
Embedded System Design

실습 6

Cho Yeongpil
Hanyang University



VPOS 커널을 포팅하기 위한 준비

1. 커널 컴파일 + 커널 이미지를 RAM에 적재
2. Startup code 작성
3. UART 설정
4. TIMER 설정
5. **Hardware Interrupt Handler 구현**
(1) UART Interrupt
6. Software Interrupt Entering/Leaving Routine 구현
7. Timer Interrupt



HARDWARE INTERRUPT



목차

1. Exception
2. Interrupt
3. Vectored Interrupt Controller
4. Interrupt Entering & leaving Routine
5. UART Interrupt Handler
6. Homework



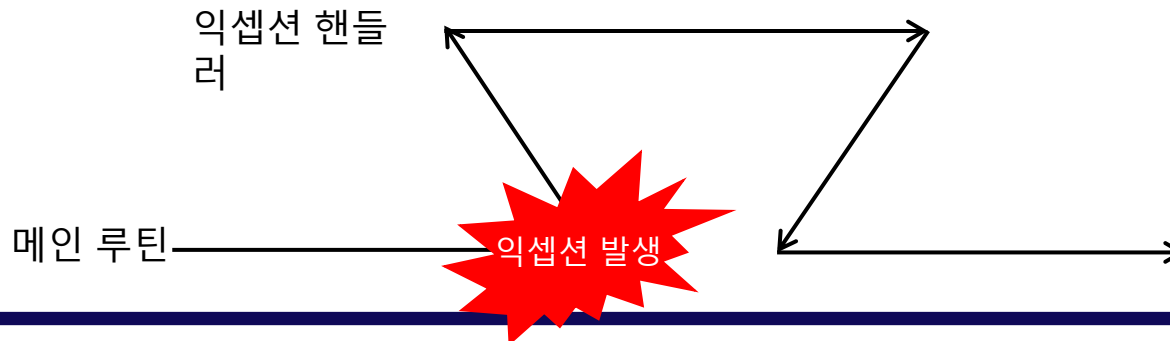
EXCEPTION



Exception

- **Exception?**

- 명령어들의 순차적인 실행 과정을 중단시켜야 하는 상태
- 예외 상황이나 인터럽트가 발생한 상태
 - 정의되어 있지 않은 명령어
 - 메모리 액세스 실패
 - 소프트웨어 인터럽트
 - 외부 하드웨어 인터럽트
 - ...
- 기존에 실행하고 있던 루틴에서 벗어나 익셉션 핸들러(Exception Handler)로 이동



Exception의 종류

Exception	Mode	Vector table offset
Reset	SVC	+0x00
Undefined Instruction	UND	+0x04
Software Interrupt(SWI)	SVC	+0x08
Prefetch Abort	ABT	+0x0c
Data Abort	ABT	+0x10
Not assigned	-	+0x14
IRQ	IRQ	+0x18
FIQ	FIQ	+0x1c



Entering Exception Handler

- 익셉션이 발생하면 CPU는 자동으로,
 1. CPSR 값을 익셉션 모드의 SPSR에 저장
 2. PC 값을 익셉션 모드의 Link Register(lr)에 저장
 3. CPSR의 모드 비트를 변경하여 해당 익셉션 모드로 진입
 4. PC에 익셉션 핸들러의 주소를 저장하여 익셉션 핸들러를 실행
 - Vector table base address + Vector table offset
 - Vector table base address는 ARM Coprocessor에 저장

```
vh_vector_start:
    b      vh_UPOS_reset
    b      vk_undef
    b      vh_software_interrupt
    b      vh_pabort
    b      vk_dabort
    b      vk_not_used_handler
    b      vh_irq
    b      vk_fiq_handler
```

Vector Table

```
// change vector table base address (0x20008044)
ldr    r0, =vh_vector_base
mcr    p15, 0, r0, c12, c0, 0
```

Vector base address 저장

Exception Handler

- **익셉션 핸들러(Exception Handler)**
 - 익셉션의 원인을 찾아 해당 익셉션을 처리
 - 익셉션이 인터럽트라면, 인터럽트를 처리하고 메인 루틴으로 복귀해야 함
- **익셉션 처리를 끝내고 원래 루틴으로 돌아오려면,**
 - `movs pc, lr`
 - Link Register의 값을 PC에 저장
 - Link register의 값은 익셉션에 따라 추가 계산 필요
 - 익셉션 모드의 SPSR을 CPSR에 저장



INTERRUPT



Polling vs. Interrupt

- 지금 디바이스를 사용할 수 있는지 어떻게 알 수 있는가?
 - Polling & Interrupt
- **Polling**
 - CPU가 디바이스의 상태를 일정 주기마다 확인
 - 상태 비트(status bit)의 값을 확인
- **Interrupt**
 - 디바이스가 데이터를 보낼 수 있거나 외부로부터 데이터를 수신하면 CPU에게 알림
 - CPU는 인터럽트가 발생할 때까지 다른 일을 할 수 있음



Interrupt

- **ARM의 Interrupt**

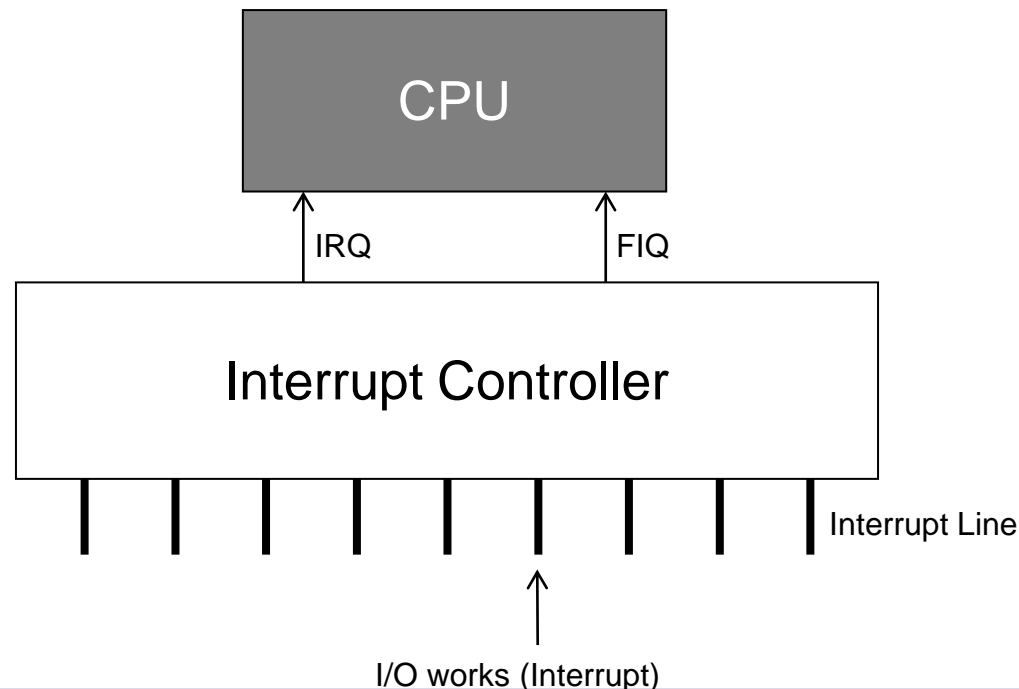
- IRQ : Normal Interrupt
 - 범용 인터럽트
 - FIQ보다 낮은 우선순위와 높은 우선순위 지연 시간
- FIQ : Fast Interrupt
 - 빠른 응답 시간을 요하는 인터럽트 소스에 사용
 - 특정 어플리케이션을 위해서만 사용
- SWI : Software Interrupt
 - 특권 모드로 진입하기 위한 소프트웨어 인터럽트
 - 커널 함수를 호출하기 위해 사용



Interrupt Controller

- 정의

- CPU의 인터럽트 요청 포트 중 하나에 여러 외부 인터럽트를 연결하기 위한 제어장치



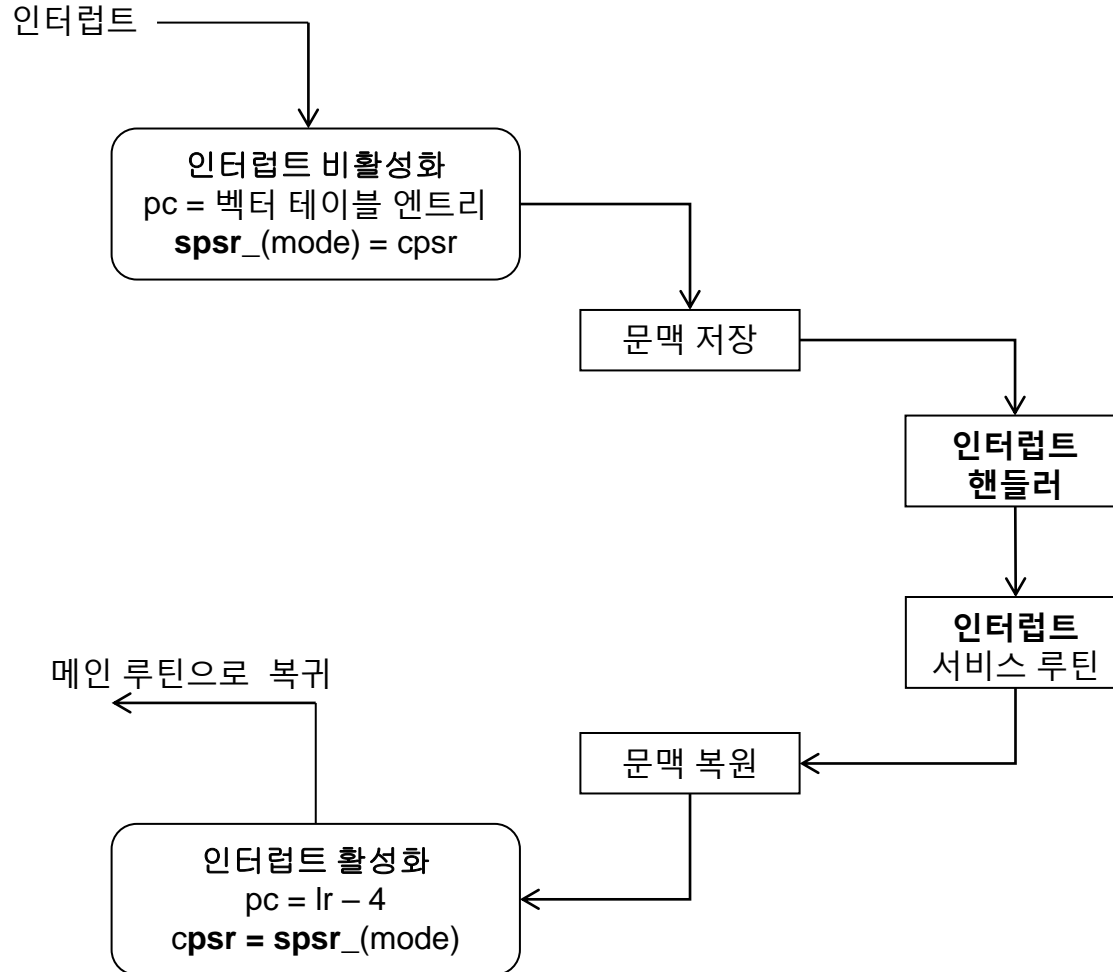
Interrupt Service Routine

- 정의

- 인터럽트가 발생하였을 때 해당 인터럽트를 처리하는 루틴
 - 타이머 인터럽트 → 스케줄러 호출
 - UART 인터럽트 → 외부에서 전송된 데이터를 수신



Interrupt 처리 방법



Interrupt 처리 방법

1. 디바이스에서 인터럽트 발생
2. ARM CPU는 IRQ 모드로 변경 (CPU가 자동으로 처리)
 - 인터럽트 비활성화(disable)
 - CPSR 값을 SPSR에 저장
 - 벡터 테이블의 인터럽트 핸들러로 이동
3. 문맥 저장
 - 이전 모드(User 모드)의 r0-r12, sp, lr 값을 스택에 저장 (stmfd)
4. 인터럽트 핸들러에서 인터럽트 소스를 찾아 해당 인터럽트 서비스 루틴(ISR) 실행
5. 인터럽트 서비스 루틴을 실행하여 인터럽트를 처리



Interrupt 처리 방법

6. 문맥 복원

- 스택에 저장한 r0-r12, sp, lr값을 이전 모드의 각 레지스터에 로드 (ldmfd)

7. 인터럽트 활성화

8. $lr = lr - 4$

- 파이프라인(pipeline)
- IRQ는 현재 명령어가 실행된 후에 발생하기 때문에 복귀할 주소는 다음 명령어, 즉 $lr-4$ 의 값을 복귀 주소로 사용해야 함

9. 'movs pc, lr' 명령어를 사용하여 원래 루틴으로 복귀



VECTORED INTERRUPT CONTROLLER

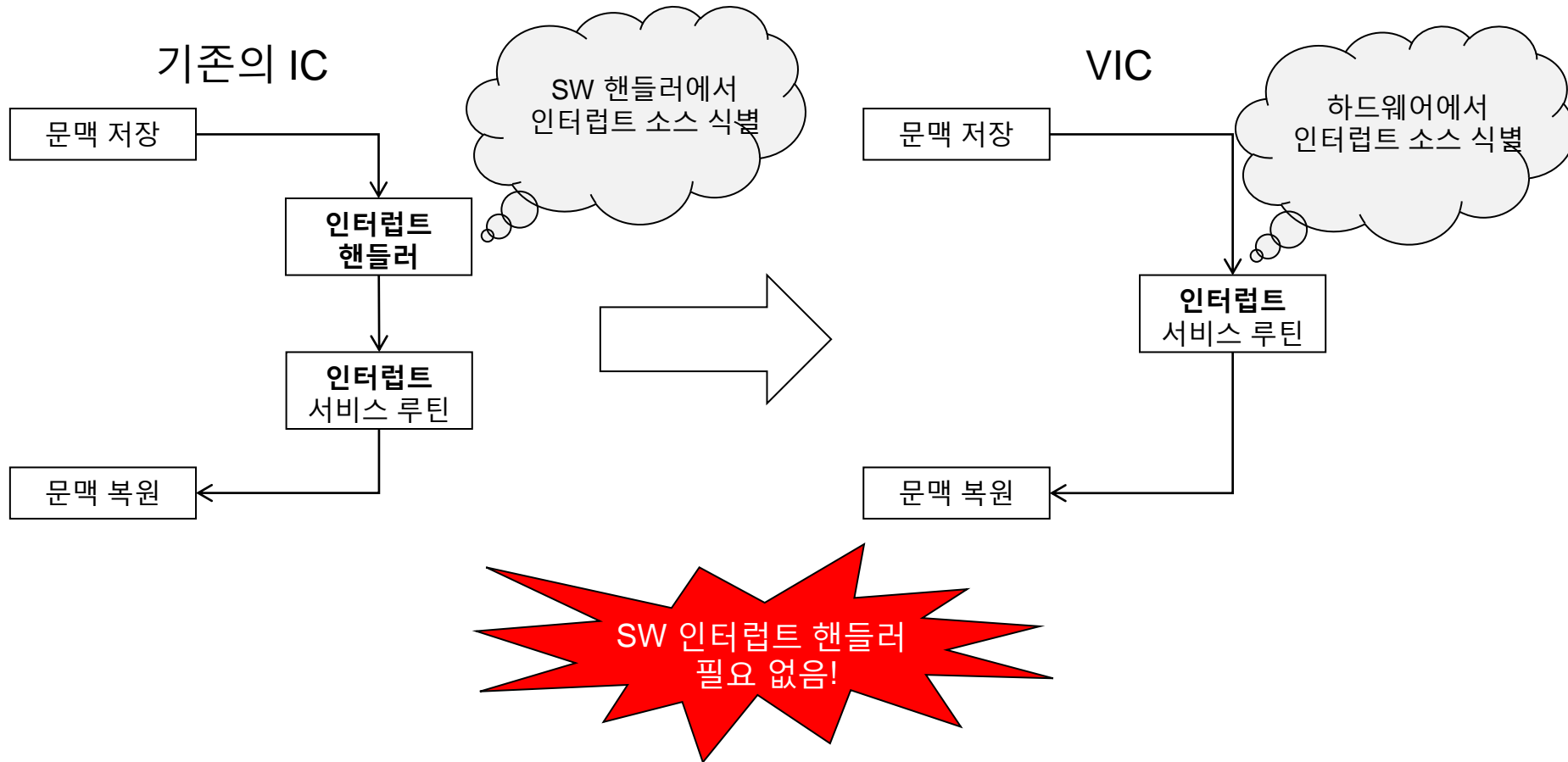


Vectored Interrupt Controller

- **VIC (Vectored Interrupt Controller)**
 - Vector방식의 interrupt 레지스터 세트를 지원
 - Interrupt source마다 레지스터를 하나씩 할당
 - 각 레지스터에는 인터럽트 서비스 루틴의 주소를 저장
 - 인터럽트가 발생하면 controller는 해당 인터럽트의 서비스 루틴 주소를 VICADDRESS 레지스터에 자동으로 로드
 - CPU는 VICADDRESS 레지스터에 저장된 인터럽트 서비스 루틴으로 점프



Vectored Interrupt Controller



Vectored Interrupt Controller

- **장점**

- 인터럽트 지연시간을 줄일 수 있음
 - 다른 Interrupt Controller에서는 소프트웨어 인터럽트 핸들러에서 인터럽트 소스를 식별
 - VIC에서는 소프트웨어 인터럽트 핸들러가 필요 없음
 - 하드웨어 상에서 인터럽트 소스를 알 수 있음
 - 다른 IC보다 더 빠르게 ISR을 실행할 수 있음



S5PC100의 VIC

- 3개의 VIC 존재
 - 하나의 VIC마다 32개의 인터럽트 소스 지원
- 인터럽트 소스(Interrupt Source) 목록
 - Datasheet Page. 360~362 참고

VIC1

ARM, power,
memory,
Connectivity,
Storage

50	IrDA	IrDA Interrupt
49	SPI2	SPI2 Interrupt
48	SPI1	SPI1 Interrupt
47	SPI0	SPI0 Interrupt
46	I2C0	I2C0 Interrupt
45	UART3	UART3 Interrupt
44	UART2	UART2 Interrupt
43	UART1	UART1 Interrupt
42	UART0	UART0 Interrupt
41	CFC	CFCON Interrupt
40	NFC	NFCON Interrupt

VIC 전체에서 43번
VIC 1에서 11번

VIC의 레지스터

- 제어 레지스터
 - VICINTSELECT
 - VICINTENABLE
 - VICINTENCLEAR
 - VICSWPRIORITYMASK
- 상태 레지스터
 - VICIRQSTATUS
- ISR 주소 저장 레지스터
 - VICVECTADDR[0-31]
 - VICADDRESS



VICINTSELECT

- 레지스터 : VICINTSELECT

- Interrupt Select Register

- Interrupt Source를 IRQ/FIQ로 설정하는 레지스터
 - 하나의 인터럽트 소스는 하나의 비트와 대응
ex) 0번 인터럽트 소스는 0번 비트에 대응

VICINTSELECT	Bit	Description	Reset Value
IntSelect	[31:0]	Selects interrupt type for interrupt request: 0 = IRQ interrupt 1 = FIQ interrupt There is one bit of the register for each interrupt source.	0x00000000

VICINTENABLE

- 레지스터 : VICINTENABLE

- Interrupt Enable Register

- Interrupt Source를 활성화(enable)시켜주는 레지스터
 - 비활성화(disable)하는 기능은 없음

VICINTENABLE	Bit	Description	Reset Value
IntEnable	[31:0]	<p>Enables the interrupt request lines, which allows the interrupts to reach the processor.</p> <p>Read: 0 = Disables Interrupt 1 = Enables Interrupt</p> <p>Use this register to enable interrupt. The VICINTENCLEAR Register must be used to disable the interrupt enable.</p> <p>Write: 0 = No effect 1 = Enables Interrupt.</p> <p>On reset, all interrupts are disabled.</p> <p>There is one bit of the register for each interrupt source.</p>	0x00000000

VICINTENCLEAR

- **VICINTENCLEAR**

- Interrupt Enabler Clear Register

- Interrupt Source를 비활성화(disable)시켜주는 레지스터

VICINTENCLEAR	Bit	Description	Reset Value
IntEnable Clear	[31:0]	<p>Clears corresponding bits in the VICINTENABLE Register:</p> <p>0 = No effect 1 = Disables Interrupt in VICINTENABLE Register.</p> <p>There is one bit of the register for each interrupt source.</p>	-



VICSWPRIORITYMASK

- **VICSWPRIORITYMASK**

- Software Priority Mask Register

- 16개 우선순위 레벨의 인터럽트들을 mask하는 레지스터

VICSWPRIORITYMASK	Bit	Description	Reset Value
Reserved	[31:16]	Reserved, read as 0, do not modify	0x0
SWPriorityMask	[15:0]	Controls software masking of the 16 interrupt priority levels: 0 = Interrupt priority level is masked 1 = Interrupt priority level is not masked Each bit of the register is applied to each of the 16 interrupt priority levels.	0xFFFF

VICIRQSTATUS

- 레지스터 : VICIRQSTATUS

- IRQ Status Register

- 어떤 Interrupt가 발생했는지 나타내는 레지스터
 - 특정 Interrupt가 발생하면 해당 interrupt를 가리키는 비트가 1로 set

VICIRQSTATUS	Bit	Description	Reset Value
IRQStatus	[31:0]	Shows the status of the interrupts after masking by the VICINTENABLE and VICINTSELECT Registers: 0 = Interrupt is inactive 1 = Interrupt is active. There is one bit of the register for each interrupt source.	0x00000000

VICVECTADDR[0-31]

- 레지스터 : VICVECTADDR[0-31]

- Vector Address Register

- 각 Interrupt Source의 ISR 주소를 저장하는 레지스터
 - 32개의 Interrupt Source에 대응하는 32개의 VICVECTADDR 존재
 - S5PC100은 3개의 VIC가 있으므로 총 96개의 VICVECTADDR 존재

VICVECTADDR[0-31]	Bit	Description	Reset Value
VectorAddr 0-31	[31:0]	Contains ISR vector addresses.	0x00000000

VICADDRESS

- 레지스터 : VICADDRESS

- Vector Address Register

- Interrupt가 발생했을 때 해당 Interrupt source의 ISR 주소를 로드
 - Controller가 자동으로 로드

ex) 12번 interrupt source에서 interrupt 발생

VICADDRESS ← VICVECTADDR[12]

VICADDRESS	Bit	Description	Reset Value
VectAddr	[31:0]	<p>Contains the address of the currently active ISR, with reset value 0x00000000.</p> <p>A read of this register returns the address of the ISR and sets the current interrupt as being serviced. A read must be performed while there is an active interrupt.</p> <p>A write of any value to this register clears the current interrupt. A write must only be performed at the end of an interrupt service routine.</p>	0x00000000

UART INTERRUPT CODE (C CODE)



VPOS_kernel_main()

- 소개

- VPOS 커널 데이터 구조체를 초기화
- 시리얼 장치와 타이머 등 하드웨어를 초기화
- 인터럽트 enable
- 부팅 메시지 출력
- 쉘 스레드 생성
- 스케줄러 호출하는 VPOS_start 루틴으로 진입

- 소스 코드 위치

- vpos/kernel/kernel_start.c

```
void VPOS_kernel_main( void )
{
    pthread_t p_thread, p_thread_0, p_thread_1, p_thread_2;

    /* static and global variable initialization */
    vk_scheduler_unlock();
    init_thread_id();
    init_thread_pointer();
    vh_user_mode = USER_MODE;
    vk_init_kdata_struct();

    vk_machine_init();
    set_interrupt();

    printk("%s\n%s\n%s\n", top_line, version, bottom_line);

    /* initialization for thread */
    race_var = 0;
    pthread_create(&p_thread, NULL, UPOS_SHELL, (void *)NULL);
    //pthread_create(&p_thread_0, NULL, race_ex_1, (void *)NULL);
    //pthread_create(&p_thread_1, NULL, race_ex_0, (void *)NULL);
    //pthread_create(&p_thread_2, NULL, race_ex_2, (void *)NULL);

    UPOS_start();

    /* cannot reach here */
    printk("OS ERROR: UPOS_kernel_main( void )\n");
    while(1){}
}
```



set_interrupt()

- 위치
 - vpos/kernel/kernel_start.c
- 코드 추가
 - vh_serial_irq_enable() 함수 호출

```
void set_interrupt(void)
{
    // interrupt setting
    vh_serial_irq_enable();
}
```

vh_serial_irq_enable()

- 역할

- UART1 Interrupt를 활성화(enable)시키는 함수

- 위치

- vpos/hal/io/serial.c

vh_serial_irq_enable()

- 실행 순서

1. VIC1VECTADDR11 레지스터에 ISR 주소 저장
 - vh_serial_interrupt_handler() 함수 주소 저장
2. VIC1INTENABLE 레지스터에서 UART1 인터럽트를 활성화
 - 11번 비트를 1로 set
3. VIC1INTSELECT 레지스터에서 UART1 인터럽트를 IRQ로 설정
 - 11번 비트를 0으로 clear
4. VIC1SWPRIORITYMASK 레지스터를 모두 mask
 - 모든 비트를 1로 set

vh_serial_irq_enable()

- 코드 추가

```
void vh_serial_irq_enable(void)
{
    /* enable UART1 Interrupt */
    vh_VIC1VECTADDR11 = (unsigned int)&vh_serial_interrupt_handler;
    vh_VIC1INTENABLE |= vh_VIC_UART1_bit;
    vh_VIC1INTSELECT &= ~vh_VIC_UART1_bit;
    vh_VIC1SWPRIORITYMASK = 0xffff;
}
```

```
/* *****
VIC - S5PC100
***** */
#define vh_VIC_UART1    11
#define vh_VIC_UART1_bit (1 << vh_VIC_UART1)

#define vh_VIC_TIMER4    25
#define vh_VIC_TIMER4_bit (1 << vh_VIC_TIMER4)
```

vpos/hal/include/vh_io_hal.h

vh_serial_interrupt_handler()

- 설명
 - UART1 Interrupt Handler
 - 키보드 입력을 받아 버퍼에 저장
- 위치
 - vpos/hal/io/serial.c



vh_serial_interrupt_handler()

- 실행 순서

1. URXH 레지스터에 수신된 키보드 문자 데이터를 버퍼에 저장
 - vk_serial_push() 함수 호출
2. UART1 인터럽트를 pending clear
 - VICINTENCLEAR 레지스터에서 UART1 인터럽트를 비활성화
 - 11번 비트를 1로 set
 - VIC1INTENABLE 레지스터에서 UART1 인터럽트를 활성화
 - 11번 비트를 1로 set
 - UINTP1 레지스터에서 UART1 인터럽트를 다시 설정
 - 모든 비트를 1로 set
 - UINTP1 레지스터를 1로 set해서 인터럽트를 clear



vh_serial_interrupt_handler()

- 코드 추가

```
void vh_serial_interrupt_handler(void)
{
    vk_serial_push(); ←
    vh_VIC1INTENCLEAR |= vh_VIC_UART1_bit;
    vh_VIC1INTENABLE |= vh_VIC_UART1_bit;
    vh_UINTP1 = 0xf;
}
```

URXH 레지스터에 수신된
키보드 문자 데이터를 버퍼에 저장

UART1 인터럽트를 Pending Clear

getc() 수정

- 코드 수정

- vpos/hal/io/serial.c

```
char getc(void)
{
    char c;
    /*
        unsigned long rxstat;

        while(!vh_SERIAL_CHAR_READY());

        c = vh_SERIAL_READ_CHAR();
        rxstat = vh_SERIAL_READ_STATUS();
    */

    while(pop_idx == push_idx){
    }
    c = serial_buff[pop_idx++];

    return c;
}
```

주석 처리

주석 해제

UART INTERRUPT CODE (ASSEMBLY)



Interrupt 진입 루틴

- 루틴 흐름

1. Link register 보정
2. 이전 모드의 레지스터들과 SPSR을 스택에 저장
3. VICIRQSTATUS를 확인하여 3개의 Controller 중 어디에서 인터럽트가 발생하였는지 확인
 - 본 실습에서는 VIC 0과 VIC 1만 사용
4. pc에 VICADDRESS의 값을 저장



Interrupt 복귀 루틴

- 루틴 흐름

1. CPSR을 수정하여 IRQ 모드로 바꾸고 IRQ Mask bit를 1로 set
2. 이전 모드의 레지스터와 SPSR을 스택에서 복원
3. `movs pc, lr` 명령어를 사용하여 원래의 루틴으로 복귀



Interrupt 진입 루틴 : vh_irq

- 코드

- vpos/hal/cpu/HAL_arch_startup.S

vh_irq:

```
sub    lr,lr,#4
str    sp, vk_save_irq_mode_stack_ptr
stmfd  sp,{r14}^
sub    sp, sp, #4
stmfd  sp,{r13}^
sub    sp, sp, #4
stmfd  sp!,{r0-r12}
mrs    r0, spsr_all
stmfd  sp!,{r0, lr}
str    sp, vk_save_irq_current_tcb_bottom
ldr    r0, =0xe4000000
ldr    r1, [r0]
cmp    r1, #0x0
bne    vh_irq_VIC0
ldr    r0, =0xe4100000
ldr    r1, [r0]
cmp    r1, #0x0
bne    vh_irq_VIC1
```

문맥 저장

^ : 이전 모드의 레지스터

SPSR과 Link Register 저장

VIC0IRQSTATUS

VIC1IRQSTATUS

Interrupt 진입 & 복귀 루틴 :

vh_irq_VIC1

- 코드

– vpos/hal/cpu/HAL_arch_startup.S

vh_irq_VIC1:

ldr r0, =0xe4100f00

ldr r1, [r0]

mov r14, pc

mov pc, r1

VIC0ADDRESS에 저장된 인터럽트 핸들러 주소로 점프

msr cpsr_c, #vh_IRQMODE | 0x80

ldr r14, =0xe4100f00

str r14, [r14]

VIC 인터럽트 서비스 완료

ldmfd sp!, {r0, lr}

msr spsr_cxsf, r0

ldmfd sp!, {r0-r12}

ldmfd sp, {r13}^

add sp, sp, #4

ldmfd sp, {r14}^

add sp, sp, #4

movs pc, lr

문맥 복원



HOMEWORK



실습 및 과제

- **UART1 Interrupt 관련 코드 추가**
 - vpos/hal/io/serial.c
 - UART1 Interrupt Enable
 - vh_serial_irq_enable()
 - UART1 Interrupt Handler
 - vh_serial_interrupt_handler()
 - getc() 함수 주석 처리 수정
 - vpos/hal/cpu/HAL_arch_startup.S
 - vh_irq와 vh_irq_VIC1



실습 및 과제

- VIC 관련 레지스터 주소 정의
 - vpos/hal/include/vh_io_hal.h
 - VIC1INTSELECT
 - VIC1INTENABLE
 - VIC1INTENCLEAR
 - VIC1SWPRIORITYMASK
 - VIC1VECTADDR11



보고서 제출

- 보고서

- 학과, 학번, 이름
- 코드를 캡처해서 보고서에 첨부
 - serial.c에서 수정한 부분 (함수 3개)
 - vh_serial_irq_enable()
 - vh_serial_interrupt_handler()
 - getc()
 - HAL_arch_startup.S에서 수정한 부분(레이블 2개)
 - vh_irq
 - vh_irq_VIC1
 - vh_io_hal.h의 VIC 관련 레지스터 주소 정의 부분



SOFTWARE INTERRUPT



VPOS 커널을 포팅하기 위한 준비

1. 커널 컴파일 + 커널 이미지를 RAM에 적재
2. Startup code 작성
3. UART 설정
4. TIMER 설정
5. Hardware Interrupt Handler 구현
(1) UART Interrupt
6. Software Interrupt Entering/Leaving Routine 구현
7. Timer Interrupt



목차

1. Software Interrupt
2. Scheduler of VPOS
3. SWI Entering Routine & Leaving Routine
4. Homework



SOFTWARE INTERRUPT



Software Interrupt

- 정의

- 프로그램의 실행을 **그 프로그램에 포함시킨 명령어로** 중단시키고, 다른 프로그램의 실행으로 제어를 옮기는 것
- 하드웨어적으로 인터럽트를 거는 방식이 아님
- SWI 명령어를 사용해 인터럽트를 발생시킴

- 목적

- 커널 함수를 호출하기 위해 특권모드에 진입하고자 할 때 사용
 - User Mode → Privileged Mode(Supervisor Mode)

Hardware Interrupt vs. Software Interrupt

- 비교

	하드웨어 인터럽트(IRQ/FIQ)	소프트웨어 인터럽트(SWI)
목적	일반 인터럽트 처리	운영체제 보호모드
벡터 테이블 오프셋	0x18	0x08
익셉션 우선순위	4	6
인터럽트 활성화 여부	비활성화	비활성화 하지 않음
링크 레지스터 보정	$lr = lr - 4$	불필요

Hardware Interrupt vs. Software Interrupt

- 링크 레지스터 보정

- 하드웨어 인터럽트

- 현재 실행 중인 명령어를 실행한 뒤 인터럽트를 처리
 - 다음에 실행할 명령어는 $pc - 4$ 에 위치
 - $lr = lr - 4$ 를 통해 링크 레지스터의 값을 수정해야 함

- 소프트웨어 인터럽트

- 파이프라인의 3단계 중 decode 단계에서 swi 명령어를 인식하고 소프트웨어 인터럽트를 발생시킴
 - PC가 가리키는 명령어는 swi 다음 명령어
 - 링크 레지스터 값을 수정할 필요 없음



SWI

- **SWI 명령어**

- 소프트웨어 인터럽트를 발생시키는 명령어
- 프로세스 모드를 Supervisor mode로 변경시켜 운영체제 루틴이 특권 모드에서 호출될 수 있도록 해줌
- 표기법 : SWI {<조건>} SWI_number
 - SWI_number : 특별한 함수 호출이나 특징을 나타내는 데 사용

SCHEDULER OF VPOS



VPOS에서의 스케줄러

- **vk_scheduler()**
 - VPOS의 스케줄러 함수
 - 32단계의 정적 우선순위 Ready 큐 구조
 - 같은 우선순위 내에서는 Round-Robin 스케줄링 방식
 - 타이머 인터럽트로 일정 주기마다 vk_scheduler()를 호출하여 스레드는 정해진 time slice동안만 실행
 - 커널 함수
 - User Mode가 아닌 Privileged Mode나 System Mode에서 호출해야 함
 - 커널 함수를 사용하기 위해서는 User Mode에서 Privileged Mode로 전환해야 함
- ➔ **Software Interrupt 사용**



스케줄러 호출 순서 (VPOS)

1. 현재 실행 중인 스레드의 구조체 변수 중 'swi_number' 변수에 'CS' 저장
2. "swi 0x00" 명령어로 소프트웨어 인터럽트를 발생시킴
3. SWI 진입 루틴에서 vk_swi_classifier() 함수로 점프
4. 'swi_number' 변수 값을 확인하여 커널 함수 호출
 - CS인 경우 vk_scheduler() 호출
5. 스케줄링 시작



vk_swi_scheduler() & vh_swi()

- 현재 스레드의 구조체 변수 중 'swi_number' 변수에 'CS' 저장
 1. VPOS_start() 실행
 2. vk_swi_scheduler() 실행
 3. 현재 실행중인 스레드의 구조체 변수 중 'swi_number' 변수에 'CS' 저장
 4. vh_swi() 실행
- “swi 0x00” 명령어로 소프트웨어 인터럽트를 발생시킴



vk_swi_scheduler() & vh_swi()

- 현재 스레드의 구조체 변수 중 'swi_number' 변수에 'CS' 저장

```
//pthread_create(&p_thread_1, NULL, race_ex_0,  
//pthread_create(&p_thread_2, NULL, race_ex_2,  
  
UPOS_start();  
  
/* cannot reach here */  
printf("OS ERROR: UPOS_kernel_main( void )\n");  
while(1){}
```

```
void UPOS_start(void)  
{  
    vk_swi_scheduler();  
}
```

```
void vk_swi_scheduler(void)  
{  
    unsigned temp;  
  
    vk_current_thread->swi_number = CS;  
    temp = (unsigned)vk_current_thread;  
    vh_swi(temp);  
}
```

```
void vh_swi(unsigned thread)  
{  
    asm volatile("swi 0x00");  
}
```

vk_swi_classifier()

- SWI 진입 루틴을 통해 vk_swi_classifier() 함수로 점프

SWI 벡터 엔트리

```
vh_software_interrupt:  
vh_entering_swi:
```

```
b1      vk_swi_classifier
```

```
vh_leaving_swi:
```



```
void vk_swi_classifier(unsigned thread)  
{  
    unsigned number;  
    vk_thread_t *vector;  
    unsigned temp;  
    int i;  
    unsigned int *k=vk_save_swi_mode_stack_ptr;  
    unsigned int *kk=vk_save_swi_current_tcb_bottom;  
    printk("vk_swi_classifier switch up\n");  
    vector=(vk_thread_t *)thread;  
    number=vector->swi_number;  
    switch(number)  
    {  
        case EI:  
            vector->interrupt_state = FALSE;  
            vh_enable_interrupt(vector);  
            break;  
        case DI:  
            vector->interrupt_state = TRUE;  
            vh_disable_interrupt(vector);  
            break;  
        case SC:  
            vh_save_thread_ctx((unsigned)vector->tcb_bottom);  
            break;  
        case RC:  
            temp = (unsigned)vector->func;  
            vh_restore_thread_ctx((unsigned)vector->tcb_bottom);  
            break;  
        case CS:  
            vk_scheduler();  
            break;  
    }  
}
```

APCS

r0 – r3 and **Stack**

```
int add(int a1, int a2, int a3, int a5, int a6, int a7)
{
    int v1, v2, v3, v4, v5, v6;
    int sum;

    v1 = a1;
    v2 = a2;
    ...

    sum = v1 + v2 + ... + v6;
    return sum;
}
```

r4 – r10 and **Stack**

r0

SWI ENTERING ROUTINE & LEAVING ROUTINE



Software Interrupt 진입 루틴

- 루틴 흐름

1. 이전 모드의 레지스터들과 SPSR, lr을 스택에 저장
2. r0~r3 레지스터에 커널 함수가 사용할 매개 변수 저장(APCS)
 - 함수를 호출하면서 r0~r3 레지스터에 자동으로 매개 변수 저장
 - 진입 루틴에서 r0~r3 레지스터를 수정하였다면 커널 함수를 실행하기 전에 다시 원래 값으로 되돌려야 함
3. SWI 핸들러로 점프
 - vk_swi_classifier()로 점프



Software Interrupt 복귀 루틴

- 루틴 흐름

1. 이전 모드의 레지스터와 SPSR을 스택에서 복원
2. `movs pc, lr` 명령어를 사용하여 원래의 루틴으로 복귀



HOMEWORK



VPOS_kernel_main()

• 소개

- VPOS 커널 데이터 구조체를 초기화
- 시리얼 장치와 타이머 등 하드웨어를 초기화
- 인터럽트 Enable
- 부팅 메시지 출력
- 쉘 스레드 생성
- 스케줄러 호출하는 VPOS_start 루틴으로 진입

• 소스 코드 위치

- vpos/kernel/kernel_start.c

```
void VPOS_kernel_main( void )
{
    pthread_t p_thread, p_thread_0, p_thread_1, p_thread_2;

    /* static and global variable initialization */
    vk_scheduler_unlock();
    init_thread_id();
    init_thread_pointer();
    vh_user_mode = USER_MODE;
    vk_init_kdata_struct();

    vk_machine_init();
    set_interrupt();

    printk("%s\n%s\n%s\n", top_line, version, bottom_line);

    /* initialization for thread */
    race_var = 0;
    pthread_create(&p_thread, NULL, UPOS_SHELL, (void *)NULL);
    pthread_create(&p_thread_0, NULL, race_ex_1, (void *)NULL);
    pthread_create(&p_thread_1, NULL, race_ex_0, (void *)NULL);
    pthread_create(&p_thread_2, NULL, race_ex_2, (void *)NULL);

    UPOS_start();

    /* cannot reach here */
    printk("OS ERROR: VPOS_kernel_main( void )\n");
    while(1){}
```

주석 처리 해제

실습 및 과제

- SWI 관련 코드 추가
 - vpos/hal/cpu/HAL_arch_startup.S
 - SWI Entering Routine
 - vh_entering_swi (레이블)
 - SWI Leaving Routine
 - vh_leaving_swi (레이블)

```
vh_software_interrupt:  
vh_entering_swi:  
  
        bl        vk_swi_classifier  
  
vh_leaving_swi:
```



vh_entering_swi

- SWI 진입 루틴 구현 1

1. 'vk_save_swi_mode_stack_ptr' 변수에 현재 sp 값을 저장 (str)
2. 이전 모드의 pc, lr 레지스터와 범용 레지스터들을 스택에 저장
 - 이전 모드 : ^ 사용
3. SPSR과 lr을 스택에 저장
 - mrs 명령어로 SPSR을 r0에 저장한 뒤 r0를 스택에 저장
4. IRQ 익셉션을 비활성화
 - r0 레지스터에 cpsr 값을 저장하고 인터럽트 마스크 비트 수정
5. 'vk_save_swi_current_tcb_bottom' 변수에 현재 sp 값을 저장

vh_entering_swi

- SWI 진입 루틴 구현 2

6. r0 레지스터에 커널함수가 사용할 매개 변수를 다시 저장
 - ldr 명령어를 사용. sp에 오프셋을 더해 2.에서 저장한 r0값을 스택에서 로드
 - ldr r0, [sp, #offset]
7. SWI 핸들러로 점프
 - “bl vk_swi_classifier”



vh_leaving_swi

- **SWI 복귀 루틴 구현**

1. 스택에 저장했던 모든 레지스터들을 복원
 - msr 명령어로 SPSR에 저장해야 함
2. `movs pc, lr` 명령어를 사용하여 원래의 루틴으로 복귀



힌트

- **Entering_SWI 구현**

- Hardware Interrupt랑 매우 유사함
- 6. 과정만 추가됨(p.72)
- 6. 과정을 수행하는 이유
 - 3. 4.에서 r0에 사용했기 때문에 다시 복구해야함
- 강의에서 배웠던 스택의 지식을 활용할 것

- **Leaving_SWI 구현**

- Hardware Interrupt 랑 똑같다



보고서 제출

- 보고서

- 학과, 학번, 이름
- 코드를 캡처해서 보고서에 첨부
 - HAL_arch_startup.S에서 추가한 부분(레이블 2개)
 - vh_entering_swi
 - vh_leaving_swi
- kernel_start.c의 주석을 수정한 후 화면 캡처
 - 익셉션이 발생하지 않고 쉘까지 출력되어야 함
 - 쉘에 'ls' 입력 후 캡처



보고서 제출

```
Starting kernel ...
```

```
*****  
*  QURIX version 3.0    xx/10/2012    *  
*****
```

```
timbuffer[0] - 16113  
timbuffer[1] - 16082  
timbuffer[2] - 16051  
timbuffer[3] - 16020  
timbuffer[4] - 15989  
timbuffer[5] - 15957  
timbuffer[6] - 15926  
timbuffer[7] - 15895  
timbuffer[8] - 15863  
timbuffer[9] - 15832
```

```
vk_swi_classifier switch up  
vk_swi_classifier switch up  
vk_swi_classifier switch up  
vk_swi_classifier switch up
```

```
Race condition value = 1200000
```

```
Shell>ls
```

```
vk_swi_classifier switch up
```

```
*****Command_List*****
```

```
- help  
- ls  
- debug  
- thread  
- temp
```

```
*****  
vk_swi_classifier switch up
```

```
Shell>
```



제출 방법

- 제출 방법
 - 워드나 한글로 작성하여 블랙보드에 제출
 - 문서 제목에 학번과 이름을 적을 것
- 마감일
 - 다음 실습 수업시간 전까지 (5/24)



수고하셨습니다.

