1. Show the interconnection of three modules in Fig.3.15.

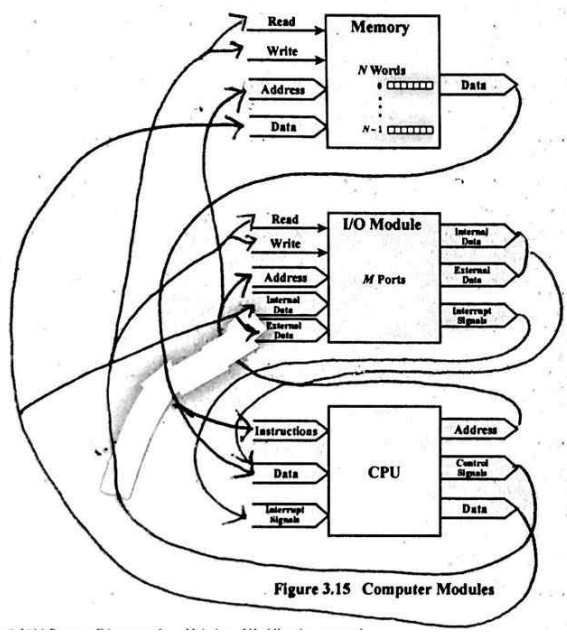

Figure 3.15 Computer Modules

컴퓨터는 세 개의 기본 요소들 혹은 모듈들(프로세서, 기억장치 및 I/O장치)로 구성되며, 이들은 서로 통신을 한다. 사실상 컴퓨터는 기본 모듈들의 네트워크(network)이므로, 이 모듈들을 서로 연결하기 위한 경로들(paths)이 필요하다. 여러 모듈들을 연결하는 경로들의 집합을 상호연결 조직(interconnection structure)이라고 한다. 이 조직은 모듈들 간에 어떠한 교환들이 이루어져야 하는 지에 따라 설계된다. 그림 3.15는 각 모듈의 유형 별로 주요 입력 및 출력 신호들을 보여줌으로써, 교환의 유형들을 제시하고 있다.

• 기억장치 : 전형적으로 기억장치는 동일한 길이를 가진 N개의 단어들로 구성된다. 각 단어에는 고유의 주소가 할당된다. 읽기/쓰기 제어 신호에 따라 한 단어의 데이터를 기억장치로부터 읽거나 쓸 수 있다. 그 동작이 수행되는 위치는 주소에 의해 지정된다.
• I/O 모듈 : 컴퓨터 시스템 내부의 관점에서 볼 때, I/O 모듈은 기능적으로 기억장치와 비슷하다. 즉, 읽기와 쓰기 동작이 있다. 그리고 한 개의 I/O 모듈은 한 개 이상의 외부 장치들을 제어할 수 있다. 외부 장치와 연결된 각 인터페이스를 포트(port)라고 하며, 각각에 고유의 주소가 부여된다. 이 외에도 I/O 모듈에는 외부 장치와 데이터를 주고받기 위한 별도의 데이터 경로들이 있다. 마지막으로, I/O 모듈은 프로세서로 인터럽트 신호를 보낼 수도 있다.
• 프로세서 : 프로세서는 명령어들과 데이터를 읽어 들이고, 처리한 데이터를 내보내며, 시스템의 전반적인 동작을 제어하기 위하여 제어 신호들을 사용한다. 또한 인터럽트 신호도 받는다.

지금까지는 서로 교환될 데이터를 정의하였다. 상호연결 조직은 다음과 같은 유형의 전송들을 반드시 지원해야 한다.

• Memory to processor : 프로세서는 기억장치로부터 명령어 혹은 데이터를 읽는다.
• Processor to Memory : 프로세서는 데이터를 기억장치에 저장한다.
• I/O to Processor : 프로세서는 I/O 모듈을 통하여 I/O 장치로부터 데이터를 읽는다.
• Processor to I/O : 프로세서는 I/O 장치로 데이터를 보낸다.
• I/O to or from Memory : 두 가지 경우 모두에서 I/O 모듈은, 프로세서를 통하지 않고, DMA를 이용하여 기억장치와 직접 데이터를 교환하는 것이 허용된다.

2. Explain the locality of memory reference used in the design of cache memory.

프로그램 실행이 진행되는 동안에 프로세서가 명령어와 데이터를 읽기 위해 기억장치를 참조하는 위치들은 밀집되는 경향을 참조의 지역성(locality of reference)라 한다. 참조의 지역성의 종류에는

• 공간(spatial) 지역성 : 특성 클러스터의 기억 장소들에 대해 참조가 집중적으로 이루어지는 경향
• 시간(temporal) 지역성 : 최근 사용되었던 기억 장소들이 집중적으로 액세스되는 경향
• 순차(sequential) 지역성 : 데이터가 순차적으로 액세스되는 경향

이 있다. 명령어는 보통 공간 지역성이 높고 데이터는 보통 시간 지역성이 높다. 이 둘을 나누어 서로 다른 지역성을 이용할 수 있다. 또한 명령어와 데이터를 동시에 읽어올 수 있게 함으로써 CPU의 파이

프라이닝 성능을 향상시킬 수 있다. CPU가 데이터를 요청했을 때 캐시 메모리가 해당 데이터를 가지고 있다면 이를 캐시 히트라 부르고, 해당 데이터가 없어서 DRAM에서 가져와야 한다면 캐시 미스라 부른다. 캐시 미스 발생 시의 처리 방법은 캐시 정책에 따라 다르며, 데이터를 읽어 오는 시점으로 사용하기도 한다.

3. Explain why a memory hierarchy is employed in contemporary computer systems.

레지스터와 캐시는 CPU 내부에 존재한다. 당연히 CPU는 아주 빠르게 접근할 수 있다. 메모리는 CPU 외부에 존재한다. 레지스터와 캐시보다 더 느리게 접근 할 수 밖에 없다. 하드 디스크는 CPU가 직접 접근할 방법조차 없다. CPU가 하드 디스크에 접근하기 위해서는 하드 디스크의 데이터를 메모리로 이동시키고, 메모리에서 접근해야 한다. 아주 느린 접근 밖에 불가능하다. 따라서 레지스터와 캐시 같이 빠르고 메모리와 하드 같이 용량이 큰 메모리를 사용하고 싶겠지만 현실적으로 불가능하다. 따라서 용량이 작지만 빠르고 느리지만 용량이 큰 메모리를 함께 사용함으로써 우리는 1) 비트당 가격 감소 2) 용량 증가 3) 엑세스 속도 감소 4) 프로세서에 의해 기억장치가 엑세스되는 빈도 감소 등의 효과를 얻을 수 있다. 또 4번의 경우에는 자주 쓰는 정보를 캐시에 저장해 놓음으로써 메모리의 엑세스 빈도를 감소시키는 효과를 얻는다.

4. If the access time of main memory = T_m, the cache access time = T_c, the cache hit ratio is H, what is the effective memory access time? Explain the ways to improve the speedup by the cache.

$$T_a = T_c + (1 - H)T_m$$

(T_a : 평균엑세스시간, T_c : 캐시엑세스시간, T_m : 기억장치엑세스시간, H : 적중률)

캐시의 속도에 가장 큰 요인은 H : 적중률 즉, 히트율이다. 결국 캐시의 속도를 향상시키는 방법을 설명하고자하면 당연히 히트율에 대한 설명이 필요하다. 히트란 CPU가 원하는 정보가 캐시에 있을 경우를 말한다. 즉 히트율을 높인다는 것은 CPU가 원하는 정보를 어떻게 캐시에 있게 만드는 지를 의미한다. 따라서 CPU가 원하는 정보가 캐시에 들어있게 하기 위해서는 캐시의 크기가 크면 클수록 많은 정보가 들어갈 수 있기 때문에 캐시의 크기를 키워주는 것은 히트율을 높이는 방법 중 한가지라고 할 수 있다. 또 히트율에 관련이 있는 사항에는 mapping이 있다. 히트율이 가장 높은 mapping은 associate mapping이기 때문에 associate mapping 방식을 활용하여 히트율을 높이는 것도 캐시의 속도를 높이는 방법 중 하나가 될 수 있다.

5. Consider a byte addressable main memory of 8Gbytes, block size of 8bytes, and a cache consisting of 1M lines.

1) Specify the number of bits in the memory address.

byte addressable : 각 주소에는 1byte 존재 (byte 단위로 저장)
8Gbytes => 0 ~ 2^{33}번지 => 2^{33}이므로 메모리 주소의 비트수는 33bits이다.

∴ memory address 수 = 33bits

2) How many bits in memory address are used for TAG in direct, associative and 2-way set associative mapping.

block size of 8bytes = 2^3 〉〉 word = 3bits

cache 1M lines = 2^{20} 〉〉 line = 20bits

set 1개당 들어있는 line 수 = $2^{20}/2 = 2^{19}$ 〉〉 set 19bits

TAG에 사용되는 메모리 주소의 비트 수	
Direct	33 - 20(Line) - 3(Word) = 10bits
Associate	33 - 3(Word) = 30bits
Set-associate	33 - 19(Set) - 3(Word) = 11bits

3) Compare the mapping process of direct and associative mapping.

direct mapping은 주기억장치의 각 블록을 한 개의 캐시 라인으로만 mapping하는데 비해 associate mapping은 주기억장치 블록이 캐시의 어떤 라인으로도 적재될 수 있다. direct mapping에서는 캐시 제어 논리가 주기억장치의 주소를 tag, line, word로 해석하는 반면 associate mapping은 태그가 일치하는지 모든 라인들과 동시에 비교하기 때문에 라인 위치에 대한 정보가 필요하지 않으므로 주기억장치의 주소는 tag, word로 구성되어 있다. direct mapping은 간단하고 구현 비용이 적게 들지만 어떤 블록이든 들어갈 수 있는 캐시 위치가 고정되어 있다. 따라서, 프로그램이 같은 라인에 맵핑되는 두 개의 블록들로부터 단어들을 반복해서 읽어와야 한다면, 그 블록들은 캐시에서 반복적으로 교체될 것이고, 결과적으로 히트율이 낮아지게 된다(이러한 현상을 스레싱(thrashing)이라고 한다). associate mapping은 캐시 라인들의 태그들을 병렬로 검사하기 위한 복잡한 회로가 필요하다. 하지만 direct mapping이 들어가는 라인이 정해진 블록들을 교체하기 위한 과정에서의 효율이 떨어지는 반면에 associate mapping에서는 LRU(least recently used)와 같은 교체 알고리즘을 사용하기 때문에 블록 교체의 효율이 높다.

6. Explain the difference between write through policy and write back policy.

write through 기법은 모든 쓰기 동작들이 캐시로 뿐 아니라 주기억장치로도 동시에 행해짐으로써, 주기억장치의 내용들은 항상 유효하도록 보장한다. 이 방법의 주요 단점은 기억장치 사용량이 많아져서 병목 현상을 야기할 수 있다는 것이다. write back 기법은 기억장치에 대한 쓰기 동작을 최소화시켜준다. write back에서는 데이터의 갱신이 캐시에서만 일어나고 갱신 동작이 일어날 때, 그 라인의 더티 비트(dirty bit) 혹은 사용 비트(use bit)가 세트된다. 나중에 그 블록이 교체될 때, 더티 비트가 세트된 경우에만 주기억장치에 갱신된 후에 교체된다. write back의 문제점은 주기억장치의 일부분이 무효(invaild) 상태에 있다는 점이다. 따라서 I/O 모듈에 의한 액세스는 반드시 캐시를 통하여 이루어져야 한다. 결과적으로 회로가 복잡해지며, 병목이 발생할 가능성이 있다.

7. Explain advantages of a split cache compared to a unified cache. Which of L1, L2 and L3 cache is proper for split cache?

병렬 명령어 실행 및 예측된 미래 명령어들의 선인출을 강조하는 슈퍼스칼라 기계들을 위하여, L1은 분리 캐시로 더 높은 레벨에서는 통합 캐시로 구성되고 있다. 분리 캐시 설계의 주요 이점은 명령어 인출/해독 유니트와 실행 유니트 간에 캐시로 인한 경합을 없앨 수 있다는 점이다. 이것은 명령어 파이프라인에 의존하는 설계들에서는 중요하다. 통합 캐시의 경우 실행 유니트가 데이터 적재와 저장을 위해 기억장치를 액세스할 때는 그 요구가 통합 캐시로 보내지고 만약 그와 동시에 명령어 선인출기(instruction prefetcher)가 명령어 인출을 위해 캐시로 읽기 요구를 보냈다면, 요구는 일시적으로 중단되고, 캐시는 실행 유니트를 먼저 서비스하여 현재의 명령어 실행을 완료할 수 있게 해준다. 이러한 캐시 경합은 명령어 파이프라인의 효율적인 사용을 방해함으로써 성능을 저하시킨다. 분리 캐시는 이러한 어려움을 해결해준다. 〉〉 분리캐시로는 L1 캐시가 적합하다.

8. For a system with 2 levels of cache, T_{c1} = 1st level cache access time, T_{c2} = 2nd level cache access time, T_m = main memory access time, H_1 = 1st level cache hit ratio, H_2 = 2nd level cache hit ratio. Provide an equation for effective memory access time T_a for a read operation.

$$T_a = (T_{c1} + (1 - H_1)T_{c2}) + (1 - H_2)T_m$$

〉〉〉 일반적인 읽기 동작에 대한 단일-단계 캐시의 성능에 대한 식은 $T_a = T_c + (1-H)T_m$이다. 여기서 T_c는 캐시 액세스 시간, T_m은 기억장치 액세스 시간, H는 히트율이다. 그러므로 원래 식의 T_c에는 $(T_{c1} + (1 - H_1)T_{c2})$식을 넣어줌으로써 T_{c2}를 일반식의 기억장치 액세스 시간이라 생각하고 원래 식에 대신 넣어준다. 또 일반식의 $(1-H)T_m$부분에는 첫 번째/두 번째 단계 캐시들의 통합 적중률인 H_2를 H 대신 넣어준다.

9. If 4 clock cycles (address, read, wait, data in each clock) are required for memory read operations in a synchronous bus (clock frequency : 2GHz) with 16 data lines (16bits), calculate the maximum memory transfer rate (in bps).

memory transfer rate(bps) = clock frequency(Hz) / clock cycles * data lines(bit)
= 2G * 16 / 4
= 8G bps

10. The memory(8GB) of a computer is built from 4G × 1 DRAM chips.

1) How many DRAM chips are needed?

4G * 1 DRAM chips = $4 * 2^{30} * 1 = 2^{32} bits$
memory(8GB) = $8(8) * 2^{30}(G) * 2^3(B) = 2^{36} bits$
$2^{36} / 2^{32} = 2^4 = 16$
∴ 16개의 DRAM chip이 필요하다.

2) Each row of a DRAM chip must be refreshed at least once every 1 m/sec. What is the time period between successive refresh requests?

4G * 1 DRAM chip : 2^{32}
=〉 DRAM address : 32 / 2 = 16
=〉 매 1회 이상 refresh되어야함 –〉 1m/sec * 16 = 16m/sec

11. For a disk with average seek time = 4ms, 6000rpm, 512bytes/sector, 500sectors/track;
We wish to read a file consisting of 250 sectors. Estimate the access time and data transfer time for reading the file. Assume the file is stored as compactly as possible on the disk.
11. 평균 탐색 시간이 4ms, 6000rpm, 512bytes / sector, 500sectors / track 인 디스크의 경우;
250 개의 섹터로 구성된 파일을 읽으려고 합니다. 파일을 읽기위한 액세스 시간 및 데이터 전송 시간을 추정하십시오. 파일이 디스크에 가능한 한 컴팩트하게 저장되었다고 가정하십시오.

- rpm : rotation per minute 〉〉 6000rpm : 1분에 6000번 회전 〉〉 1초에 100번 회전
  〉〉 1번 회전하는데 10ms(rotation time = 10ms) 〉〉 rotational latency = 5ms
- track 1개 당 500개의 sector 〉〉 transfer time = 10msec / 500 = 0.2msec
  〉〉 250개의 sector을 읽는 데 걸리는 시간 = 0.2msec * 250 = 5msec
  〉〉 총 걸리는 시간 = 4ms + 5ms + 5ms = 14ms

12. How does an external device indicate an interrupt request to the processor?

외부 장치가 다시 서비스를 받을 준비가 되었을 때, 즉 프로세서로부터 다시 데이터를 수신할 준비가 되었을 때 외부 장치는 인터럽트 요구(interrupt request) 신호를 프로세서로 보낸다. 그러면 프로세서는 현재 수행 중인 동작을 일시 중단하고, 특정 I/O장치를 서비스하기 위해 인터럽트 처리기(interrupt handler)로 이동하여, 장치에 대한 서비스를 마친 다음에 원래 수행하던 동작을 다시 시작한다.