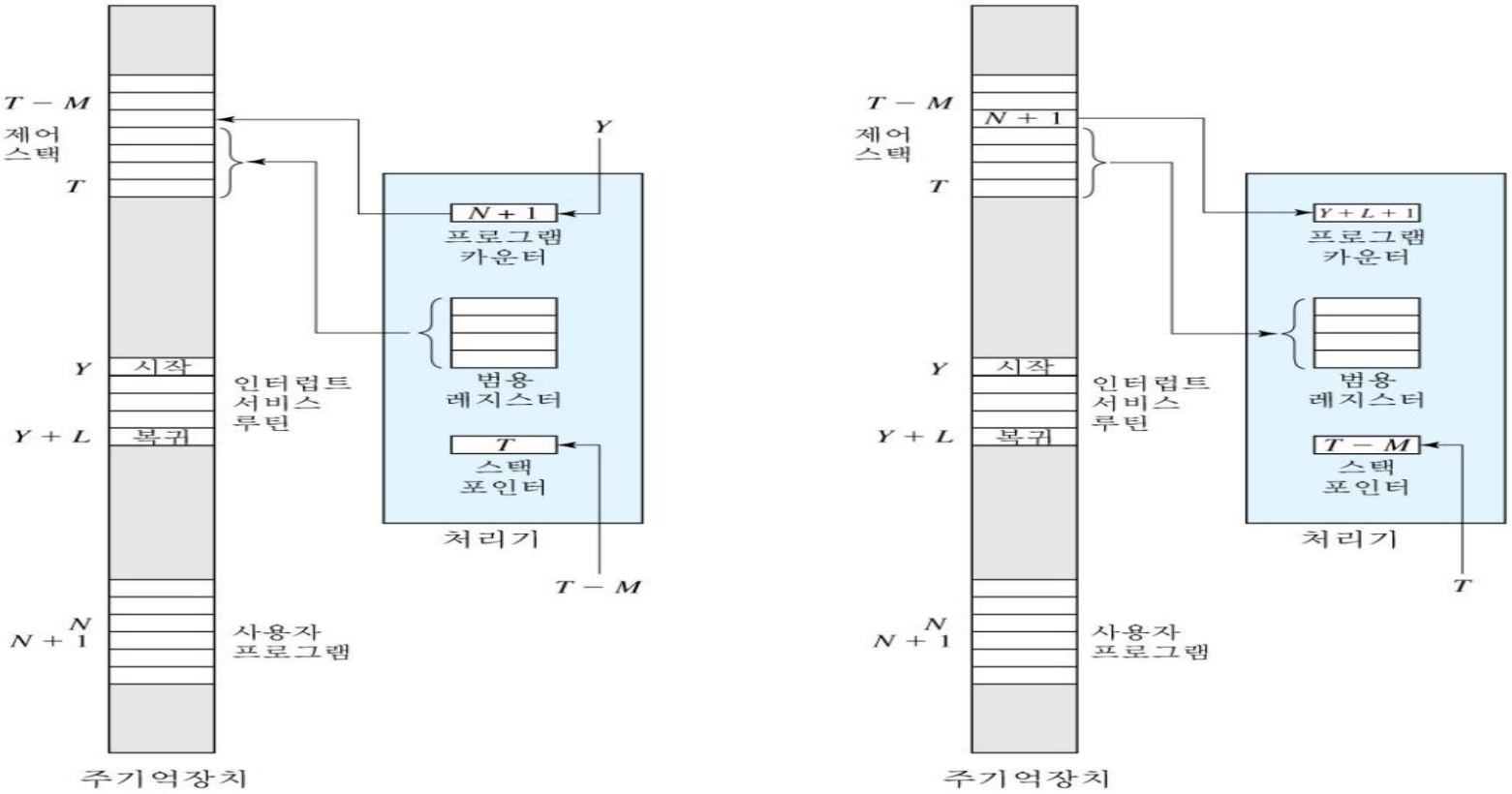


Figure 1.9 Program Timing: Long I/O Wait

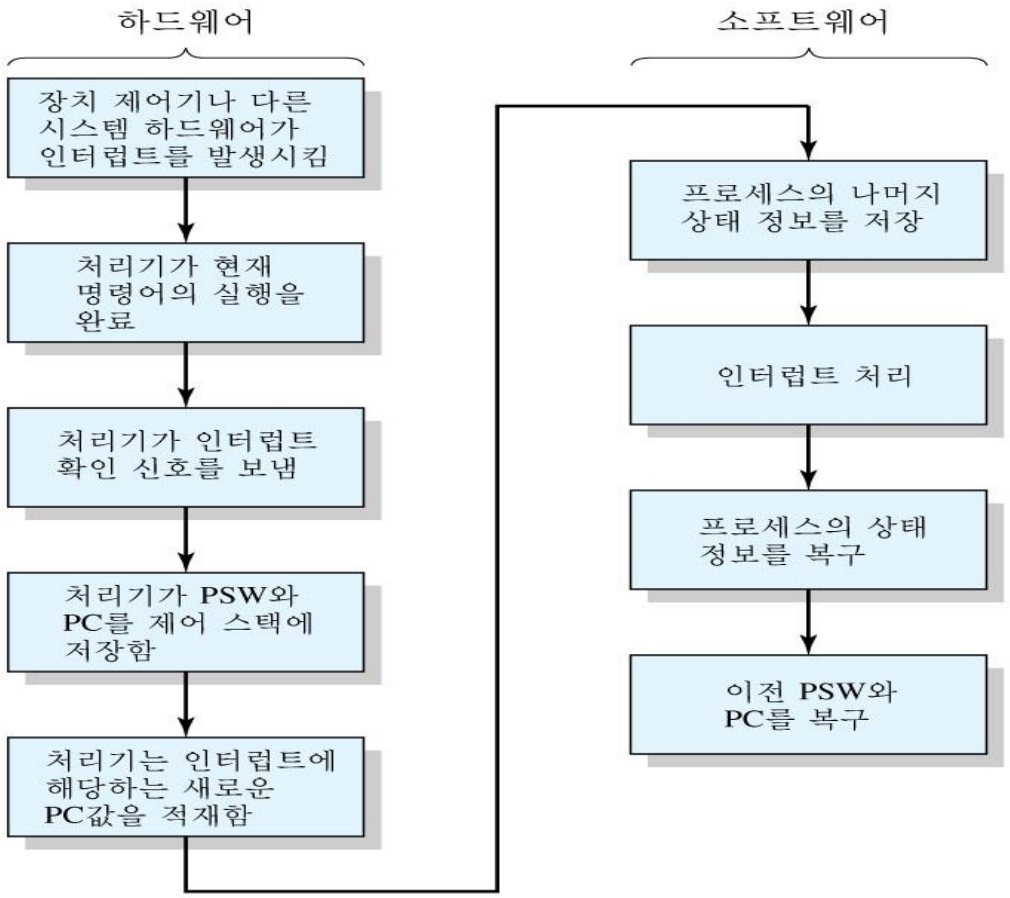
Interrupt Processing

- 인터럽트 처리를 위한 메모리와 레지스터의 상태 변화



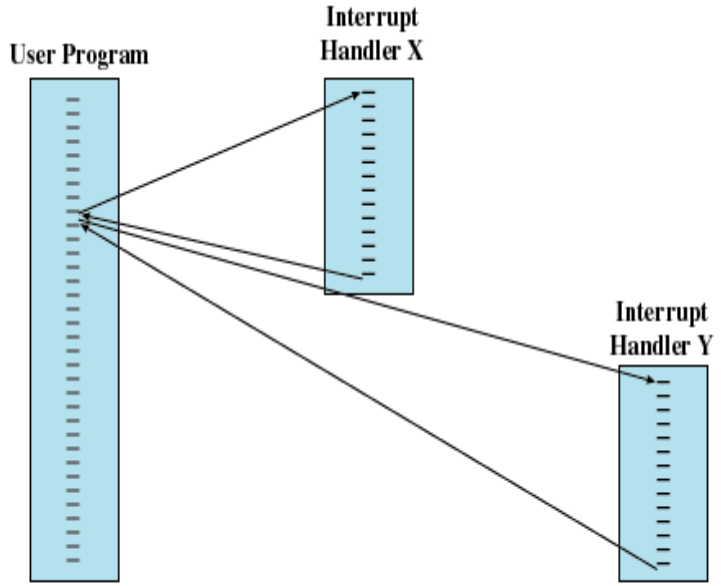
Interrupt Processing

- Typical Interrupt Processing

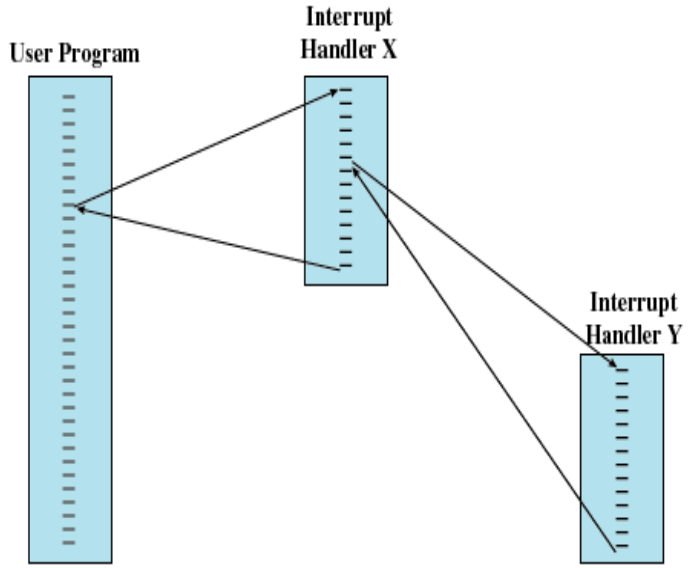


Multiple Interrupts

- It is possible for a communication interrupt to occur while a printer is generating an interrupt
- Two approach
  - ✓ Disable interrupts while an interrupt is being processed
  - ✓ Interrupt Priority 지정



(a) Sequential interrupt processing



(b) Nested interrupt processing

Multiple Interrupts

- Multiple Interrupt Handling Examples
  - ✓ Three I/O devices: printer, disk, communication with interrupt priorities of 2, 4, and 5 respectively
  - ✓ printer interrupt (t=10), communication interrupt (t=15), disk interrupt (t = 20)

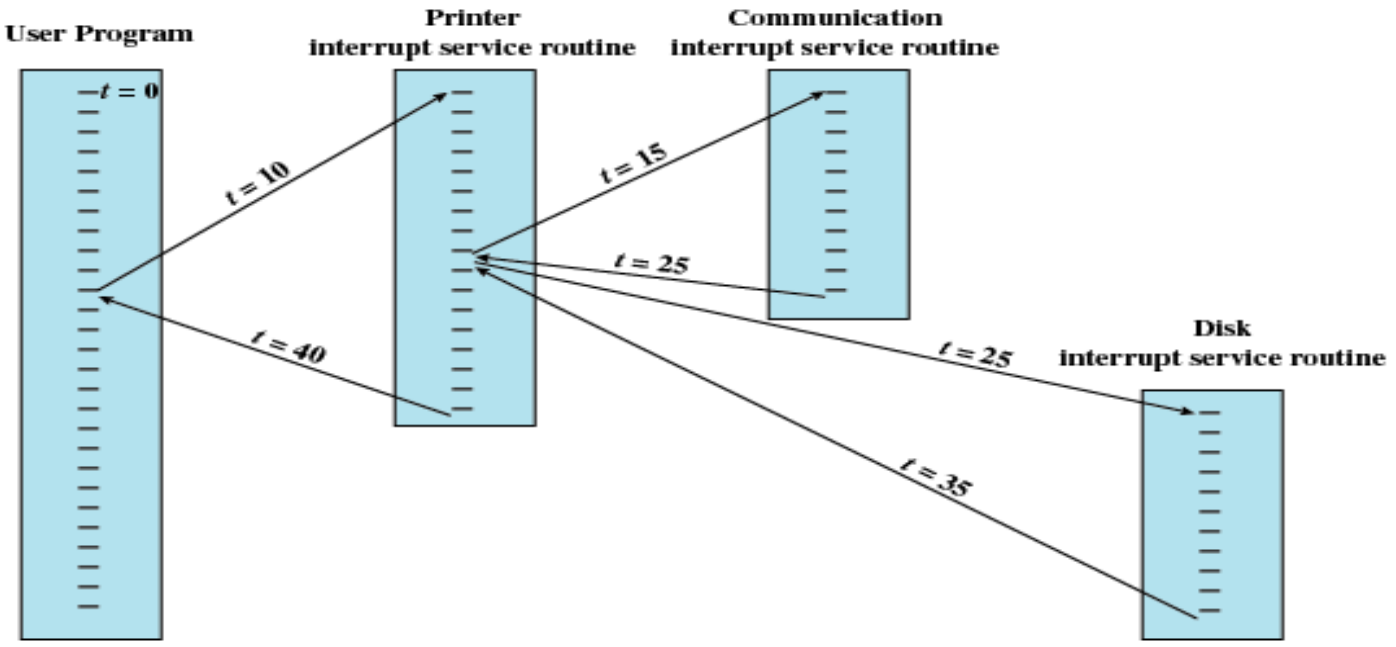


Figure 1.13 Example Time Sequence of Multiple Interrupts

Multiprogramming

- Necessity
  - ✓ If the time required to complete a I/O operation is much greater than the user code between I/O calls (a common case), the processor will be idle much of the time.

- Multiprogramming
  - ✓ 처리기는 실행 가능한 하나 이상의 프로그램을 보유
  - ✓ The sequence the programs are executed depend on their relative priority and whether they are waiting for I/O
  - ✓ 인터럽트 처리 루틴이 실행을 마치면, 인터럽트가 발생한 당시에 수행 중이던 프로그램으로 제어가 돌아가지 않을 수 있음

1.5 Memory Hierarchy

- Design constraints on a computer memory
  - ✓ How much? How fast? How expensive?

- Trade-off among the three characteristics: capacity, access time and cost
  - ✓ Faster access time, greater cost per bit
  - ✓ Greater capacity, smaller cost per bit
  - ✓ Greater capacity, slower access speed

☞ Not single memory components, but employ a memory hierarchy

Memory Hierarchy

- Memory Hierarchy – 메모리를 필요에 따라 여러 종류로 구분, 대개 CPU가 메모리를 더 빨리 접근하기 위함

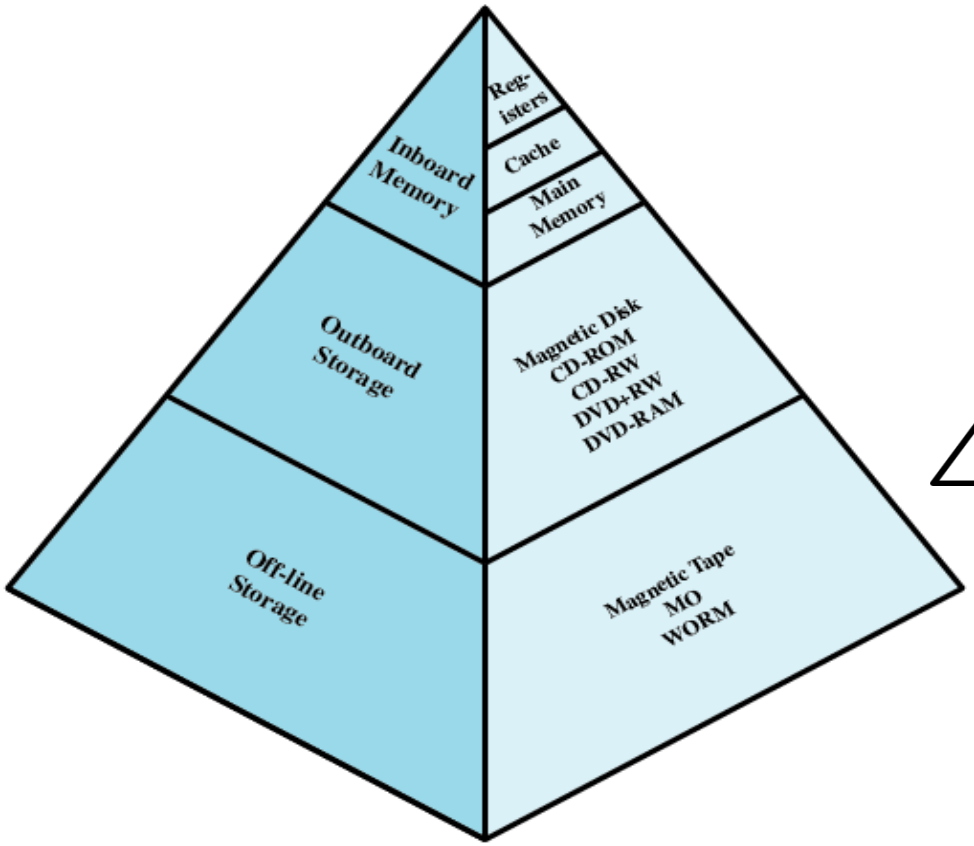
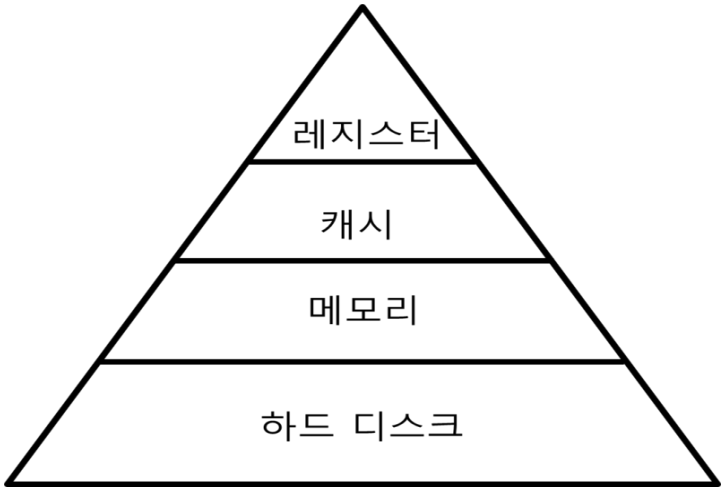


Figure 1.14 The Memory Hierarchy

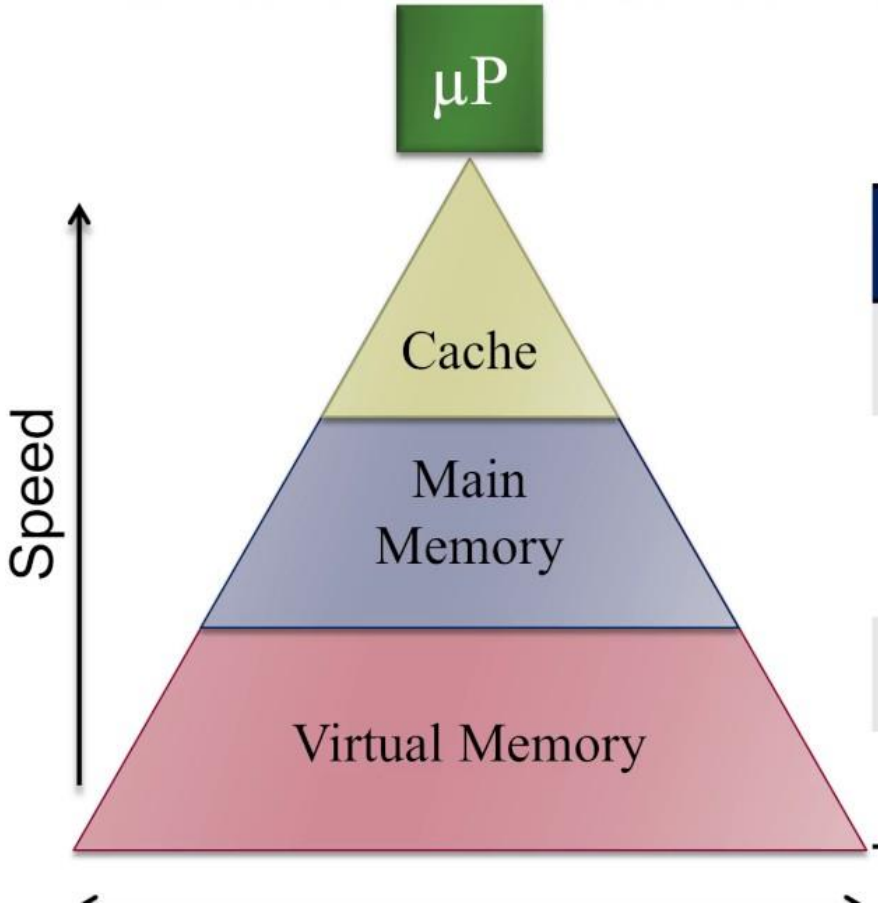


계층구조 아래로 내려갈수록

- ☞Decreasing cost per bit
- ☞ Increasing capacity
- ☞ Increasing access time
- ☞ CPU가 메모리를 접근하는 횟수가 줄어듬

Memory Hierarchy

Solution: The Memory Hierarchy

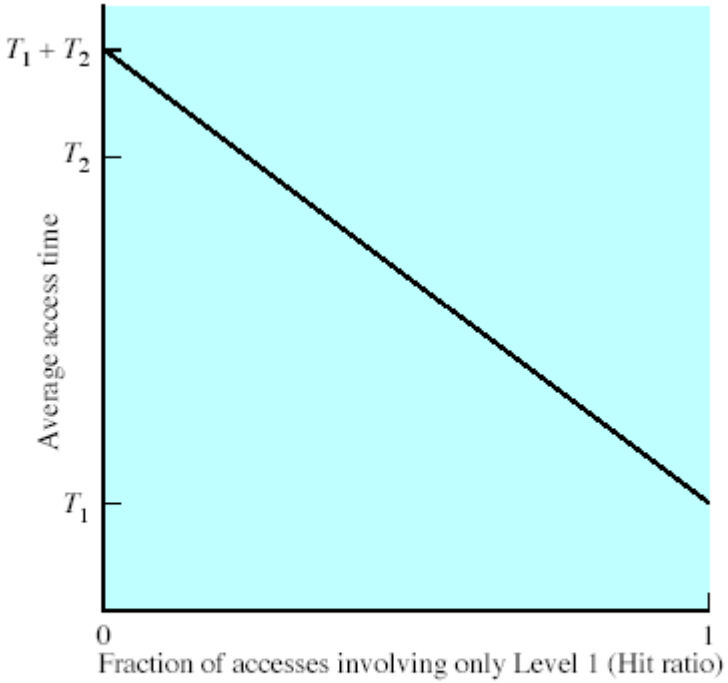


	Speed	Bit Area	Cost/GB
SRAM	~1ns	Large (≥6T)	~$10k
DRAM	~10-50ns	Small (1T & capacitor)	~$10
Disk (SSD)	~10^5ns	Very small	$1
Disk (HDD)	~10^7ns	Very small	$0.1

Numbers roughly from 2012

Memory Hierarchy

- Effectiveness of memory hierarchy



- Model
  - two levels of memory
  - level 1: 1000 bytes, 0.1us (T1)
  - level 2: 100,000 bytes, 1us (T2)
  - a byte in level 1 can be accessed directly by the processor, but a byte in level 2 first transferred to level 1 and then accessed by the processor
  - hit: the accessed data is found in level 1
- if H=0.95, then the average access time

  0.95*0.1+0.05(0.1 + 1) = 0.15μs

Figure 1.15 Performance of a Simple Two-Level Memory

☞ Hit ratio ?

Secondary Memory

- Auxiliary memory
- Disk, tape, and optical storage
- Used to store program and data files (or extension of virtual memory)

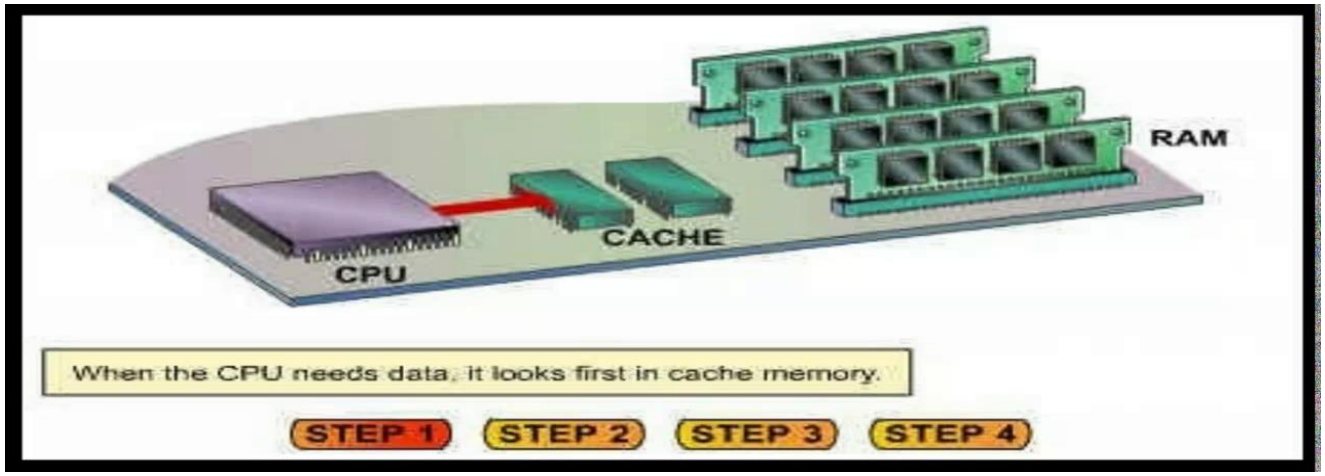
- Nonvolatile

  cf) registers, CPU cache, main memory are volatile
- Visible to the programmer in terms of files or blocks

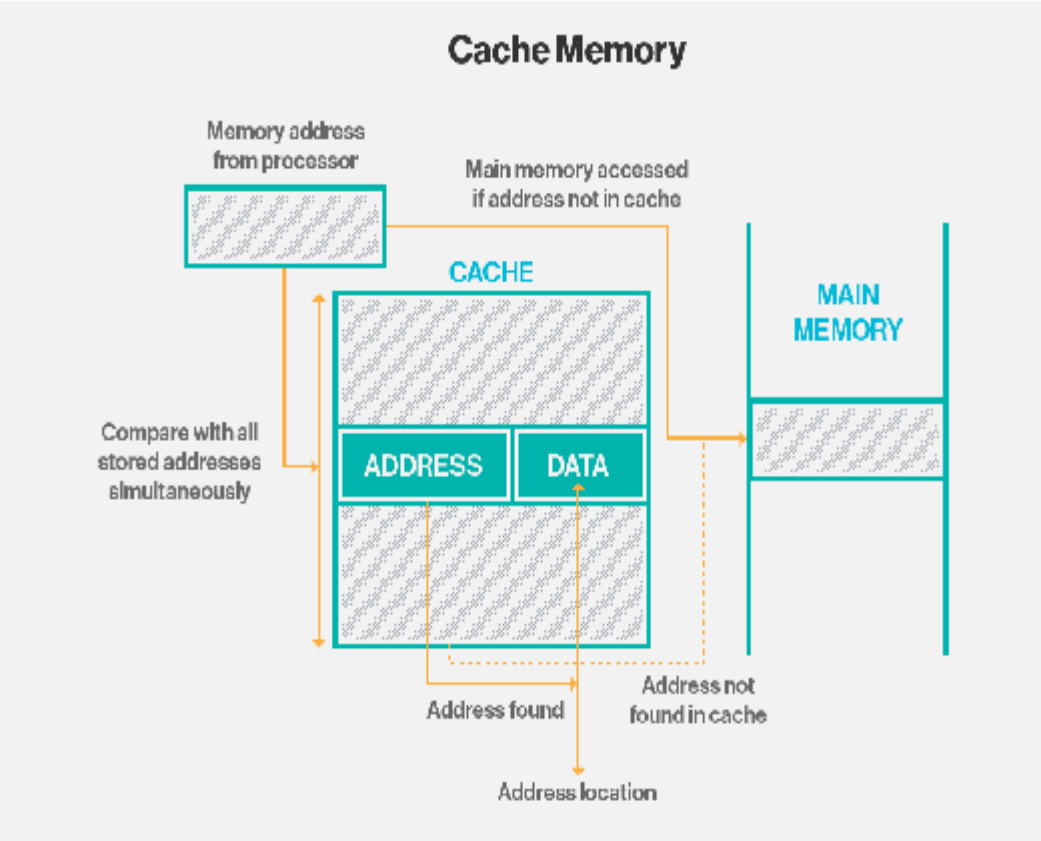
  cf) individual bytes or words

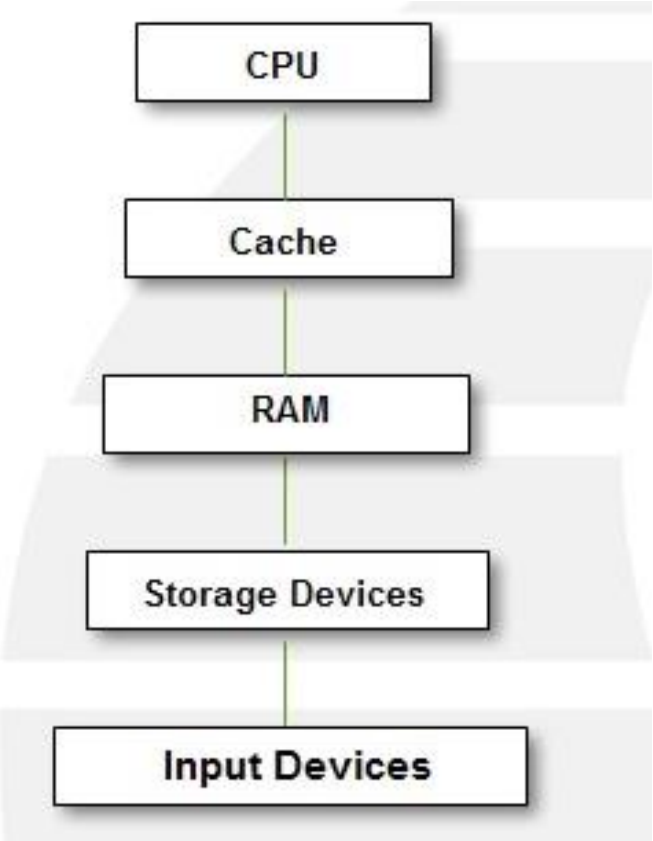
1.6 Cache Memory

- ***캐싱(Caching):***
  - ✓ ***빠른 저장 장치에 정보를 복사해 놓는 기법***
  - ✓ **고속의 메모리에 최근에 접근한 데이터를 저장**
  - ✓ **캐시 관리 정책(*cache management* policy)이 필요함.**
  - ✓ 처리기는 실행 가능한 하나 이상의 프로그램을 보유
  - ✓ **캐싱은 메모리 계층 구조에 또 다른 레벨을 도입하는데, 하나 이상의 레벨에 동시에 저장된 데이터 사이에 내용이 일치(consistent) 해야 한다**



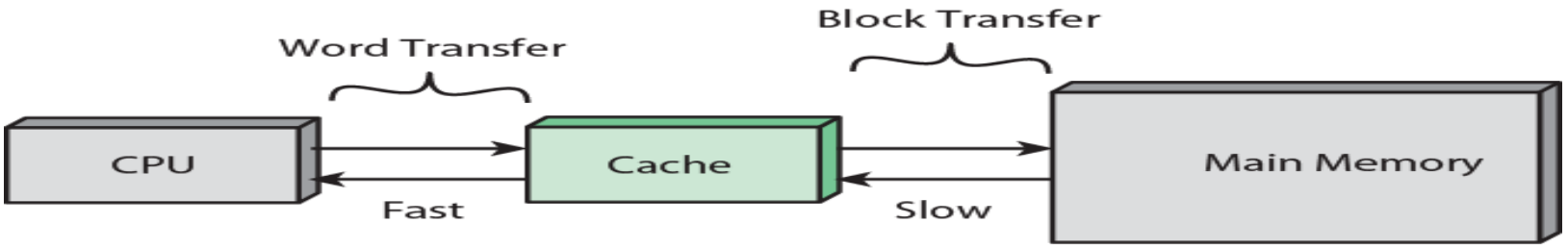
캐시 메모리


Cache Memory
Memory address from processor
Main memory accessed if address not in cache
CACHE
MAIN MEMORY
Compare with all stored addresses simultaneously
ADDRESS
DATA
Address found
Address not found in cache
Address location

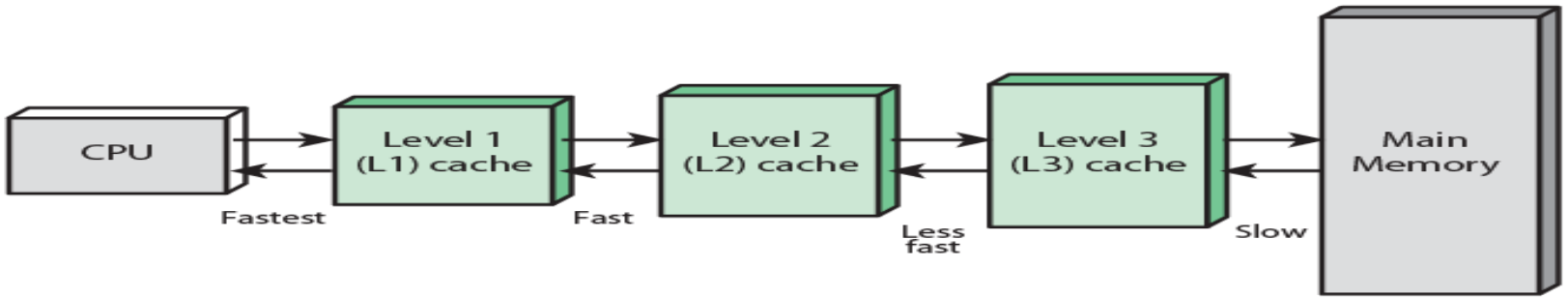

CPU
Cache
RAM
Storage Devices
Input Devices

캐시 원칙

- 주 메모리의 한 영역의 복사본을 가지고 있음
- 처리기는 먼저 캐시를 참조
- 캐시에 없으면, 필요한 정보가 저장된 메모리 블록이 캐시로 읽혀짐
- 참조 지역성 때문에 향후 참조될 데이터는 캐시에 존재할 가능성이 높음



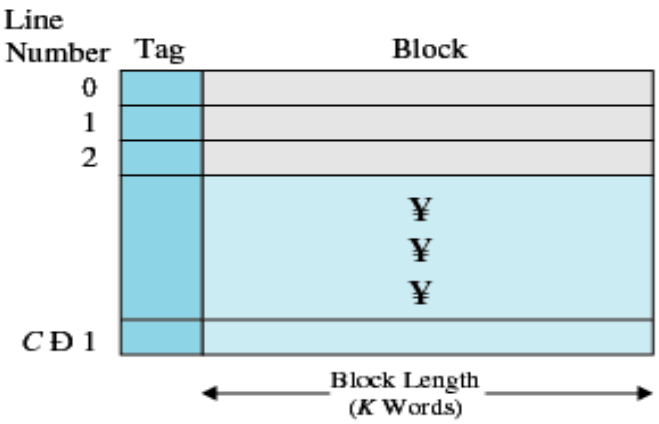
(a) Single cache

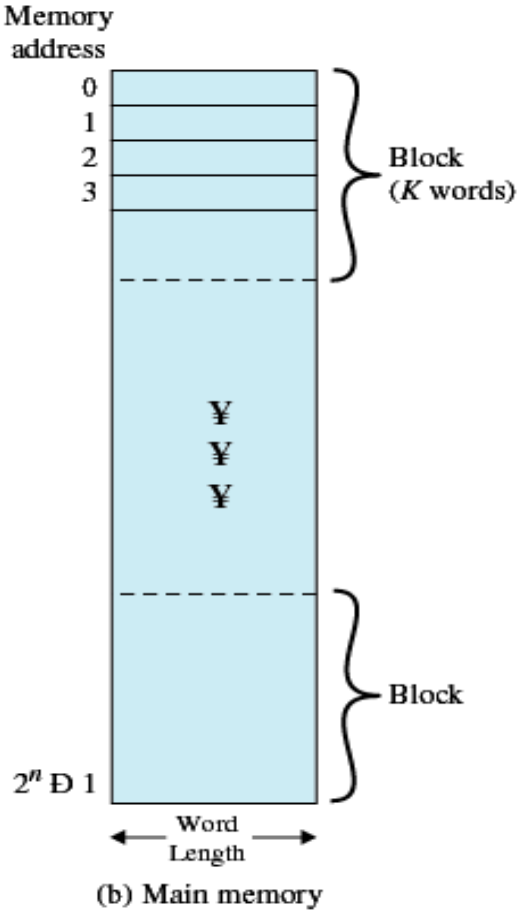


(b) Three-level cache organization

Cache/Main Memory System

- main memory: consist of up to 2^n addressable words
- cache: consists of *C* lines (slots) of *k* words each.





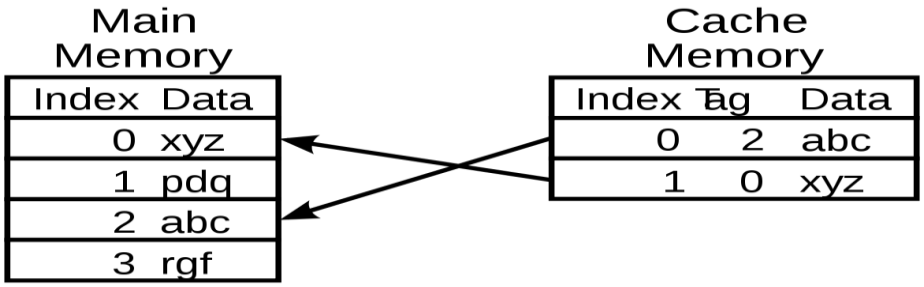
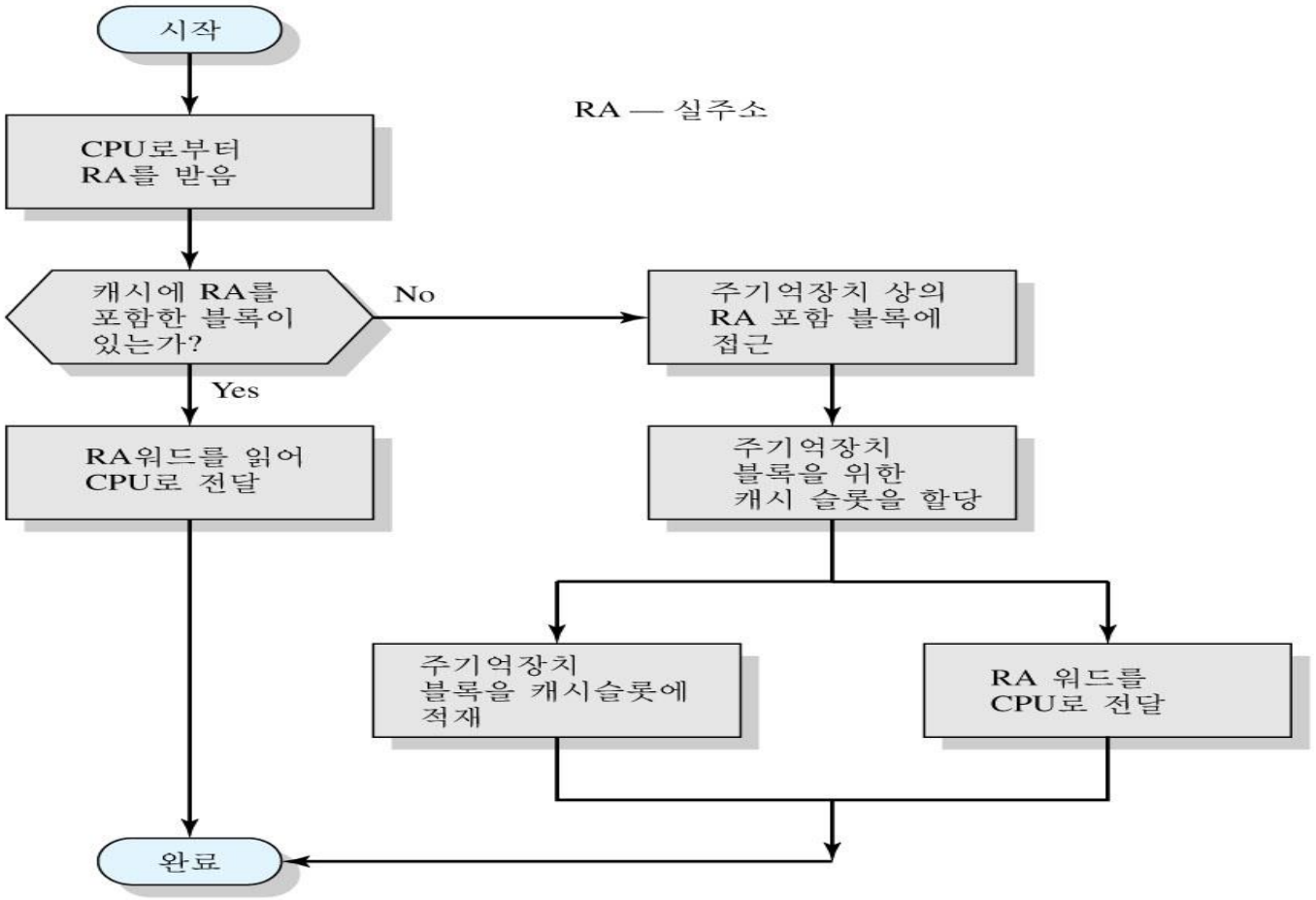


Figure 1.17 Cache/Main-Memory Structure

Cache Read Operation

- cache read operations



캐시 메모리

- **캐시 메모리 (cache memory)**
  - 운영체제에게 보이지 않고 하드웨어적으로 처리됨.
  - 메모리 접근 속도를 향상시킴
  - 프로세서의 속도는 메모리의 속도보다 빠르다.
  - 참조 지역성(locality of reference)의 원리를 이용한다.

- **디스크 캐시(Disk Cache, Buffer Cache: UNIX)**
  - 메인 메모리의 일부가 디스크에 저장된 데이터의 일부를 일시적으로 저장하기 위해서 사용됨.
  - 디스크 쓰기는 클러스터되는 경향이 있다.
  - 어떤 데이터들은 또 접근될 수 있는데, 이 경우 디스크가 아닌 버퍼 캐시로 부터 데이터를 읽어오므로 접근 속도가 빠르다.

Cache Design

- Cache size
  - ✓ Small caches have a significant impact on performance
- Block size (cache line)
  - ✓ 캐시와 메인 메모리 사이에 교환되는 데이터의 단위
  - ✓ 블록의 크기가 클수록 높은 적중률을 보임
    - 단, 새로 반입된 데이터를 참조할 확률이 캐시에서 방출된 데이터를 참조할 확률보다 적어질 때까지
- Mapping function
  - ✓ 블록이 캐시의 어느 위치에 저장될 지를 결정
- Replacement algorithm
  - ✓ 교체될 블록을 선정, Depend on Mapping functions
  - ✓ Least-Recently-Used (LRU) algorithm
- 쓰기(Write) 정책
  - ✓ 어느 시점에 메모리로 쓸 것인가를 결정
  - ✓ 블록이 갱신될 때마다 씀
  - ✓ 블록이 교체될 때마다 씀
    - 메인 메모리의 데이터가 최신이 아님

1.7 직접 메모리 접근

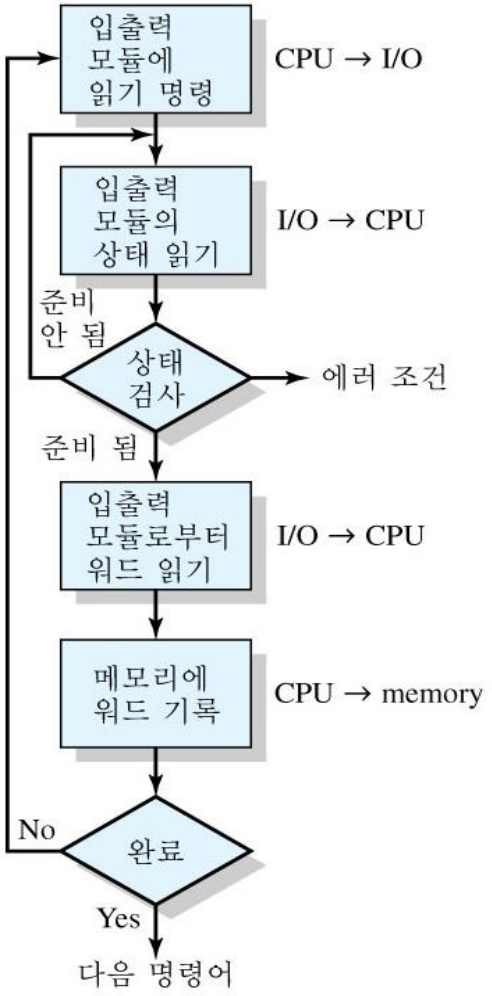
- Three techniques for I/O operations
  - ✓ Programmed I/O
  - ✓ Interrupt-driven I/O
  - ✓ DMA

프로그램된 I/O (Programmed I/O)

- I/O module
  - ✓ CPU가 아닌 입출력모듈이 수행
  - ✓ then sets appropriate bits in the I/O status register
  - ✓ takes no further action to alert processor (No interrupts occur)

- 인터럽트가 없으며, 처리기는 연산이 완료될 때까지 I/O 모듈의 상태를 주기적으로 검사

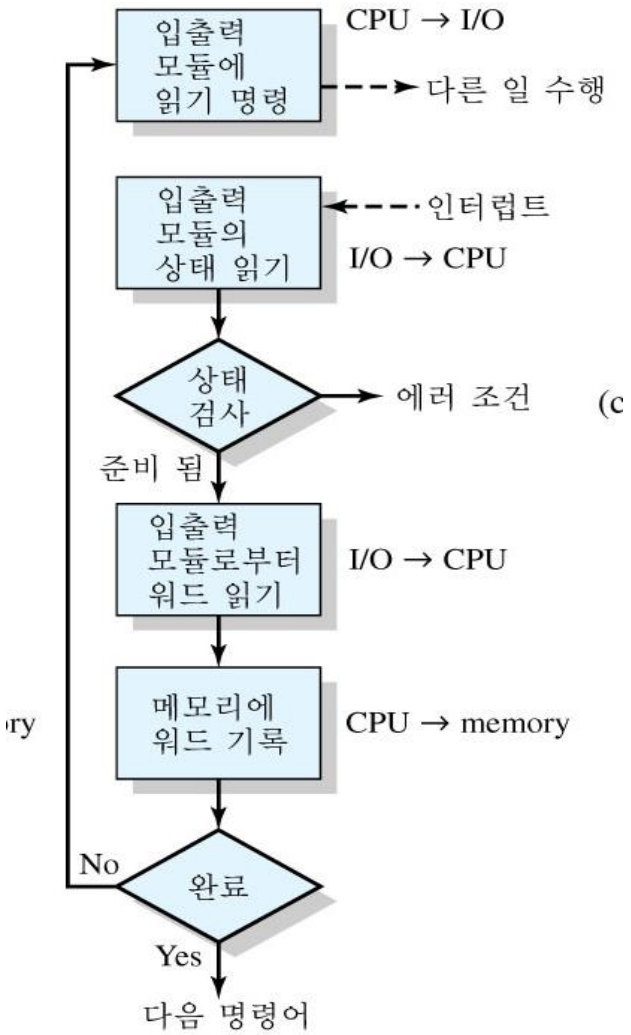
- I/O instruction categories
  - ✓ Control
  - ✓ Status
  - ✓ Transfer



(a) 프로그램된 입출력 (b)

Interrupt-Driven I/O

- No needless waiting
  - ✓ 불필요한 대기시간을 줄여서, 프로그램된 입출력보다 훨씬 효과적
- Interrupt
  - ✓ I/O 모듈이 데이터를 전달할 준비가될 때 처리기를 인터럽트
  - ✓ Processor saves context of program executing and begins executing interrupt-handler

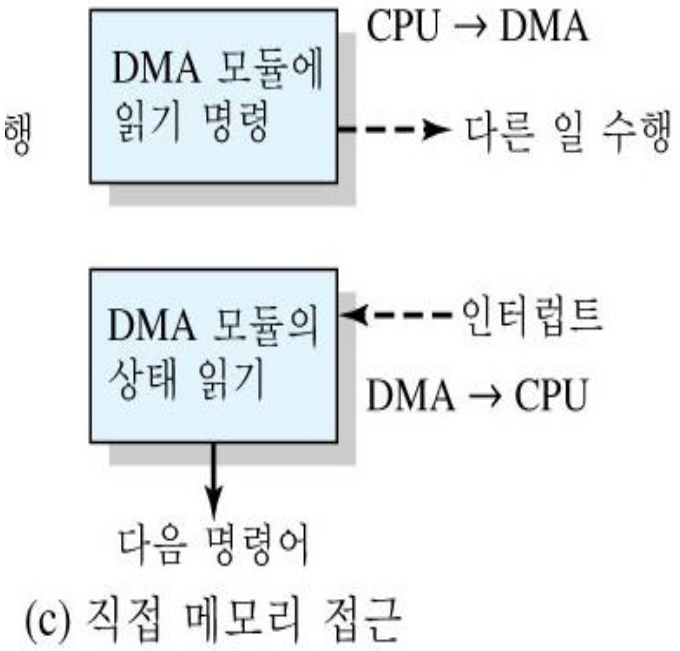


(b) 인터럽트 구동 입출력

- 메모리에서 입출력모듈로 또는 그 반대로 전송되는 데이터의 워드(word)가 CPU를 반드시 거쳐야 하므로, 여전히 CPU시간 낭비

Direct Memory Access

- CPU 개입없이 블록 단위로 대량의 전송 가능하여 시스템 전반 속도 향상 기법
- An interrupt is sent when the transfer is complete
- Processor continues with other work
- 대량의 데이터가 이동되어야 할때, 직접 메모리접근(DMA)과 같은 효율적인 기술 필요
  - ✓ DMA기능은 시스템 버스상에 있는 별도의 모듈에의해 수행되거나, 입출력모듈에 포함됨



(c) 직접 메모리 접근

- DMA command
  - ✓ 읽기요청인지 쓰기 요청인지 여부
  - ✓ 관련 I/O 장치의 주소
  - ✓ 읽거나 쓸 메모리내의 시작위치
  - ✓ 읽거나 쓸 워드의 갯수

1.8 멀티프로세서와 멀티코어 구조

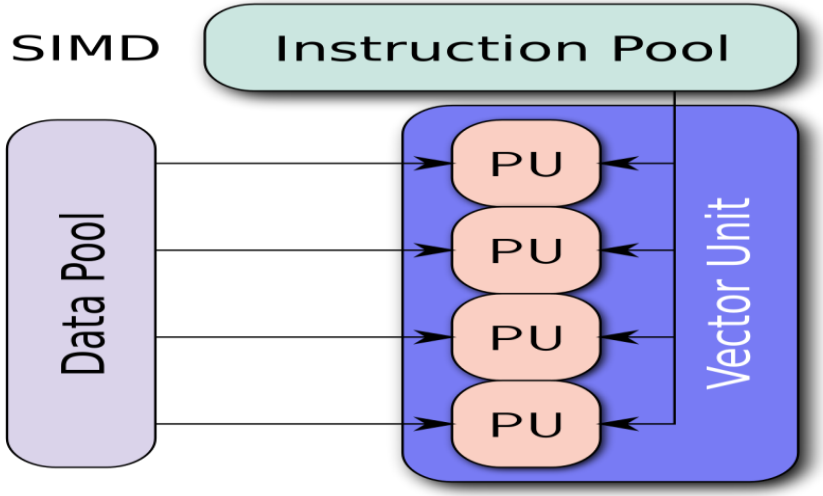
- 전통적으로 : 컴퓨터는 순차 기계 (sequential machine)로 알려짐
  - ✓ 프로세서가 순차적으로 한번에 하나씩 명령을 수행
  - ✓ 병렬적으로 프로그램을 수행하는 방법

- 병렬성(parallelism)을 제공하는 일반적인 방법
  - ✓ 대칭형 멀티프로세서: Symmetric MultiProcessors (SMPs)
  - ✓ 멀티코어
  - ✓ 클러스터 (16장)

대칭적 다중처리

- 컴퓨터 시스템의 분류 (by Flynn)
  - ✓ Single Instruction Single Data (SISD) stream
    - 단일 처리기가 한 메모리에 저장된 데이터를 처리하기 위해 단일 명령 스트림을 수행
  - ✓ Single Instruction Multiple Data (SIMD) stream
    - 각 (동일) 명령이 서로 다른 데이터 집합에 대하여 서로 다른 처리기에 의해 수행 ->하나의 명령어로 여러 개의 값을 동시에 계산하는 방식.
    - 벡터 및 배열 처리기 (vector and array processors)
    - Pentium 처리기의 superscalar 구조
  - ✓ Multiple Instruction Single Data (MISD) stream
    - (같은) 일련의 데이터가 처리기들의 집합에 전송되고, 각 처리기는 서로 다른 명령을 수행
    - 지금까지 구현된 적이 없음
  - ✓ Multiple Instruction Multiple Data (MIMD) stream
    - 다수의 처리기가 서로 다른 데이터 집합에 대하여 서로 다른 일련의 명령어들을 동시에 수행

병렬컴퓨터의 SIMD 구조



Four summations (instructions)

a	+	a	=	2a
b	+	b	=	2b
c	+	c	=	2c
d	+	d	=	2d

vs.

SIMD one summation (instruction)

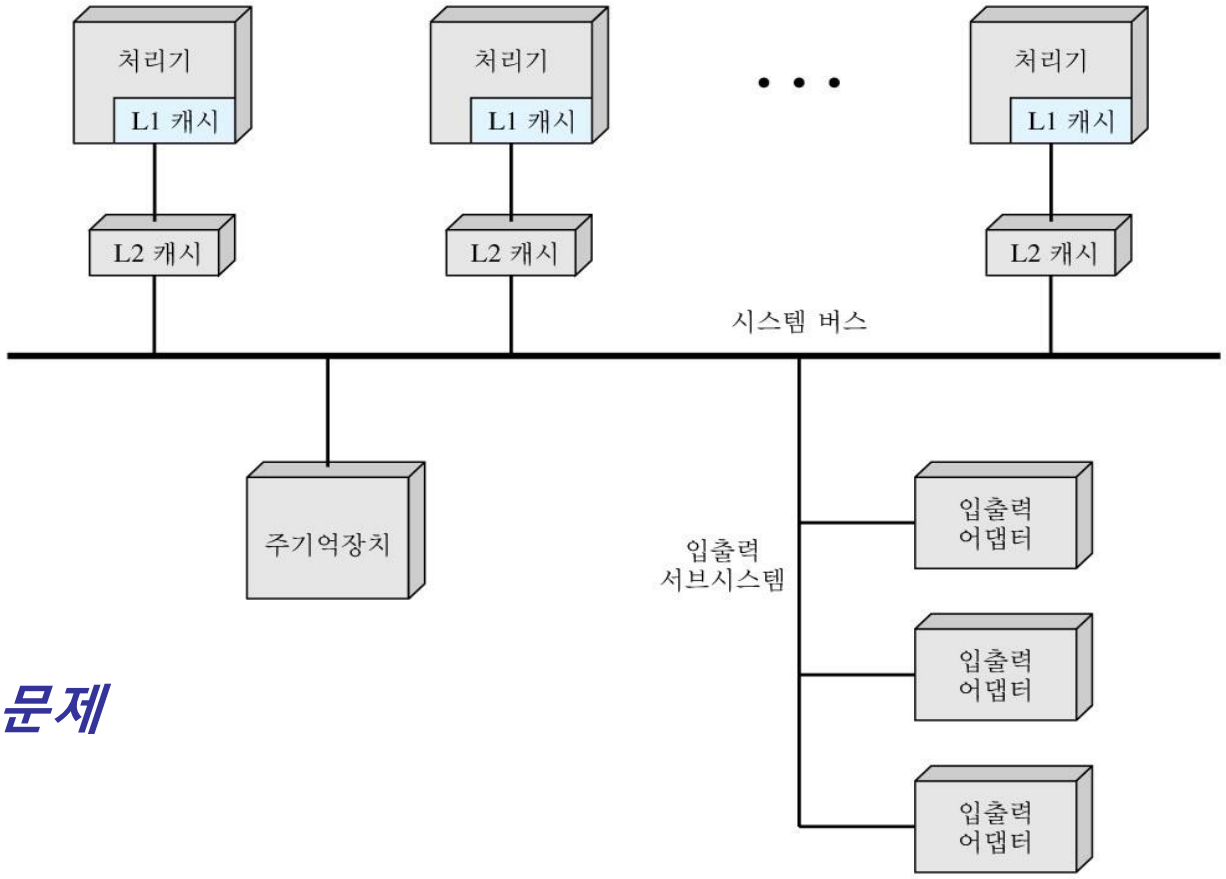
a	+	a	=	2a
b		b		2b
c		c		2c
d		d		2d

대칭적 다중처리 정의

- 두 개 이상의 유사한 수행 능력을 갖는 프로세스 들로 구성된다.
- 이들 프로세서는, 버스나 다른 내부 연결 방식에 의해 상호 연결된, 주기억장치와 I/O 장치 를 공유한다. 따라서 각 프로세서들이 메모리 접근하는 시간은 거의 동일하다.
- 동일한 장치에 이르는 경로들을 제공하는 채널이 동일하거나 달라도 모든 프로세서는 I/O 장치 접근을 공유한다.
- 모든 처리기는 동일한 기능을 수행할 수 있다(이런 의미로 대칭형이라는 용어 사용).
- 시스템은 프로세서들과 작업(job), 태스크(task), 파일, 그리고 데이터 요소 수준에서 프로 그램들 간의 상호작용을 제공하는 하나의 통합된 운영체제에 의해 제어된다.

대칭적 다중처리

- SMP (symmetric multiprocessor) 구성



☞ ***캐시 일관성 문제***

대칭적 다중처리 장점

- 성능:
  - ✓ 컴퓨터가 수행해야 할 작업의 일부가 병렬로 처리될 수 있다면, 여러 개의 처리기로 구성된 시스템은 동일한 타입의 단일 처리기로 구성된 시스템에 비해 많은 성능 향상이 가능하다.
- 가용성:
  - ✓ 대칭형 멀티프로세서 시스템에서 모든 처리기들이 동일한 기능을 수행할 수 있기 때문에, 하나의 처리기가 고장나더라도 시스템은 다소 성능이 떨어진 상태에서 지속적으로 동작할 수 있다.
- 점진적 확장:
  - ✓ 사용자는 성능 향상을 위해 필요할 때마다 처리기를 추가로 설치할 수 있다.
- 크기 조정(scaling):
  - ✓ 벤더들은 시스템을 구성하는 처리기의 수에 따라 가격과 성능이 다른 다양한 제품을 공급할 수 있다.

대칭적 다중처리

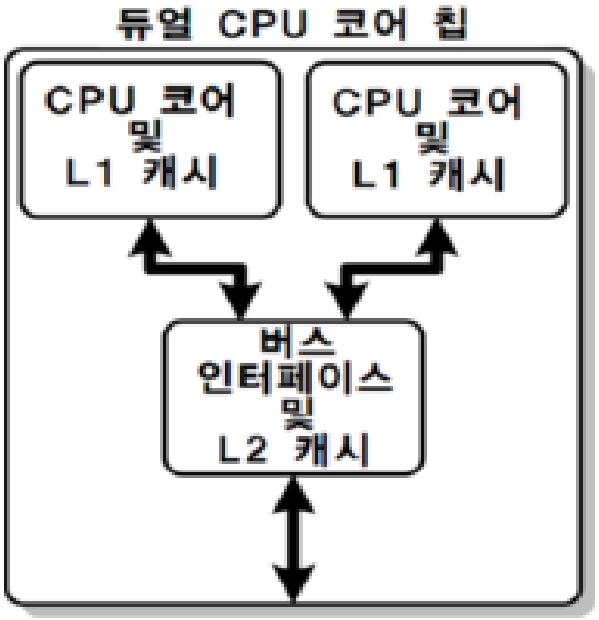
- SMP를 위한 OS 설계 시 고려 사항
  - ✓ 목적: 사용자가 다중프로그래밍 단일처리기 시스템 (multiprogramming uniprocessor system)과 동일한 관점을 가질 수 있게 자원을 관리
  - ✓ 동시적 병행(simultaneous concurrent) 프로세스 또는 쓰레드
    - 재진입(reentrant) 커널 코드, 분리된(separated) 커널 자료 구조
  - ✓ SMP를 위한 스케줄링
    - 전역 큐 / 지역 큐, 집단 스케줄링(gang scheduling), 친화성(affinity)
  - ✓ 동기화
    - 상호배제(mutual exclusion), 사건 순서화(event ordering)
  - ✓ 메모리 관리
    - 다중포트 메모리(multiport memory): dualport memory
  - ✓ 신뢰성 및 결함 허용 (fault tolerance)
    - 이주 (migration)

멀티코어 컴퓨터

- 칩 멀티프로세서로 알려져 있음
- 하나의 실리콘 다이(die)에 2개 이상의 코어라고 불리는 프로세서가 내장되어 있으며, 각 코어는 독립적인 프로세서의 모든 구성요소를 포함하고 있음
  - ✓ 각 core는 독립된 프로세서의 모든 컴포넌트 들, 즉 레지스터, ALU, pipeline HW, control unit 등으로 구성
  - ✓ L1 명령어와 데이터 캐시 포함
- 멀티코어 칩은 L2 캐시와 L3 캐시를 포함할 수도 있음

- 컴퓨터구성요소를 소형화 시켜 프로세서를 캐시에 가깝게 위치
- 캐시 선반입(prefetching) 기법 많이 사용
  - ✓ 메모리 액세스 패턴을 보고 곧 요구될 것 같은 데이터를 추측하여 캐시에 미리 가져다 놓음

멀티코어 컴퓨터



인텔코어**2 E6600-** 듀얼코어 프로세서

AMD Athlon X2 6400- 듀얼코어 프로세서

멀티코어 컴퓨터

- Core i7-990X chip 2가지 통신방식 :
  - ✓ DDR3 (Dual Data Rate-3) 메모리 컨트롤러로 32 Gbps
  - ✓ 퀵패스 인터커넥트(QPI) 를 통해 점대점 링크된 프로세서 칩간 고속 통신 지원

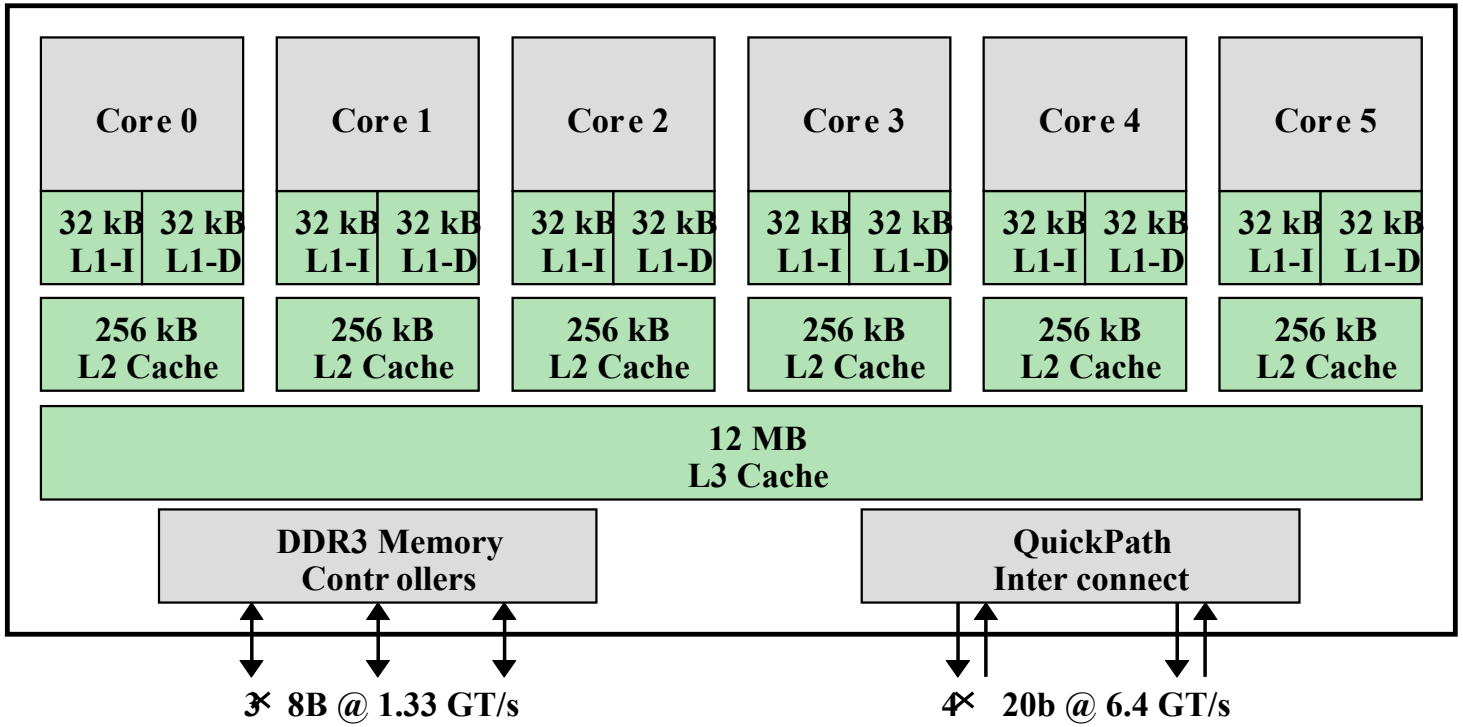


그림 1.20 Intel Core i7-990X Block Diagram

Summary

- H/W basic elements
- Processor Registers
- Instruction Execution
- Interrupt
- Memory Hierarchy: trade off among speed, capacity, and cost
- Cache Memory: design issue
- I/O Communication Techniques: Programmed I/O, Interrupt, DMA

1A Performance Characteristics of 2-level Memory

- Classes of 2-level memory

Table 1.2 Characteristics of Two-Level Memories

	Main Memory Cache	Virtual Memory (Paging)	Disk Cache
Typical access time ratios	5 : 1	10^6 : 1	10^6 : 1
Memory management system	Implemented by special hardware	Combination of hardware and system software	System software
Typical block size	4 to 128 bytes	64 to 4096 bytes	64 to 4096 bytes
Access of processor to second level	Direct access	Indirect access	Indirect access

1A Performance Characteristics of 2-level Memory

- Operation of Two-Level Memory
  - ✓ $T_s = H * T_1 + (1-H) * (T_1+T_2)$
    $= T_1 + (1-H) * T_2$
    - T_s: average (system) access time
    - T_1: access time of M_1 (cache, disk cache)
    - T_2: access time of M_2 (main memory, disk)
    - H: Hit ratio

- Performance Issues
  - ✓ hit ratio
  - ✓ speed
  - ✓ cost

1A Performance Characteristics of 2-level Memory

- Performance: Hit ratio

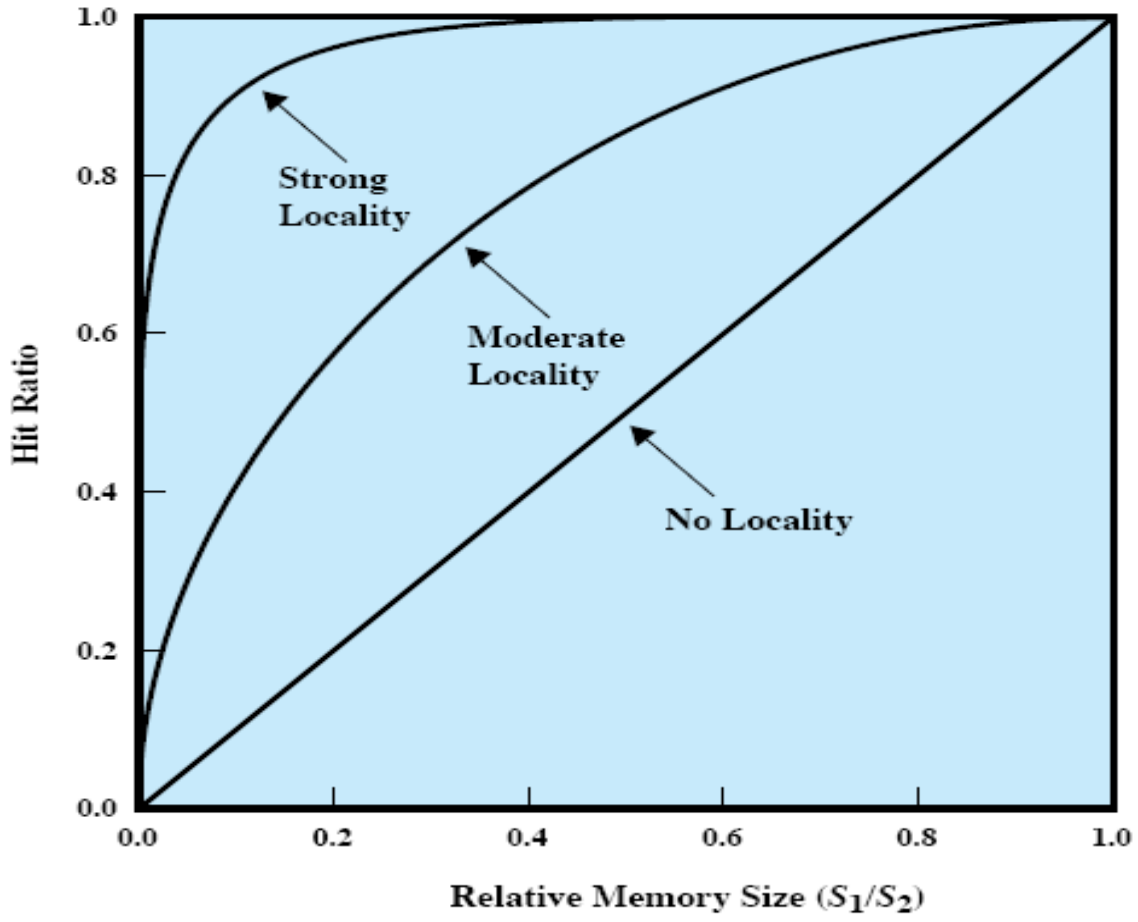


Figure 1.24 Hit Ratio as a Function of Relative Memory Size

1A Performance Characteristics of 2-level Memory

- Locality
  - ✓ program execution is sequential
  - ✓ localized to a few procedure
  - ✓ iteration
  - ✓ data structures such as arrays or sequences of records

- Temporal locality, Spatial Locality

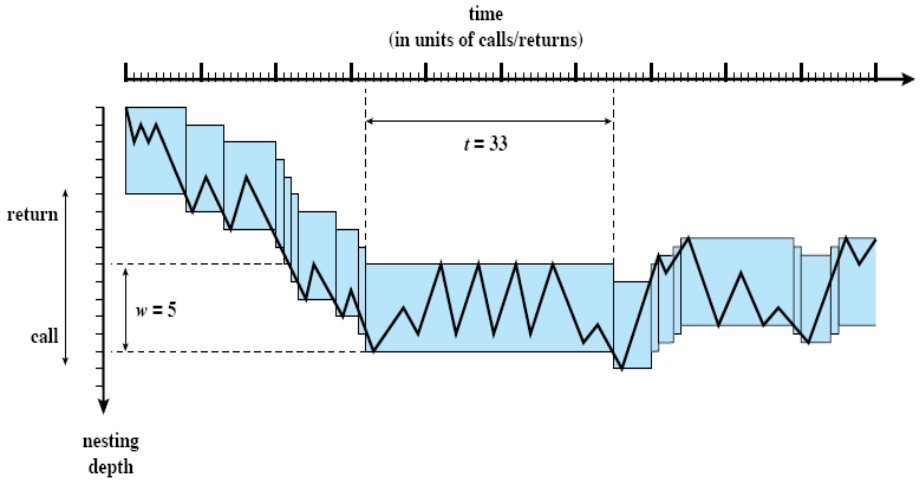


Figure 1.20 Example Call-Return Behavior of a Program

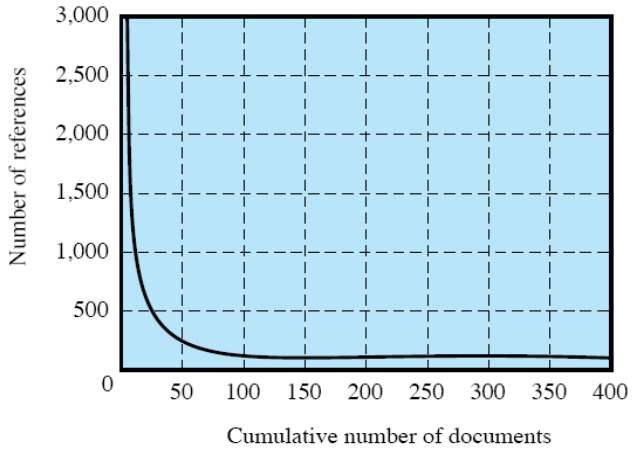


Figure 1.21 Locality of Reference for Web Pages [BAEN97]