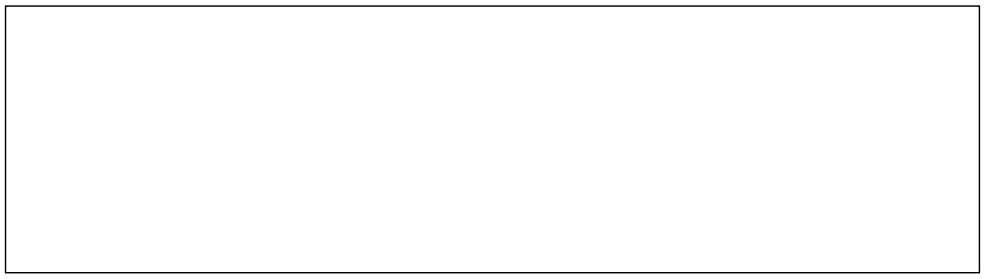
LINUX KERNEL



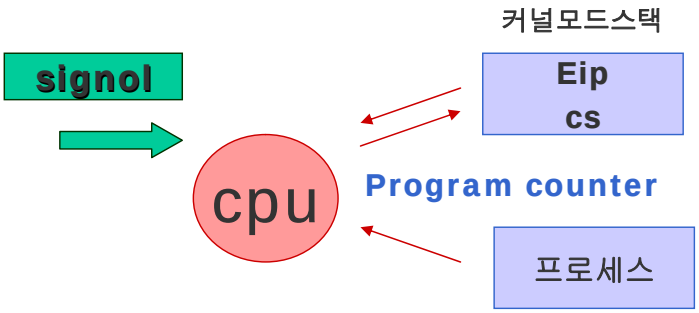
OverView

- **Interrupt and Exception**
  - Interrupt signal 역할
  - Interrupt and exception
  - Exception handler 와 Interrupt handler의 중첩 실행
  - Interrupt Descriptor Table 초기화
  - Exception Handling
  - Interrupt Handling
  - Interrupt 와 Exception에서의 복귀

인터럽트와 예외

· 프로세스가 실행하는 명령의 순서를 바꾸는 사건
· 하드웨어가 발생시키는 전기적 시그널

동기적 인터럽트 - 예외
비동기적 인터럽트 – 인터럽트 ─ • 인터럽트시그날



- 상반부(top half)
- 하반부(bottom half)
- 인터럽트 중첩허용
- 임계구역 (뒤쪽에 설명)

프로세스를 대신하여 실행하는 커널 제어 경로

- 인터럽트

마스킹 가능한 인터럽트

- 마이크로 프로세서의 INTR핀으로 전달하는 인터럽트
- eflags레지스터에서 IF플레그를 를 0으로 꺼서 금지 시킬수 있다
- 입출력 장치에서 발생하는 모든 IRQ

마스킹 불가능한 인터럽트

- eflags레지스터의 IF플레그를 0으로 꺼도 금지되지 않는다.
- 하드웨어 고장같은 몇몇 중요한 사건망이 마스킹 불가능한 인터럽로 발생한다.

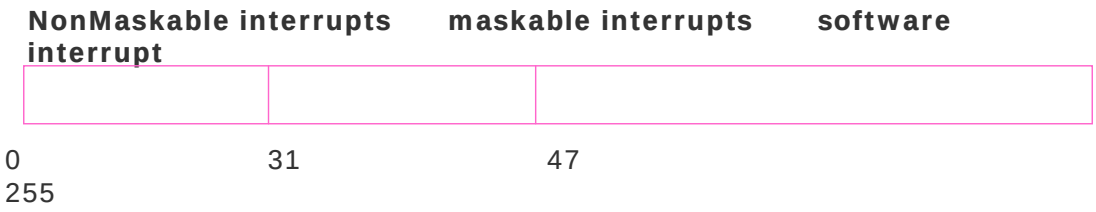
예외 (eip레지스터의 값에 따라 3분류)

폴트 – 예외 처리후 같은명령어복귀
트랩 – 예외 처리후 다음 명령어복귀
중단 - 문제가 발생한 프로세스의 강제종료

프로그래밍에 의한 예외

into와 bound 검사조건이 맞지 않을 경우 프로그램 예외
시스템 콜을 구연하기 위한것

인터럽트와 예외 벡터



트랩처리 핸들러 디바이스에서 발생하는 시스템 콜 핸들러 **(0x08)**

인터럽트 처리 핸들러

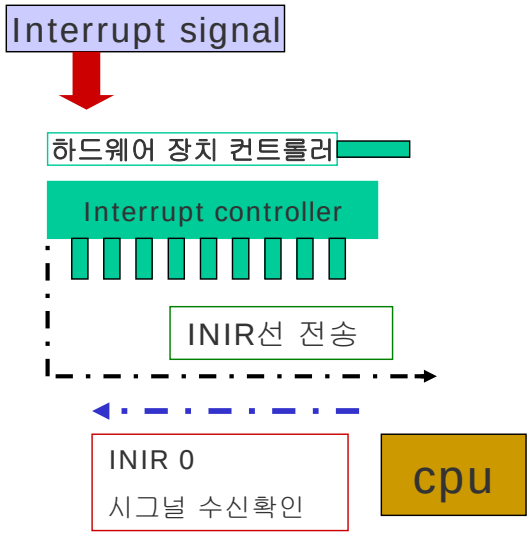
- 0 – 31 마스킹 불가능한 인터럽트(시스템 예약)
- 32 – 47 마스킹 가능한 인터럽트 (IRQ발생시킨 인터럽트)
- 48부터 255까지 소프트웨어 인터럽트
  리눅스에서는 128(ox80벡터 하나만 사용)

iRQ와 인터럽트

▶ 인터럽트 요구를 발생시킬 수 있는 하드웨어는 **IRQ** 출력선 갖음

▶ **IRQ** 선을 **'Interrupt controller'** 라고 하는 하드웨어 회로의 입력 핀으로 연결

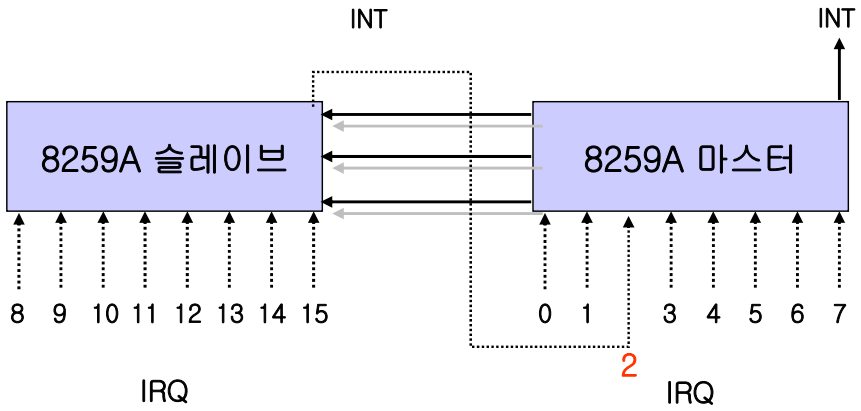
▶ **Interrupt controller**가 하는 일



① IRQ 선 감시, 시그널이 발생했을 때 이를 검사
② IRQ 선에서 발생한 시그널이라면
- 시그널을 해당 **벡터로 변환**
- 벡터를 IC의 **입출력 포트에** 저장
  (cpu가 데이터 버스를 통해 읽을 수 있게 한다)
- 발생한 시그널을 INTR 핀으로 보낸다
- CPU가 인터럽트 시그널을 받았음을 IC I/O port에 알린다.
- INTR 선을 0으로 한다

③ 모든 작업이 끝나면 1단계로 돌아간다

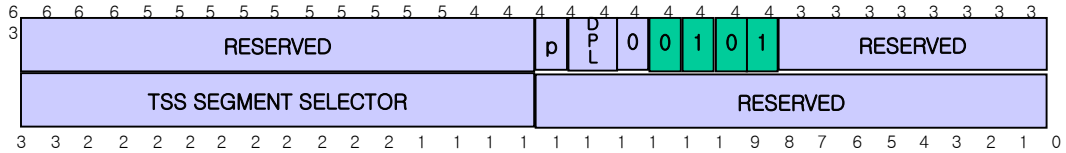
인텔 8259A PIC를 중첩 시켜 서로 다른 IRQ 입력 선 15를 처리할 수 있도록 연결한 것(전통적인 PIC)



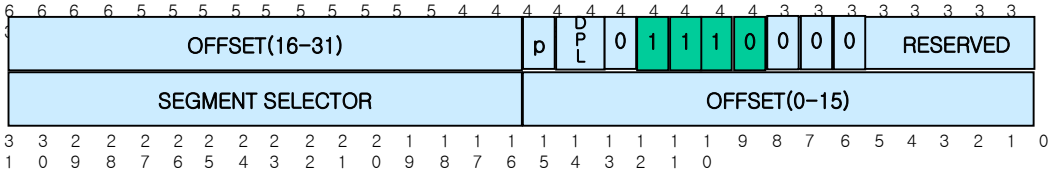
#	예외	예외 핸들러	시그널
0	나누기 에러	divide_error()	SIGFPE
1	디버그	debut()	SIGTRAP
2	NMI	nmi()	없음
3	중지점	int3()	SIGTRAP
4	오버플로우	overflow()	SIGSEGV
5	범위 검사	bounds()	SIGSEGV
6	잘못된 연산 코드	invalid_op()	SIGILL
7	장치를 사용할 수 없음	device_not_available9)	SIGSEGV
8	이중 폴트	double_fault()	SIGSEGV
9	보조프로세스 세그먼트 범람	coprocessor_segment_overrun()	SIGFPE
10	잘못된 TSS	invalid_tss()	SIGSEGV
11	세그먼트가 존재하지 않음	segment_not_present()	SIGBUS
12	스택 세그먼트	stack_segment()	SIGBUS
13	일반 보호	general_protection()	SIGSEGV
14	페이지 폴트	page_fault()	SIGSEGV
15	인텔이 예약함	없음	없음
16	부동소수점 에러	coprocessor_error()	SIGFPE
17	정렬검사	alignment_check()	SIGSEGV
18-31	개발용 예약	개발용 예약	개발용 예약

인터럽트 디스크립터 테이블

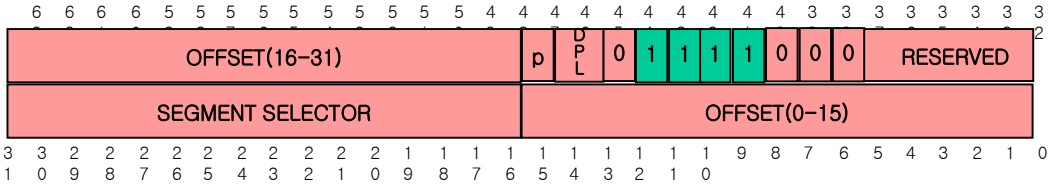
- 태스크 게이트 – **TTS** 셀렉터를 포함



- 인터럽트 게이트 – **IF** 플레그를 **0** 으로 꺼서 마스킹 가능한 인터럽트가 더 이상 발생 하지 않도록 금지



- 트랩 게이트 – 세그먼트로 제어를 넘길때 **IF** 플레그를 바꾸지 않는다



2.7 하드웨어적인 인터럽트와 예외 처리

▶ 제어유닛은 이전 명령어를 실행하는 동안 인터럽트나 예외가 발생했는지 검사

▶ 만약 발생했다면 아래의 동작을 취한다

① 발생한 인터럽트나 예외에 해당하는 벨터를 알아낸다
② IDT에서 백터 I의 엔트리를 읽어 들인다.
③ 인터럽트나 예외 핸들러를 포함한 세그먼트의 기본 주소를 지정
④ 인증된 소스에서 인터럽트가 발생했는지 검사
⑤ 권한 수준 변화 확인
- 현재 프로세스의 TSS 세그먼트에 접근
- 스택 세그먼트와 스택 포인터의 적당한 값을 설정
⑥ 예외 발생 명령의 재수행이 가능하도록 설정
⑦ 레지스터의 내용을 스택에 저장
⑧ 하드웨어 에러 코드를 스택에 저장
⑨ 레지스터를 IDT의 세그먼트와 오프셋으로 지정

▶ 커널 제어 경로
- 커널 모드에서 인터럽트와 예외를 처리하는 일련의 명령어들
- 첫째 명령어는 커널 모드 스택에 레지스터의 내용을 저장
- 마지막 명령어는 커널 모드 스택에 있는 내용을 복구

▶ 예외의 대부분은 CPU가 사용자 모드에 있을 때 발생
(프로그래밍 에러나 디버거가 발생)

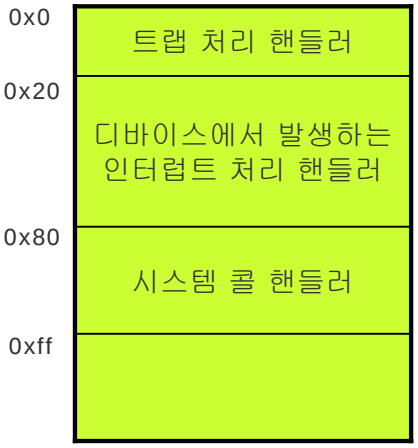
▶ 페이지 폴트는 커널 제어 경로를 최대 두 개까지 중첩(커널모드)

▶ 커널 제어 경로 중첩을 허용하는 이유
- 프로그래밍 가능한 인터럽트 컨트롤러와 장치 컨트롤러가 처리량을 늘리기 위해서
- 우선순위가 없는 인터럽트 모델을 개발하기 위해서
(커널 코드를 간단하게 하고 호환성을 높여준다)

IDT(interrupt descriptor table) 초기화

- IDT? Interrupt와 exception 처리에 사용되는 핸들러를 가지고 있는 자료구조

- 시스템을 초기화하는 동안 Kernel은 interrupt를 허용하기 전에 IDT의 시작 주소를 idtr 레지스터에 저장하고 이 테이블의 모든 엔트리를 초기화 한다.



- Setup code(linux/arch/kernel/head.S)

```
/*
 *  setup_idt
 *
 *  sets up a idt with 256 entries pointing to
 *  ignore_int, interrupt gates. It doesn't actually load
 *  idt - that can be done only after paging has been enabled
 *  and the kernel moved to PAGE_OFFSET. Interrupts
 *  are enabled elsewhere, when we can be relatively
 *  sure everything is ok.
 */
setup_idt:
        lea ignore_int,%edx
        movl $(__KERNEL_CS << 16),%eax
        movw %dx,%ax            /* selector = 0x0010 = cs */
        movw $0x8E00,%dx        /* interrupt gate - dpl=0, present */

        lea SYMBOL_NAME(idt_table),%edi
        mov $256,%ecx
rp_sidt:
        movl %eax,(%edi)
        movl %edx,4(%edi)
        addl $8,%edi
        dec %ecx
        jne rp_sidt
        ret
```

- idt를 ignore_int를 향하도록 하여 interrupt를 무시하도록 한다.
- table의 나머지 영역을 똑같이 채운다.

- Gate
  - Interrupt gate
    - user mode에서 접근 불가
    - linux interrupt handler 는 interrupt gate를 통해서만 실행.
    - kernel mode로 제한
  - System gate
    - user mode에서 접근 가능
    - int 3, into, bound, int 0x80 명령어 사용 가능
  - Trap gate
    - user mode에서 접근 불가
    - systme gate를 제외한 모든 예외 핸들러 실행

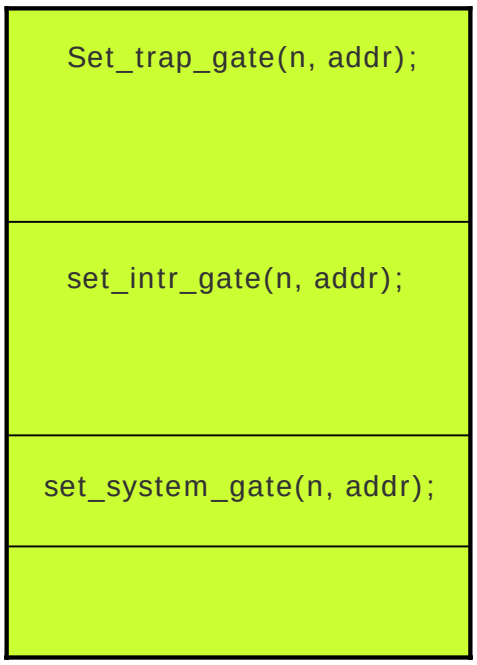
- 참고 – protect mode

- CPU가 제공하는 모든 기능과 명령어들을 활용 가능.

- privilege level
  프로세서를 활용할 수 있는 권한. 리눅스에서 0과 3만 사용
  - 0 : 모든 일을 할 수 있는 모드. 커널모드.
  - 3 : 사용자 권한

- 인터럽트나 예외처리, task switching 등도 모두 보호모드에서 지원하는 기능을 활용한다.

- IDT에 gate 삽입



IDT

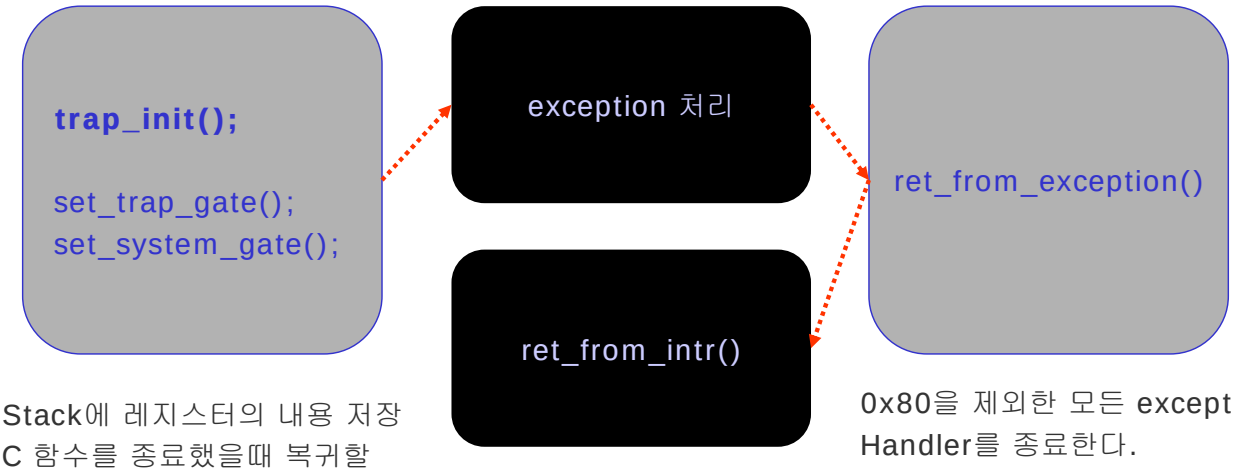
```
void __init trap_init(void)
{
#ifdef CONFIG_EISA
        if (isa_readl(0x0FFFD9) == 'E'+('I'<<8)+('S'<<16)+('A'<<24))
                EISA_bus = 1;
#endif

        set_trap_gate(0,&divide_error);
        set_trap_gate(1,&debug);
        set_intr_gate(2,&nmi);
        set_system_gate(3,&int3);       /* int3-5 can be called from all */
        set_system_gate(4,&overflow);
        set_system_gate(5,&bounds);
        set_trap_gate(6,&invalid_op);
        set_trap_gate(7,&device_not_available);
        set_trap_gate(8,&double_fault);
        set_trap_gate(9,&coprocessor_segment_overrun);
        set_trap_gate(10,&invalid_TSS);
        set_trap_gate(11,&segment_not_present);
        set_trap_gate(12,&stack_segment);
        set_trap_gate(13,&general_protection);
        set_intr_gate(14,&page_fault);
        set_trap_gate(15,&spurious_interrupt_bug);
        set_trap_gate(16,&coprocessor_error);
        set_trap_gate(17,&alignment_check);
        set_trap_gate(18,&machine_check);
        set_trap_gate(19,&simd_coprocessor_error);

        set_system_gate(SYSCALL_VECTOR,&system_call);
```

/linux/arch/kernel/trap.c

Exception Handling

- 목적

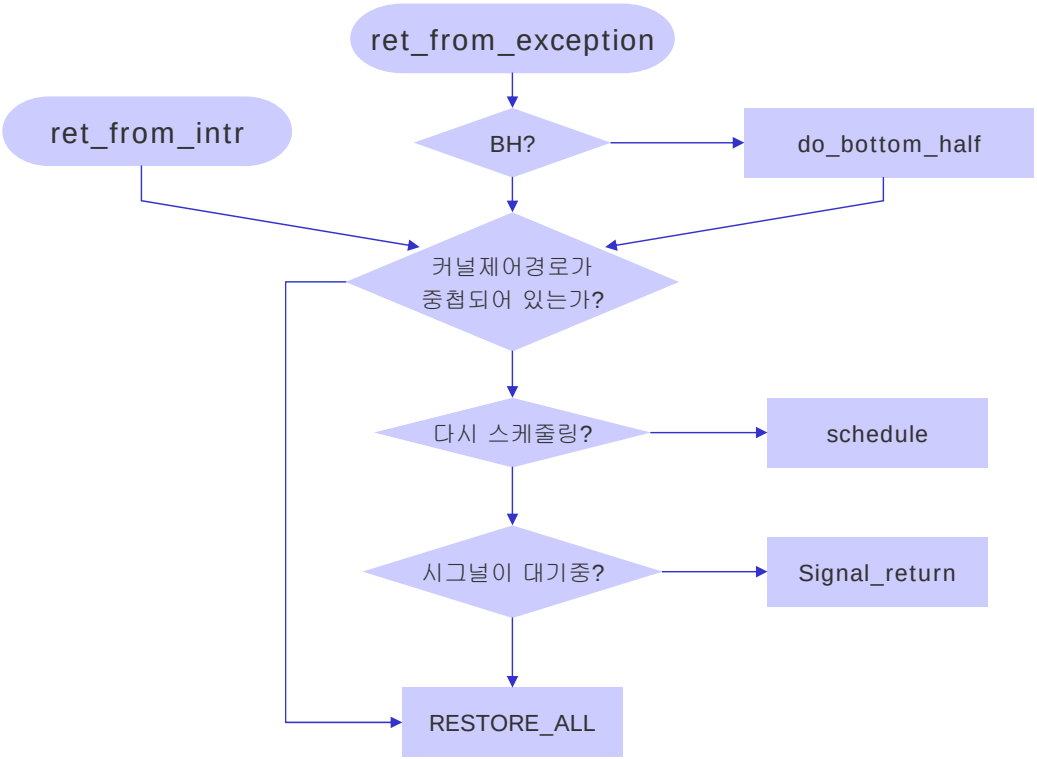
  프로세서에 비정상적인 동작을 알려주기 위함



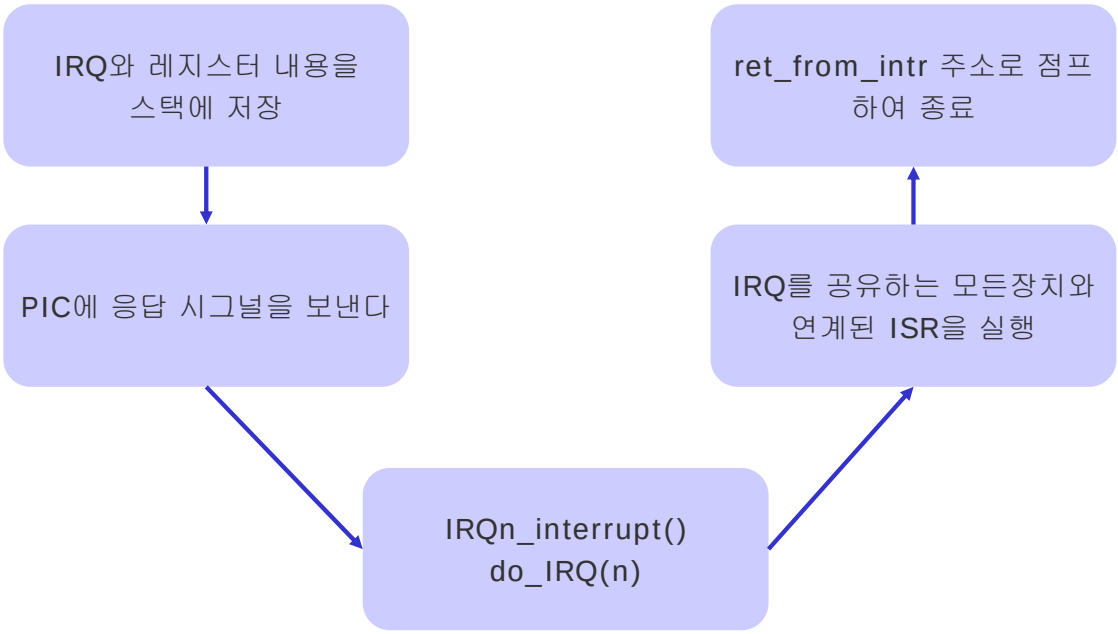
Stack에 레지스터의 내용 저장
C 함수를 종료했을때 복귀할 명령어의 주소
하드웨어 에러 코드

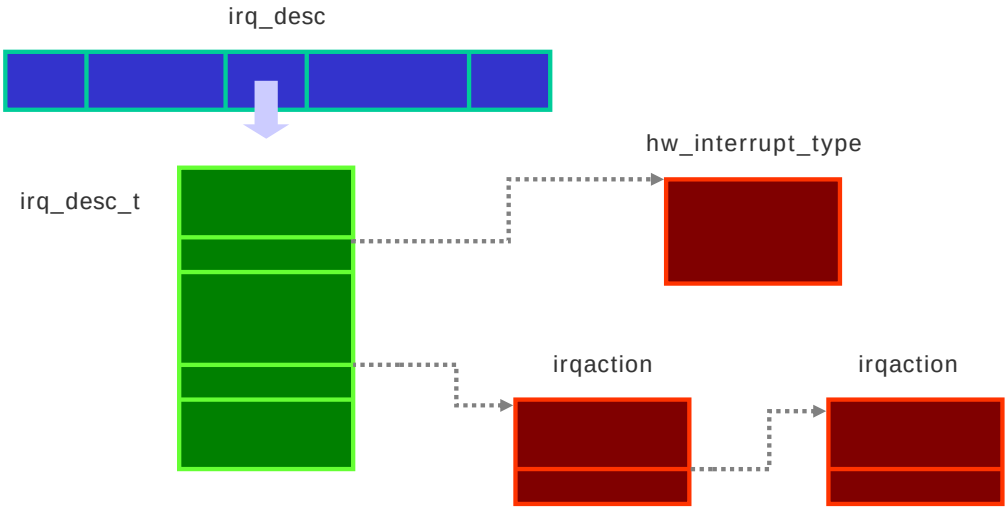
예외는 인터럽트와 같은 방식으로 종료한다.

0x80을 제외한 모든 exception Handler를 종료한다.



Interrupt handling

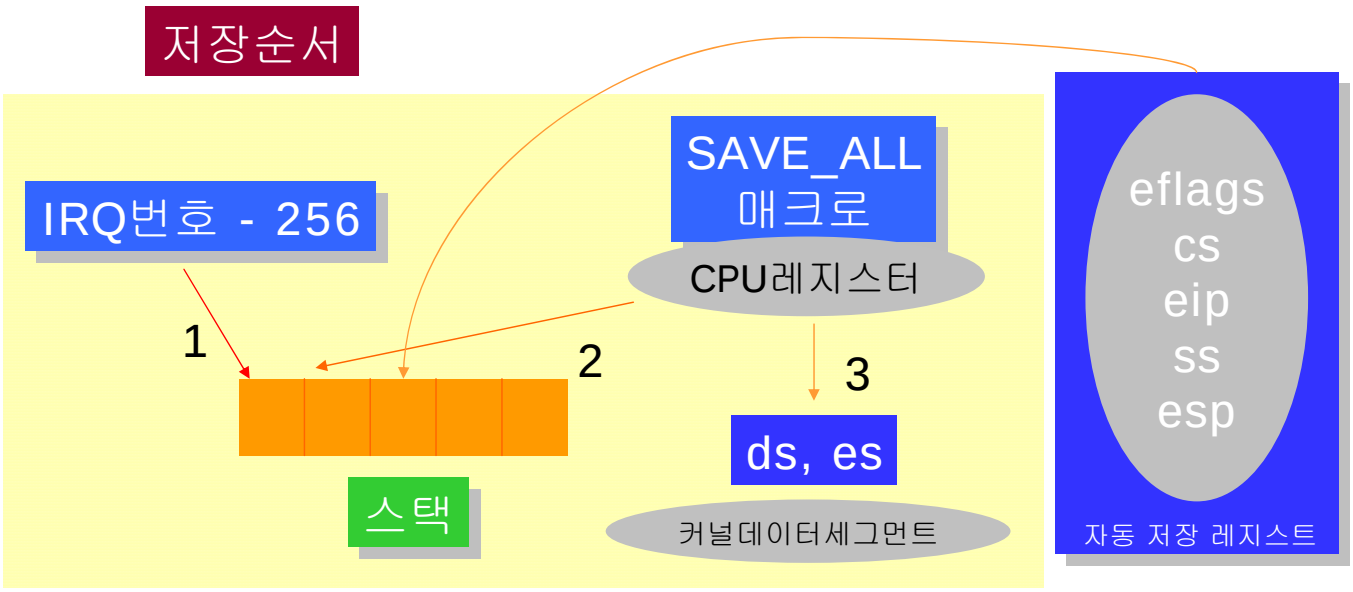




- irq_desc – irq_desc_t 의 배열
  - status, handler, action, depth
- hw_interrupt_type – 특정 PIC 회로에 동작하는 저수준 입출력함수에 관한 여러 개의 포인터
- irqaction – 여러 개의 장치가 하나의 IRQ를 공유할 수 있기 때문에 irqaction 하나가 특정 장치와 특정 인터럽트를 가리키게 한다.
  - handler, flags, name, dev_id, next

인터럽트 핸들러를 위한 레지스트 저장

- 레지스터 저장은 인터럽트 핸들러에서 처음으로 해야 할 작업



BUILD_COMMON_IRQ는 do_IRQ() 함수 호출
ret_from_intr()주소로 점프

do_IRQ()

- 인터럽트와 연계된 모든 인터럽트 서비스 루틴을 실행하기 위한 호출함수

인터럽트 서비스 루틴

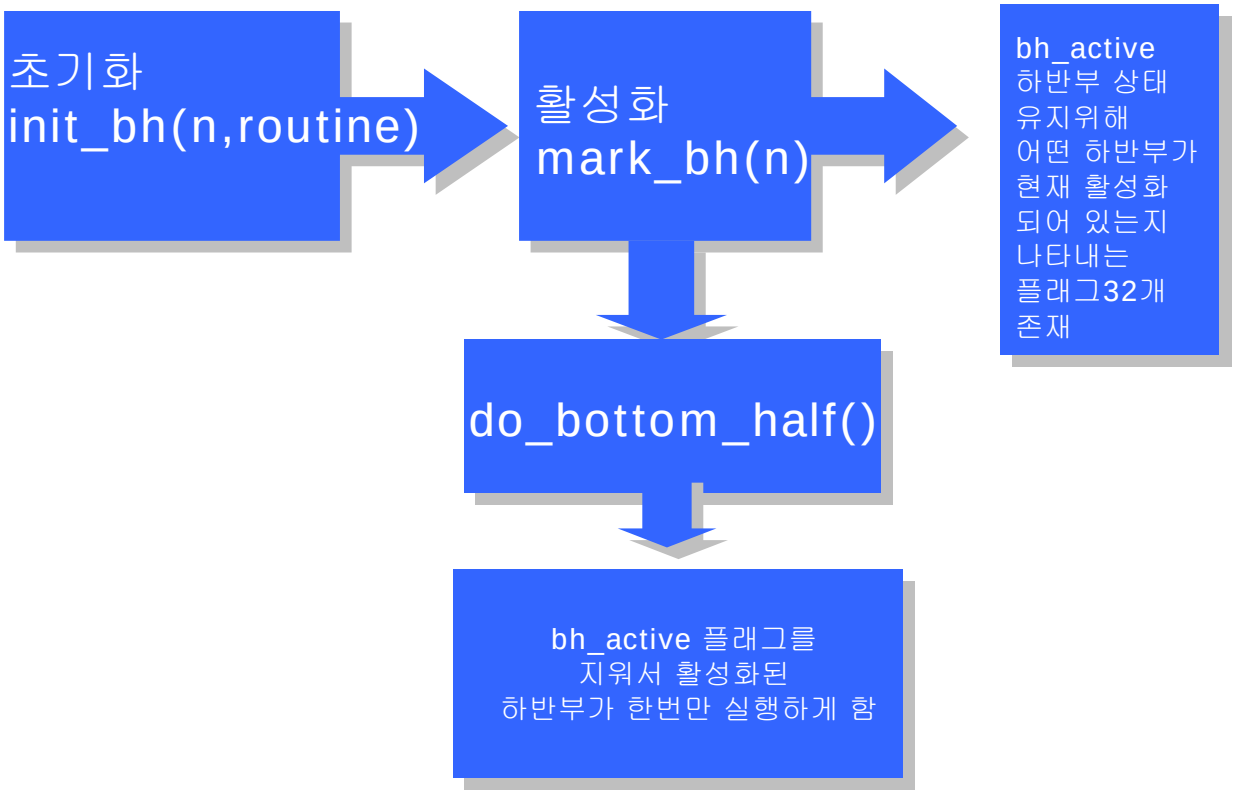
- 모든 인터럽트 서비스 루틴은 같은 매개변수로 작업

- 매개 변수
  - irq
    - IRQ 번호
    - ISR 하나가 여러 IRQ선을 다룰 수 있음
  - dev_id
    - 장치 식별자
    - ISR 하나가 같은 IRQ를 사용하는 여러 장치 구별
  - regs
    - 인터럽트가 발생 직후 저장한 레지스터를 포함한 커널 모드 스택영역 포인터
    - ISR이 인터럽트된 커널 제어 경로의 실행 컨텍스트에 접근할 수 있도록 함

하 반 부(bottom half)

- 하반부 필요성
  커널이 여러 장치에서 발생하는 인터럽트를 빨리 처리해야 하기 때문
- bh_base 테이블
  모든 하반부 핸들러를 함께 모아서 관리하는 배열

하반부 상태 활성화, 추적



하반부 상태 활성화, 추적

- 마스킹(masking)
  - 하반부가 활성화 되 있더라도 마스킹되면 실행하지 않음
  - bh_mask변수에 표시
  - disable_bh(n), enable_bh(n)
    - bh_mask의 n째 비트를 조작하여 하반부를 마스킹 또는 마스킹 해제
  - 마스킹 필요성
    - 경쟁상태회피
      - 예) 예외가 발생시