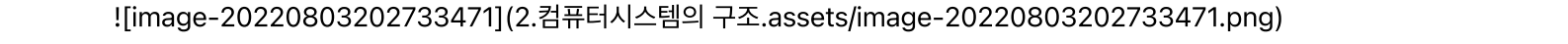
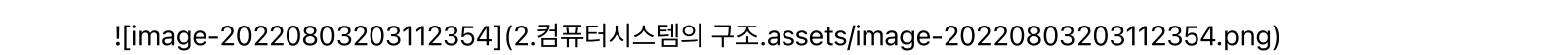
Mode bit, Timer



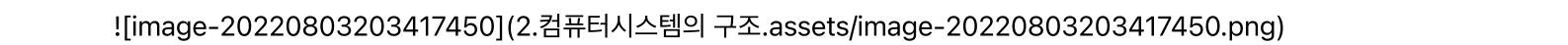
- I/O를 관리하는 작은 cpu는 ''컨트롤러''라고 부른다
- 달려있는 작은 메모리는 local buffer라고 부름
- 메모리도 디바이스니까 메모리컨트롤러가 있따
- cpu에서 기계어를 실행할때 지금 운영체제가 실행하는건지

  사용자 프로그램이 실행하는건지 구분하는게 'mode bit'이다



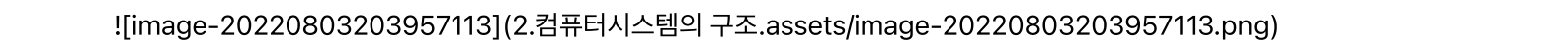
- 운영체제가 cpu로 실행중일때는 0으로 되어있음 (cpu가 아무거나 해도되는상태)
- 운영체제가 cpu를 사용자프로그램으로 넘겨줄때 Mode bit을 1로 바꾸어서 넘겨줌
  - 그러면 아무짓이나 할 수 있는게 아니고 모드빗이 1이니까 제한된 기계어만 가능하고

    위험한 기계어는 실행 x
  - 위험한 명령어 '특권명령'으로 규정 << 운영체제만 실행가능
- 그렇게 위험한 명령어를 실행하려면 자동으로 cpu가 운영체제에게 넘어감
  - 넘어가는 방법은 interrupt나 Exception(예외상황) 으로..

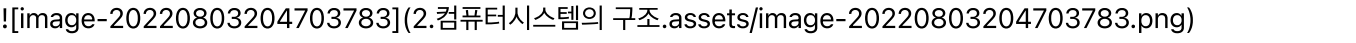


- cpu에는 Interrupt line이 들어가있는데 매 순간 메모리에서 기계어를 하나씩 실행한 다음에

  다음 기계어를 실행하기 전에 Interrupt line에 무슨 시그널이 온게 없는지 확인함(I/O들이 실행)
- I/O가 일을 다했다! 그러면 그걸 cpu에게 알려줘야하니까 알려주는 방법이 CPU한테 Interrupt를 걸음 CPU는 그다음 기계어를 실행하기 앞서서 확인하고 자동적으로 운영체제에게 넘어감
- Mode bit은 0으로 바뀌고 이제 다시 일을하겠지



- CPU는 registers 들도 있다. 얘들은 여러 아웃풋 인풋들을 저장하기위한 빠르고 작은 크기의..
- 레지스터는 여러개가있는데 그중에 프로그램 카운터(PC)라는 레지스터가있다
  - 다음번에 실행할 기계어의 메모리 주소를 가지고있다
  - 주소를 가지고 있기때문에 가르킨다고 보면됨(운영체제를 가르키면 지금 CPU가 운영체제의 기계어를 실행중이고, 사용자프로그램 A를 가르키면 그걸 실행중인것)
- 운영체제가 만약 프로그램에게 CPU를 넘겼는데 자꾸 무한루프 도는거라 CPU를 안돌려준다면 강제로 뺏어와야하는데 운영체제는 넘기는건 가능한데 가져오는건 그 프로그램이 마음대로 쓰고있기에 가져오기 불가능해서 CPU의 독점을 막기위해 운영체제를 도와주는 부가적인 하드웨어가 있는데 Timer가 있다.



- Timer는 일정시간마다 Interrupt를 발생시킴
- 즉 사용자 프로그램에게 CPU를 넘길때 Timer 시간을 세팅하고 넘김!
- CPU를 짧은시간동안 나눠서(time sharing) 쓸수있게 도와줌!

CS스터디

인터럽트란?

- CPU가 프로그램을 실행하고 있을 때, 입출력 하드웨어 등의 장치에 예외상황이 발생하여 처리가 필요할 경우에 CPU에게 알려 처리할 수 있도록 하는 것
- 우선적으로 처리해야할 일이 발생하였을 때 그것을 처리하고 원래 동작으로 돌아옴
- 내가 cpu를 운영체제로 넘기기 위해서
- 크게 하드웨어 인터럽트와 소프트웨어 인터럽트로 나눔
- 현대의 운영체제는 인터럽트에 의해 구동됨
- 운영체제가 사용자 프로그램에 CPU를 넘겼는데 다시 돌아오는 경우? 인터럽트가 걸려야 넘어감

인터럽트 라인이 세팅되어있음. 누가 세팅하느냐에 따라 두가지로 나뉨 (하드, 소프트)

- 그렇다면 프로그램이 CPU를 빼앗기게 되는 경우는 언제일까요?
- 타이머 : 시간이 지나면 자동으로 넘겨짐
- 더이상 cpu를 쓸 의지가 없음. 오래걸리는 작업을 하는 경우 등등

하드웨어 인터럽트

- **일반적**으로 인터럽트를 이르는 말
- CPU외부로부터의 인터럽트 요구 신호에 의해 발생되는 인터럽트
- Maskable interrupt, Non-maskable interrupt 가 있다. (Interrunpt Mask가 가능)
    - Maskable interruptInterrupt Mask(인터럽트가 발생하였을 때 요구를 받아들일지 말지 지정하는 것)가 가능 인텔CPU 에서 **INTR pin**으로 신호가 들어옴
    - Non Maskable interrupt Interrupt Mask가 불가능거부, 무시할 수 없음 (매우 중요함)정전, 하드웨어 고장 등 어쩔수없는 오류인텔 CPU에서 **NMI pin**으로 신호가 들어옴

하드웨어 인터럽트 종류

- 입출력 인터럽트 (I/O interrupt) - 입출력 작업의 종료나 입출력 오류에 의해 CPU의 기능이 요청됨(디스크, 키보드, 프린터, 모니터)
- 정전,전원 이상 인터럽트(Power fail interrupt) - 전원 공급의 이상
- 기계 착오 인터럽트(Machine check interrupt) - CPU의 기능적인 오류
- 외부 신호 인터럽트(External interrupt) - I/O 장치가 아닌 오퍼레이터나 타이머에 의해 의도적으로 프로그램이 중단된 경우

소프트웨어 인터럽트

- CPU내부에서 자신이 실행한 명령이나 CPU의 명령 실행에 관련된 모듈이 변화하는 경우 발생
- Trap(소프트웨어 인터럽트)

    - **Exception** : 프로그램이 오류를 범한 경우
    - **Sysetem call** : 개별 프로그램이 본인이 직접 할 수 없는 일을 운영체제한테 부탁. 커널함수 호출하는 경우
- 프로그램의 오류에 의해 생기는 인터럽트

소프트웨어 인터럽트 종류

- 프로그램 검사 인터럽트 (Program check interrupt)
    - 0으로 나누는 경우
    - OverFlow/UnderFlow
    - 페이지 부재
    - 부당한 기억장소의 참조
    - 등등...
- SVC(Supervisor Call: 감시프로그램 호출)인터럽트
    - 사용자가 프로그램을 실행시키거나 supervisior을 호출하는 동작을 수행하는 경우
    - 프로그래머에 의해 코드로 짜인 감시 프로그램을 호출하는 방식

Device Controller

I/O device controller

- 해당 i/o 장치유형을 관리하는 일종의 작은 CPU
- 제어 정보를 위해 control register, status register을 가짐
- local buffer을 가짐(일종의 data register)

I/O는 실제 device와 local buffer 사이에서 일어남

Device controller은 I/O가 끝났을 경우 interrupt로 CPU에 그 사실을 알림

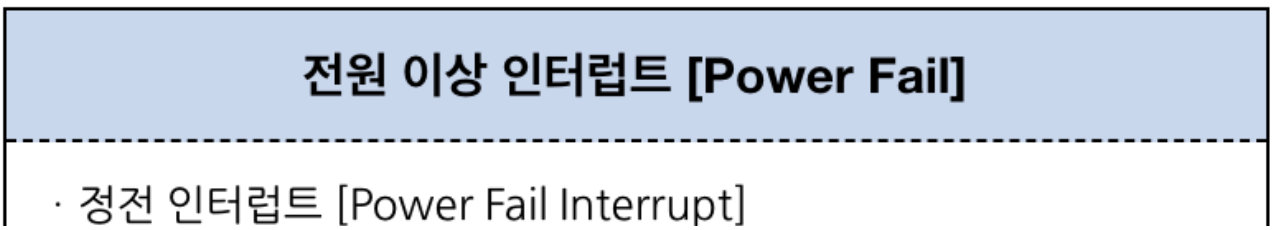
- Device Driver(장치구동기) - 운영체제 안에 있는 CPU가 이 디바이스한테 부탁하는 방법이 적혀져있음 컨트롤러가 수행하는 코드가 아니고 CPU가 수행하는 코드

:OS 코드 중 각 장치별 처리루틴 → software

- Device controller(장치제어기) - 코드가 있어서 코드에 의해서 디스크 내용을 쓰는것 펌웨어라고도 함.

:각 장치를 통제하는 일종의 작은 CPU → hardware

인터럽트 우선순위



↓

기계 고장 인터럽트[Machine Check]

· 기계 이상에 의해서 발생한 인터럽트

↓

외부 신호 인터럽트

· 타이머, · 조작원, · I/O 장치,
· 외부 프로세서 등의 요청에 의해 발생한 인터럽트

↓

프로그램 인터럽트[Program Check]

· 0으로 나누기, · Overflow/Underflow 인터럽트
· 불법적인 연산자 사용, · 보호 공간 접근
↳프로그램의 연산자나 주소 지정 방식의 잘못 으로 인한 인터럽트

↓

SVC(SuperVisor Call)

· 제어 프로그램 호출 인터럽트

관련용어 1. *인터럽트 벡터 Interrupt Vector*

- 여러가지 인터럽트에 대해 해당 인터럽트 발생시 **처리해야 할 루틴(ISR)의 주소를 보관**하고 있는 공간
- 대부분의 CPU들은 인터럽트 벡터 테이블을 가지고 있음
- 인텔x86에서는 이를 IDT(Interrupt Descriptor Table)이라고 한다.

2. *인터럽트 서비스 루틴(인터럽트 처리 루틴) Interrupt Service Routine(ISR)*

- 인터럽트 핸들러 Interrupt handler 라고도 함
- 인터럽트가 접수되면 각각의 인터럽트에 대응하여 특정 기능을 처리하는 기계어 코드 루틴(커널이 실행) 커널은 시그널 처리를 위한 다양한 시스템 콜을 지원한다.
- ex) 키보드 자판을 눌러 키보드 인터럽트가 발생하면 이에 해당하는 인터럽트 서비스 루틴이 실행됨

인터럽트 과정

> 프로세스 A 실행 중 디스크에서 어떤 데이터를 읽어오라는 명령을 받았을 때

- 프로세스 A는 system call을 통해 인터럽트를 발생시킴
- CPU는 현재 진행중인 기계어 코드를 완료
- 현재까지 수행중이었던 상태를 해당 프로세스의 **PCB(Process Control Block)**에 저장 (수행중이던 메모리 주소, 레지스터 값 등등)
- PC(Program Counter, IP)에 다음 실행할 명령의 주소 저장
- 인터럽트 벡터를 읽고 ISR 주소값을 얻어 ISR(interrupt Service Routine)으로 점프하여 루틴을 실행
- 해당 코드 실행
- 해당 일을 다 처리하면 저장했던 프로세스 상태 복구
- ISR의 끝에 IRET명령어에 의해 인터럽트가 해제
- IRET 명령어가 실행되면, 대피시킨 PC 값을 복원하여 이전 실행위치로 복원

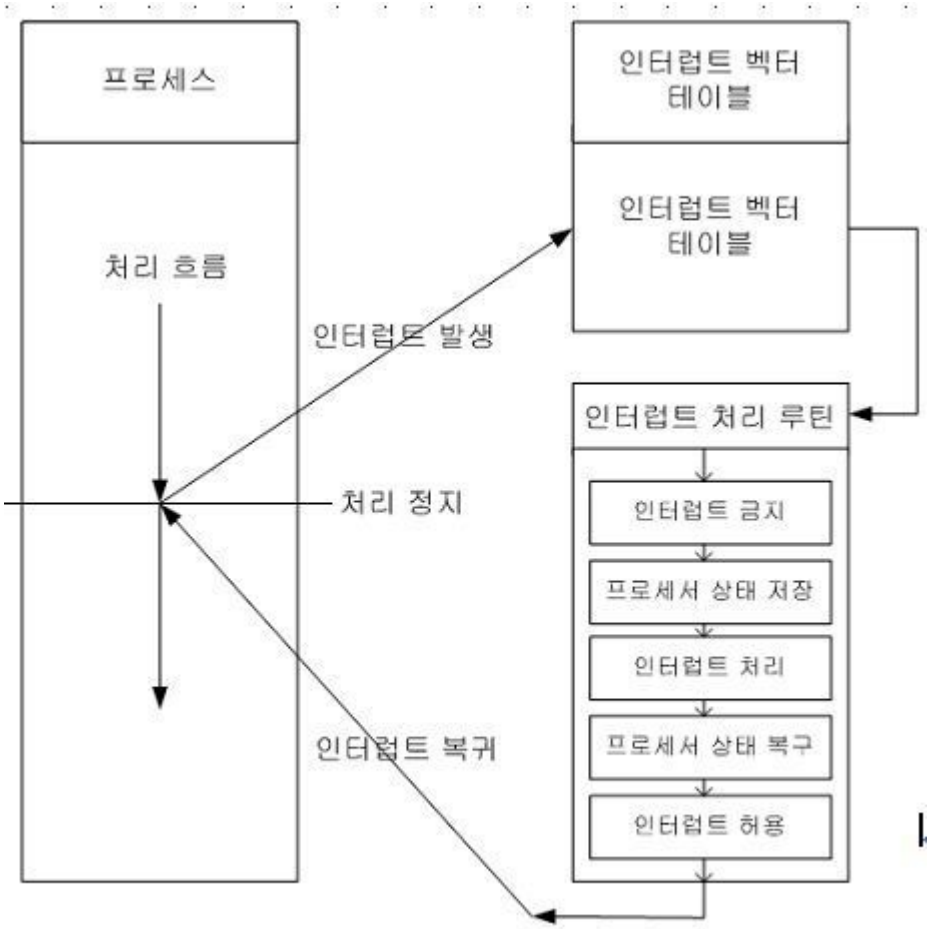


그림 1. 인터럽트 처리 흐름

- ***IRET(Interrupt Return) 명령어***: *이전 태스크로 다시 돌아가는 어셈블리 명령어. ISR의 마지막 명령어.*

동기식 입출력, 비동기식 입출력

- 동기식 입출력(synchronus I/O)

여럿이 잘 함에 있어서 시간적으로 잘 맞아떨어지는 경우. 싱크로나이즈가 맞다 이런거 들어본적 있죠?

I/O요청 후 입출력 작업이 완료된 후에 제어가 사용자 프로그램에 넘어감

구현 방법 1.

- I/O가 끝날때까지 CPU 낭비
- 매시점 하나의 I/O만 일어날 수 있음

구현방법 2.

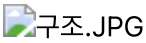
- I/O가 완료될 때 까지 해당 프로그램에게서 CPU를 빼앗음
- I/O 처리를 기다리는 중에 그 프로그램을 줄 세움
- 다른 프로그램에게 CPU를 줌
- 비동기식 입출력(asynchronous I/O)

I/O가 시작된 후 입출력 작업이 끝나기를 기다리지 않고 제어가 사용자 프로램에 즉시 넘어감.

⇒ 두 경우 모두 I/O의 완료는 인터럽트로 알려줌

DMA(Direct Memory Access)

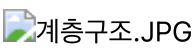
- 빠른 입출력 장치를 메모리에 가까운 속도로 처리하기 위해 사용 / 인터럽트가 자주 걸리면?
- CPU의 중재 없이 device controller가 device의 buffer storage 내용을 메모리에 block 단위로 직접 전송 / 디스크 컨트롤러는 메모리에 직접 넣을 수 없음 오직 CPU만 가능함.
- 바이트 단위가 아니라 block 단위로 인터럽트 발생시킴



CPU에서 쓰는 건 기계어 special instruction

I/O를 수행하는 기계어는

메모리 접근하는 기계어가 또 따로 있음.



CPU부터 ~ 위로갈수록 용량 작으면서 빠르고 비쌈 / 아래로 갈 수록 저렴.

실제 모든 데이터들은 제일 아래쪽에 저장되어있고 필요할때 위로 가져가 쓰도록 되어있음

필요할 때 제일 아래쪽까지 와서 들고가면 너무 느림

Cashing의 원리 : 밑에서부터 필요한것들을 올릴 수 있도록