#cs #study

0) 목차

- CPU
- 메모리
- 보조기억장치와 입출력장치

1) CPU

레지스터

- CPU 레지스터는 역할이 하나 밖에 없는가?

```
정답은 No!

레지스터는 여러 역할을 가지고 있고 역할마다 이름도 다릅니다.
단지, CPU 내부 임시 저장장치라는 공통 분모만 가지고 있을 뿐입니다.
```

- 프로그램 카운터(PC, programm counter)
  - 메모리에서 다음으로 읽을 명령어의 주소 저장
  - 별칭으로 메모리 포인터라 부르기도 하며 카운터는 기본적으로 메모리 주소를 저장할 때마다 1씩 증가한다
- 명령어 레지스터(IR, instruction register)
  - 메모리에서 읽어 들인 명령어를 저장한다.
  - CPU 내의 제어장치는 명령어 레지스터의 명령어를 해석한다.
- 범용 레지스터(general purpose register)
  - 데이터, 명령어, 주소 모두 저장
  - 일반적인 상황에서 능동적으로 쓰기 위한 레지스터
- 플래그 레지스터(flag register)
  - 연산 결과 또는 CPU 상태에 대한 정보를 저장
  - 종류
    - 부호 플래그 : 연산 결과 부호(0 : 양수, 1 : 음수)
    - 제로 플래그 : 연산 결과가 0인가?(0 : false, 1 : true)
    - 캐리 플래그 : 연산 결과에 올림수 또는 빌림수가 발생했나?(0: false, 1 : true)
    - 오버플로우 플래그 : 오버플로우가 발생했나?(0: false, 1 : true)
    - 인터럽트 플래그 : 인터럽트가 가능한가?(0: false, 1: true)
    - 슈퍼바이저 플래그 : 커널 모드로 실행 중인가?(0: 사용자 모드, 1 : 커널 모드)
- 스택 포인터

- 메모리에서 스택 영역의 가장 첫번째 위치 주소를 참조하는 레지스터
- 스택 영역 : 메모리에서 암묵적으로 스택처럼 사용하는 영역

인터럽트

- 인터럽트는 CPU 작업을 방해할 수 없다

> 정답은 No!
>
> 인터럽트는 CPU의 작업을 방해하는 신호입니다.
> 현재 수행하는 작업과 별개로 지금 수행할 작업이 있을 때, 이 신호를 보내서 CPU가 이 일을 처리하도록 요청합니다.

- 동기 인터럽트
  - CPU로부터 발생하는 신호
  - 예외 상황에서 발생
  - 종류 : 폴트, 트랩, 중단, 소프트웨어 인터럽트
  - 폴트
    - 예외를 처리하고 예외가 발생한 명령어부터 실행한다
    - 예시) 명령어 실행을 위한 데이터가 보조기억장치에 있는 경우
  - 트랩
    - 예외를 처리하고 예외가 발생한 명령어의 다음 명령어부터 실행한다
    - 예시) 디버깅의 브레이크포인트
  - 중단
    - 프로그램을 강제로 중단시킬 수 밖에 없는 심각한 오류가 있는 경우
  - 소프트웨어 인터럽트
    - 시스템 콜이 발생했을 때
- 비동기 인터럽트
  - CPU 외, 주로 입출력장치에서 발생하는 신호
  - 하드웨어의 작업이 언제 끝날 지 알 수 없으므로 비동기 적으로 결과를 주고 받아 CPU의 효율성을 높인다
  - 참고사항
    - 풀링 : 인터럽트와 달리 입출력 작업에서 지속적으로 CPU가 작업 경과를 확인하는 것
  - 처리순서
    - 입출력 장치가 CPU에 인터럽트 요청
    - CPU는 실행 사이클 종료 후에 인터럽트 여부 확인
    - 인터럽트 여부 확인 시, 요청이 있으면 인터럽트 플래그 통해 수용 여부 확인
    - 수용 가능 시, 현재 작업을 백업
    - 인터럽트 벡터를 참조하여 인터럽트 서비스 루틴 실행
    - 인터럽트 서비스 루틴 종료 후, 백업한 작업 복구

    - 인터럽트 요청 신호
        - CPU에서 인터럽트 가능 여부 판단
    - 인터럽트 플래그
        - 플래그 레지스터에 있는 인터럽트 플래그. 이 플래그가 false면 인터럽트 일부를 막을 수 있다. 막을 수 없는 인터럽트도 존재함
    - 인터럽트 서비스 루틴
        - 어떤 인터럽트가 발생했을 때, 인터럽트를 어떻게 처리하고 작동할 지에 대한 정보가 담겨 있다
    - 인터럽트 벡터
        - 인터럽트 서비스 루틴 식별 정보. 인터럽트 서비스 루틴 시작 주소를 포함한다.
    - 백업한 내용은 어디에 저장해요? 메모리 내 스택 영역에 저장합니다

CPU 성능과 연관된 설계

- CPU 클럭 속도
    - 클럭(clock) : 컴퓨터의 부품을 일시불란하게 움직일 수 있게 하는 시간 단위. 단위는 Hz.
    - 이진수로 이루어진 컴퓨터의 사고방식에서 시간 또한 ON/OFF로 나눌 수 있을 것이다. ON일 때는 작업을 확인하고 OFF일 때는 확인하지 않는다. 이 주기에 맞춰서 CPU가 연산을 수행하는 것
    - 당연하게도 ON/OFF 주기가 빠르면 빠를 수록 단시간에 더욱 많은 일처리를 할 수 있겠지만 대신에 그만큼 부품이 혹사하므로 열 에너지가 빠르게 증가한다. 즉, 발열이 심해진다.
- 멀티 코어와 멀티 스레드
    - 코어(core) : CPU에서 명령어를 읽고, 해석하고, 실행하는 부품. 하나의 코어는 하나의 업무만 수행하기에 코어가 많을 수록 여러 업무를 동시에 실행할 수 있다.
    - 스레드(thread) : 하나의 코어가 동시에 처리할 수 있는 명령어의 단위
        - CPU의 스레드를 하드웨어 스레드 또는 논리 프로세서라고도 부른다
        - 정확히는 동시에 처리하기 보다는 여러 업무를 분할하여 처리함으로서 동시에 처리하는 것처럼 보이는 것
        - 1코어 1스레드가 A업무만 처리한다면 1코어 2스레드는 A와 B업무를 A(1/4) -> B(1/4) -> A(2/4) -> B(2/4) -> ... 이런 식으로 스레드 만큼의 업무를 가져가서 분할하여 처리한다
        - 참고사항
            - 소프트웨어 스레드 : 하나의 프로그램에서 독립적으로 실행되는 단위. 프로그램 내부에서 업무를 분할하여 실행시키는 것이고 CPU가 보기에는 그냥 단일 프로그램이다.
- 병렬성과 동시성
    - 병렬성(parallelism) : 물리적으로 작업을 동시에 처리하는 것. 멀티 코어
    - 동시성(concurrency) : 빠르게 작업을 교대로 진행하여 동시에 작업을 처리하는 것처럼 보이는 것. 멀티 스레드

파이프라이닝

- 명령어 병렬 처리 기법 중 하나

- 명령어의 실행 과정을 인출(fetch), 해석(decode), 실행(execute), 저장(write back)으로 구분하고 각 과정에 다른 명령어도 같이 끼워 넣어서 처리하는 것
  - 슈퍼 스칼라 : CPU 내부에 여러 명령어 파이프라인을 포함하는 구조. 명령어 실행 과정을 하나의 파이프라인으로 보았을 때, 동일 클럭 주기 내에서 여러 명령어를 실행시키려면 복수의 파이프라인이 존재해야 한다.
- 명령어 집합에 따른 유의미한 성능 차이 존재
  - CISC : 다채로운 기능을 지원하는 복잡한 명령어로 구성된 집합. 실행 시간이 일정하지 않고 여러 클럭 주기가 필요함
  - RISC : 짧고 규격화된 명령어. 1클럭 내외로 실행되는 명령어 지향.
- 파이프라인 위험(pipeline hazard) : 파이프라이닝이 CPU 성능 향상에 실패하는 경우
  - 데이터 위험(data hazard) : 데이터 의존성이 있는 경우, 같은 주기 내에 있어도 의존하는 데이터가 아직 처리되지 않았으므로 문제가 생김
  - 제어 위험(control hazard) : 프로그램 카운터의 갑작스러운 변화. JUMP, CONDITIONAL JUMP, 인터럽트 등으로 프로그램 실행 흐름이 바뀌면 사전에 인출 또는 해석 중인 명령어 쓸모가 없어짐.
  - 구조적 위험(structural hazard) : 명령어를 겹쳐 실행하는 과정에서 서로 다른 명령어가 동시에 ALU, 레지스터 등 같은 CPU 부품 사용 시.

Quiz

- 레지스터 중 하나인 프로그램 카운터가 if문의 참을 만나거나, while 문을 탈출하거나 return 문을 만나서 실행 흐름이 순차적에서 벗어날 경우, 어떻게 될까?
- A 프로그램을 실행하다가 마우스에서 인터럽트 요청이 들어왔다. 이 때, 마우스를 뒤로 키보드와 모니터도 인터럽트 요청이 들어오면 처리 순서가 어떻게 되는가? 인터럽트 플래그는 true라고 가정한다.

2) 메모리

RAM

- RAM은 메모리 접근하는 데 O(N)이다.

```
정답은 No!

RAM은 임의 접근(Random Access) 방식이며 임의의 위치에 곧장 접근이 가능합니다. 그래서 어느 위치에 있든 한 번(O(1))에 접근할 수 있습니다.
```

- DRAM(dynamic ram)
  - 시간이 지나면 저장된 데이터가 점차 사라지는 RAM
  - 일정 주기로 데이터를 재활성화 해야 데이터 소멸을 막는다
  - 컴퓨터 메모리로 많이 사용하며 저전력, 저렴, 집적도가 높은 장점이 있음
- SRAM(static ram)

    - 시간이 지나도 저장된 데이터가 변하지 않는다. 단, RAM이기에 전원이 공급되지 않으면 저장된 내용은 없어진다.(휘발성은 여전하다)
    - 고전력, 고비용, 집적도가 낮지만 속도가 빠르기에 캐시 메모리에 사용
- SDRAM(synchronous DRAM)
    - 클럭 신호와 동기화된 DRAM
    - 클럭 신호에 맞춰 CPU와 정보를 주고 받을 수 있음
- DDR SDRAM(double data rate SDRAM)
    - 대역폭을 넓혀 속도를 빠르게 만든 SDRAM
        - 대역폭 : 데이터를 주고받을 통로의 크기
    - DDR SDRAM보다 2^n으로 커질 때는 DDR 뒤에 n에 해당하는 숫자를 붙여서 몇 배인지 표시한다. DDR4 SDRAM은 기존 DDR SDRAM의 16배

엔디안

- 엔디안이란 메모리에 바이트를 밀어 넣는 순서를 말한다
- 빅 엔디안(big endian)
    - 낮은 번지의 주소에 상위 바이트부터 저장하는 방식
    - 레지스터에 16진수로 1A2B3C4D라고 저장되어 있는 데이터를 a번지에 저장하면 1A는 a, 2B는 a + 1, 3C는 a + 2, 4D는 a + 3에 저장
    - 사람이 숫자 체계를 읽고 쓰는 순서와 동일. 그래서 메모리 값을 사람이 직접 읽거나 디버깅 할 때 편하다
- 리틀 엔디안(little endian)
    - 낮은 번지의 주소에 하위 바이트부터 저장
    - 빅 엔디안의 예시에서 반대로 저장하면 된다
    - 작은 자릿수부터 적기 때문에 읽고 쓰는 건 불편하지만 계산은 편하다. 계산은 작은 자릿수부터 시작하기 때문이다
- MSB와 LSB
    - MSB(Most Significant Bit) : 숫자의 크기에 가장 큰 영향을 미치는 유효 숫자. 빅 엔디안이 MSB가 있는 바이트부터 저장해 나가는 방식
    - LSB(Least Siginificant Bit) : 숫자의 크기에 가장 적은 영향을 미치는 유효 숫자. 리틀 엔디언이 LSB가 있는 바이트부터 저장해 나가는 방식

캐시 메모리

- CPU의 연산 속도와 메모리 접근 속도 차이를 줄이기 위해 사용하는 저장장치. CPU가 사용할 데이터를 미리 캐시 메모리로 옮기는 방식.
- CPU 코어의 거리에 따라 캐시 메모리를 구분
    - L1 캐시 : 코어 내부. 캐시 메모리 크기가 가장 작고 속도가 가장 빠르다
    - L2 캐시 : 코어 내부. 중간
    - L3 캐시 : 코어 외부. 캐시 메모리 크기가 가장 크고 속도가 가장 느리다
- 캐시 히트와 캐시 미스

    - 캐시 히트(cache heat) : 캐시 메모리가 자주 사용할 것으로 예측하여 저장한 데이터가 실제로 사용된 경우
    - 캐시 미스(cache miss) : 예측해서 가져왔지만 정작 다른 데이터를 사용하는 경우
    - 캐시 적중률(cache hit rate) : cache heat num / (cache heat num + cache miss num)
- 참조 지역성의 원리에 따른 캐시 적중률 높이기
    - 시간 지역성 : CPU는 최근에 접근했던 메모리 공간에 다시 접근하려는 경향이 있음
        - 예시) 변수
    - 공간 지역성 : CPU는 접근한 메모리 공간의 근처에 다시 접근하려는 경향이 있음
        - 예시) 배열
- 캐시 메모리의 일관성
    - CPU가 메모리에 저장된 변수값을 바꿀 때, 어떻게 바꿀 것인가? 캐시 메모리는 어떻게 하고?
    - 일관성 준수 방식
        - 즉시 쓰기(write-through) : 캐시 메모리와 메모리 영역의 값을 동시에 바꾸는 방법. 일관성을 준수할 수 있으나 데이터를 쓸 때마다 메모리를 참조하는 단점
        - 지연 쓰기(write-back) : 캐시 메모리 값을 바꾸고 나중에 메모리 값 바꾸기. 속도는 빠르지만 일관성이 깨질 위험성.

Quiz

- SDRAM은 클럭 신호와 동기화한 메모리다. 그렇다면 클럭 신호와 동기화하면 무슨 장점이 있는가?
- 컴퓨터 입장에서는 어떤 엔디안이 편할까? 이유는?
- CPU 코어 내부에 L1, L2 캐시가 있다. 그렇다면 두 개의 코어가 같은 L3 캐시를 쓰는 경우, 어떤 문제가 발생할 수 있는가?

3) 보조기억장치와 입출력장치

RAID

- 여러 개의 독립적인 보조기억장치를 하나의 보조기억장치처럼 사용하는 기술
- RAID 기술 방식은 여러개가 있으며 보통 RAID에 숫자를 붙여서 구분한다
- RAID0
    - 데이터를 여러 보조기억장치에 나누어 저장(스트라이핑 방식)
    - 데이터가 분할되었기에 각 보조기억장치에서 단숨에 데이터를 읽을 수 있어 입출력이 빠름. 대신에 보조기억장치가 하나라도 고장나면 데이터가 불완전해짐
- RAID1
    - 데이터가 분할되어 저장한 방식 그대로 복사본을 만들어 다른 보조기억장치에 동일한 방식으로 저장(미러링 방식)
    - 데이터 손실이 발생하면 복사본을 사용하면 되기에 복구가 간단하여 안정성이 높음. 하지만 데이터를 쓸 때 원본과 복사본 둘 다 적용할 필요가 있고 별도의 복사본이 있어서 용량을 잡아먹는다.
- RAID4
    - 패리티 정보를 저장하는 디스크를 별도로 두는 방식

        - 패리티(parity) : 오류 검출용 정보
    - RAID1에 비해 적은 하드디스크로도 안전하게 데이터 보관 가능. 하지만 각 보조기억장치에서 데이터가 변동사항이 생기면 그에 따른 각 데이터의 패리티 정보를 패리티 담당 보조기억장치에 몰아서 저장하기에 병목현상이 생길 수 있음
- RAID5
    - 패리티를 각 보조기억장치에 분산하여 저장하는 방식
    - RAID4의 병목 현상 극복 가능
- RAID6
    - 서로 다른 2개의 패리티를 두는 방식
    - RAID5처럼 패리티를 분산 저장하면서 각 데이터마다 서로 다른 패리티를 두 개 두는 구성으로 패리티 안정성을 높였다. 하지만 그만큼 패리티 정보를 더 많이 저장하므로 쓰기 속도는 RAID5보다 느리다.

입출력 기법

- 장치 컨트롤러와 장치 드라이버
    - 장치 컨트롤러 : CPU와 입출력장치 사이의 통신을 중개
    - 장치 드라이버 : 장치 컨트롤러의 동작을 알고, 장치 컨트롤러가 컴퓨터 내부와 정보를 주고받을 수 있도록 하는 프로그램
- 프로그램 기반 입출력
    - 프로그램 속 명령어로 입출력을 수행하는 방법을 의미한다
- 인터럽트 기반 입출력 : 다중 인터럽트
    - 여러 입출력 장치를 동시에 사용해서 인터럽트가 동시에 들어오는 경우, 우선순위가 더 높은 인터럽트부터 먼저 처리한다
    - 단, 인터럽트 플래그가 비활성화된 경우에도 무시할 수 없는 인터럽트(NMI)가 발생하면 그게 최우선 순위임
    - 프로그래머블 인터럽트 컨트롤러(PIC) 하드웨어를 사용하여 장치 컨트롤러의 인터럽트 요청 우선순위 판별 및 처리를 수행함
- DMA 입출력
    - 프로그램 기반 입출력과 인터럽트 기반 입출력 둘 다 CPU가 입출력 장치와 메모리 간 데이터 이동을 주관하며 이동하는 데이터들은 반드시 CPU를 거친다
    - CPU의 부담을 줄이기 위해 CPU를 거치지 않고 입출력 장치와 메모리가 상호작용 할 수 있게 DMA 컨트롤러가 나옴
    - 입출력 과정
        - CPU가 DMA 컨트롤러에게 입출력장치 주소, 수행할 연산, 연산할 메모리 주소 등의 정보와 함께 입출력 작업을 명령
        - DMA 컨트롤러가 CPU 대신에 장치 컨트롤러와 상호작용하며 입출력 작업을 수행
        - DMA 컨트롤러는 입출력 작업이 끝나면 CPU에게 인터럽트를 걸어 작업 종료를 알림