

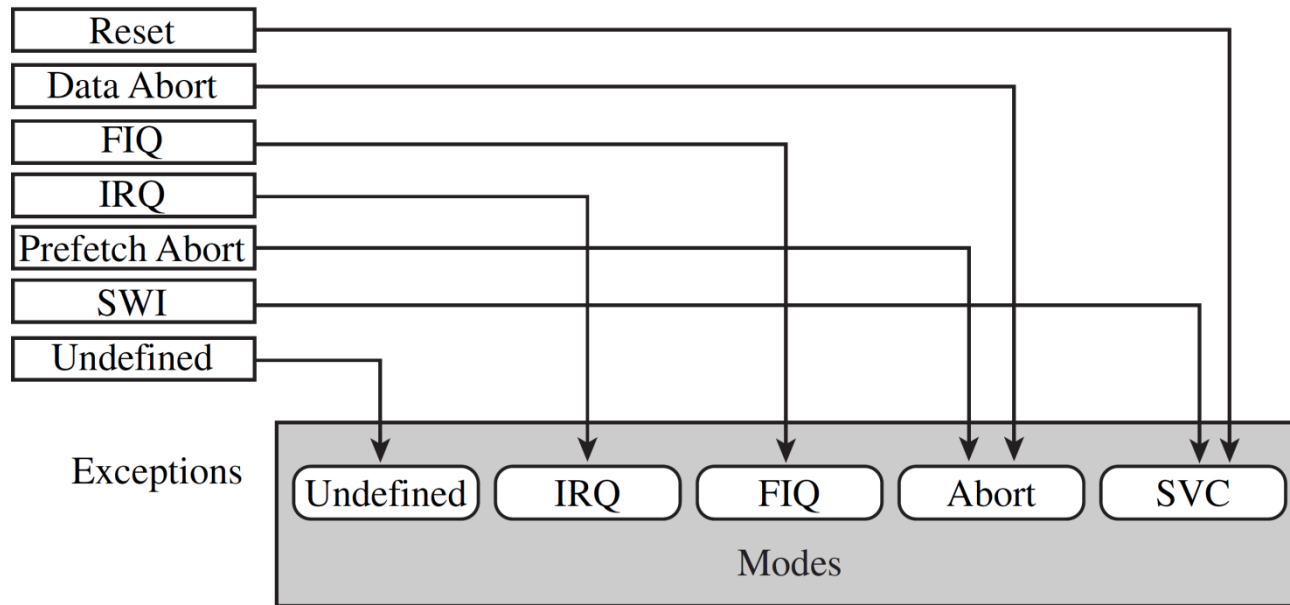
6 Exception Programming

내용

- Exceptions and Modes
- Register Organization
- Program Status Registers
- Exception Processing
- Exception Stacks
- Exception Vector
- 실습 프로그램: start.S, exception.c, handler.c

Exceptions and Modes

- ARM은 7가지 종류의 exception이 있고 이들 exception이 발생하면 processor는 해당되는 5개의 mode로 들어간다



- ARM은 위 5개 processor mode외에 user mode 및 system mode가 있다 (주: ARM은 총 7개의 processor mode가 있음)

Exceptions

- ① Reset exception: reset input에 의하여 발생하는 exception임. return에 대한 정의는 없음
- ② Undefined exception: 정의되지 않거나 허용되지 않은 instruction을 수행하면 발생하는 exception임
- ③ Software interrupt exception: 명령어 SWI 수행 시 발생하는 exception임. OS의 system call 구현에 사용
- ④ Prefetch abort exception: 명령어 fetch 시 해당 메모리 접근에 문제가 있으면 발생하는 exception임. 명령어 BKPT 수행 시에도 발생함
- ⑤ Data abort exception: load/store 명령어 수행 시 해당 메모리 접근에 문제가 있으면 발생하는 exception임
- ⑥ IRQ: processor의 IRQ input을 assert하면 발생하는 exception임
- ⑦ FIQ: processor의 FIQ input을 assert하면 발생하는 exception임

Processor Modes


- ① User(usr): 일반 프로그램을 수행하는 모드
- ② FIQ(fiq): 고속 data transfer 혹은 channel process 용 모드
- ③ IRQ(irq): 일반 목적의 interrupt 처리용 모드
- ④ Supervisor(svc): OS 커널 수행을 위한 모드
- ⑤ Abort(abt): virtual memory 혹은 memory protection을 구현하기 위한 모드
- ⑥ Undefined(und): 정의되지 않은 명령어를 emulation하는 경우 사용되는 모드
- ⑦ system(sys): OS가 실행하는 task의 수행 모드

Processor Mode의 분류

- User mode: 응용 프로그램이 수행되는 mode로 protected system resource를 access할 수 없다
- Privileged mode: User mode가 아닌 나머지 6개 mode를 privileged mode라고 하는데 이들 mode에서는 자유롭게 system resource를 access할 수 있다
- Exception mode: Privileged mode 중 5개인 FIQ, IRQ, Supervisor, Abort, Undefined는 exception 발생 시 들어가는 exception mode라고 한다 (주: 각 exception mode는 몇개의 추가적인 register를 가지고 있다)
- System mode: privilege를 가지는 특별한 user mode로서 user mode에서 사용되는 같은 register를 사용한다

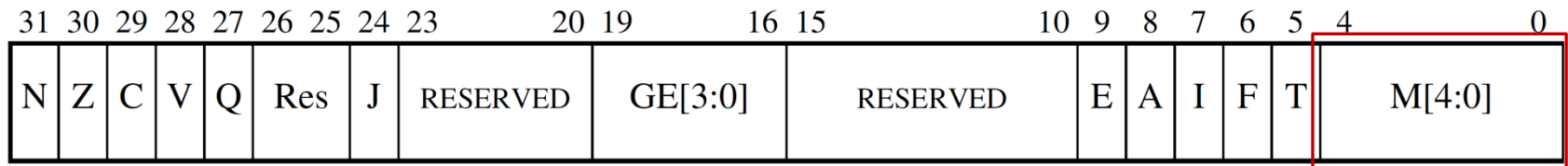
Register Organization

Modes							
Privileged modes							
Exception modes							
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt	
R0	R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14	R14_fiq
PC	PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq		SPSR_fiq

 indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

Program Status Registers

- CPSR(Current Program Status Register)는 모든 processor mode에서 같은 register를 사용한다 (1개)
- 각 exception mode마다 exception 발생 시 CPSR을 저장하는 SPSR(Saved Program Status Register)가 하나씩 있다 (5개)
- CPSR에서 processor mode는 M[4:0]이 표시함



Program status register 모습

CPSR의 Mode Bits

M[4:0]	Mode
0b10000	User
0b10001	FIQ
0b10010	IRQ
0b10011	Supervisor
0b10111	Abort
0b11011	Undefined
0b11111	System

Exception Processing

- Exception이 발생하면 ARM core는 아래를 실행한다
 - ① $R14_{\langle exception_mode \rangle} = \text{return address (exception마다 다름)}$
 - ② $SPSR_{\langle exception_mode \rangle} = CPSR$
 - ③ $CPSR[4:0] = \text{exception mode number}$
 - ④ 각 exception에 따라 IRQ 혹은 FIQ를 disable
 - ⑤ $PC = \text{exception vector address (아래 표)}$

Exceptions	Processor modes	Exception vector addresses
Reset	Supervisor	0x00000000
Undefined instruction	Undefined	0x00000004
Software interrupt (SWI)	Supervisor	0x00000008
Prefetch abort	Abort	0x0000000c
Data abort	Abort	0x00000010
-	-	0x00000014
Interrupt request (IRQ)	Interrupt request (IRQ)	0x00000018
Fast Interrupt request (FIQ)	Fast interrupt request (FIQ)	0x0000001c

Return Addresses

- 각 exception 마다 R14_<exception_mode>에 저장되는 return address는 아래와 같다

Exceptions	R14_<exception_mode> 저장되는 address
Reset	정의되지 않음
Undefined instruction	Undefined instruction의 주소 + 4
Software interrupt (SWI)	SWI instruction의 주소 + 4
Prefetch abort	Aborted된 instruction의 주소 + 4
Data abort	Aborted된 instruction의 주소 + 8
Interrupt request (IRQ)	다음 수행할 instruction의 주소 + 4
Fast Interrupt request (FIQ)	다음 수행할 instruction의 주소 + 4

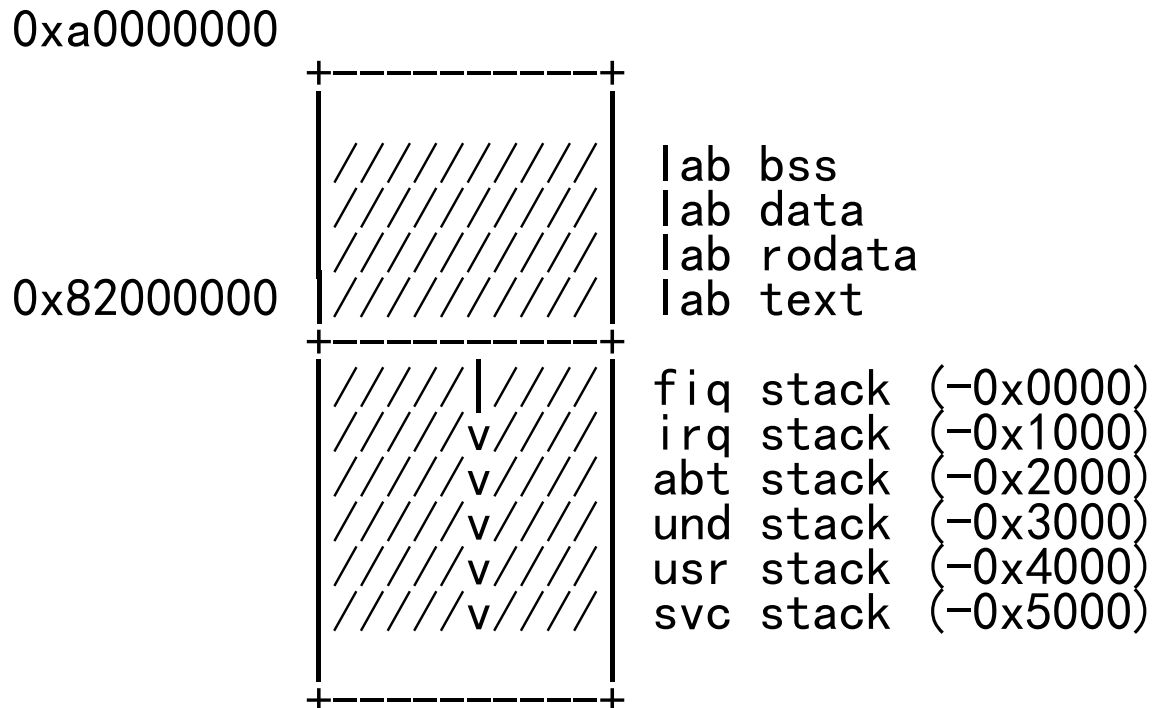
Interrupt Disable

- Exception이 발생하면 아래 표와 같이 exception에 따라 CPSR의 I bit 혹은 F bit이 set되어 IRQ 혹은 FIQ가 disable된다
- 여러 exception이 동시에 발생되면 아래 표의 exception priority에 따라 최종 exception이 결정된다

Exceptions	Priority	<i>I</i> bit	<i>F</i> bit
Reset	1	1	1
Data Abort	2	1	—
Fast Interrupt Request	3	1	1
Interrupt Request	4	1	—
Prefetch Abort	5	1	—
Software Interrupt	6	1	—
Undefined Instruction	6	1	—

Exception Stacks

- 각 exception mode는 각각 독자적인 stack pointer (=R13_<exception_mode>)를 가지고 있다. 각 mode의 stack의 위치 및 크기는 잘 design되어야 한다 (아래는 BeagleBone에서의 보기)



Exception Stack Setup

- Supervisor mode stack: ARM은 reset 후 supervisor mode로 시작되기 때문에 초기에 $sp(=r13)$ 을 설정하면 된다
- 타 mode stack: CPSR의 M[4:0]를 수정하여 processor mode를 원하는 mode로 변경한 후 $sp(=r13)$ 을 설정한다

```
ldr sp, _svc_stack
```

```
...
```

```
_svc_stack:
```

```
.word 0x82000000-0x5000
```

```
mrs r4, cpsr
```

```
bic r5, r4, #0x1f
```

```
orr r5, r5, #0x1b
```

```
msr cpsr, r5
```

```
ldr sp, _undefined_stack
```

```
...
```

```
_undefined_stack:
```

```
.word 0x82000000-0x3000
```

Exception Vector

- 일반적으로 exception vector에는 아래와 같은 instruction을 가지고 있어서 exception이 발생되면 해당 exception에 해당하는 프로그램으로 jump한다

b address

```
ldr pc, saved_address
```

```
mov pc, #immediate
```

Exception Vector 보기

```
.align 5
```

```
new_vector_base:
```

```
    ldr pc, _reset
```

```
    ldr pc, _undefined_instruction
```

```
    ldr pc, _software_interrupt
```

```
    ldr pc, _prefetch_abort
```

```
    ldr pc, _data_abort
```

```
    ldr pc, _not_used
```

```
    ldr pc, _irq
```

```
    ldr pc, _fiq
```

```
_reset:
```

```
    .word reset
```

```
_undefined_instruction:
```

```
    .word undefined_instruction
```

```
_software_interrupt:
```

```
    .word software_interrupt
```

```
_prefetch_abort:
```

```
    .word prefetch_abort
```

```
_data_abort:
```

```
    .word data_abort
```

```
_not_used:
```

```
    .word _not_used
```

```
_irq:
```

```
    .word irq
```

```
_fiq:
```

```
    .word fiq
```


Vector Base Address Register

- ARMv7는 exception vector의 시작 주소를 정의하는 vector base address register(VBAR)가 coprocessor 15 register 12에 있다
- 이 register를 read/write하는 명령어는 아래와 같다

```
mrc p15, #0, <Rt>, c12, c0, #0 // read
```

```
mcr p15, #0, <Rt>, c12, c0, #0 // write
```

참고: 이 register는 bit[4:0]이 0이기 때문에 vector base address는 2^5 으로 align되어야 한다 (.align 5)

실습 프로그램

- start.S: 각종 초기화 후 C 언어 함수 main()을 call하는 부분과 exception vector를 통하여 C 언어로 작성된 exception handler 함수들을 call하는 부분으로 구성된다
- exception.c: 다양한 exception을 시험하기 위하여 의도적으로 exception을 발생시키는 함수 main()으로 구성된다
- handler.c: C 언어로 작성된 exception handler 함수들로 구성된다

start.S - 함수 main() 부름

- ① 레지스터 lr을 u-boot의 stack(=sp_svc임)에 push한다
- ② u-boot의 sp를 변수 _saved_sp에 저장한다
- ③ 현재 VBAR 값을 변수 _saved_vector_base_register에 저장하고 새로운 vector 주소를 VBAR에 저장한다
- ④ 각 모드(fiq, irq, abt, und, usr, svc)의 stack pointer를 (0x82000000-0x?)로 설정한다
- ⑤ bss section을 0으로 초기화한다
- ⑥ C 언어 함수 main()을 call한다
- ⑦ 변수 _saved_vector_base_register로부터 VBAR을 복구한다
- ⑧ 변수 _saved_sp로부터 sp를 복구한다
- ⑨ pc를 u-boot의 stack에서 pop하여 얻은 값으로 수정한다

start.S - Exception Handler 부름

```
.align 5

new_vector_base:                // exception vector
    ldr pc, _reset
    ldr pc, _undefined_instruction
...
_undefined_instruction:
    .word undefined_instruction
...

undefined_instruction:
    stmfd sp!, {r0-r12, lr}      // r0-r12, lr을 stack에 저장
    mov r0, sp                  // r0=sp (첫째 인수 unsigned int *sp)
    mrs r1, spsr                // r1=spsr (둘째 인수 unsigned int spsr)
    bl handle_undefined_instruction // call C handler
    ldmdf sp!, {r0-r12, pc}^     // r0-r12, pc, cpsr를 복구
...
```

exception.c - 함수 main()

```
int main(int argc, char *argv[])
{
    char input[80];

    for (;;) {
        UART_printf("=====\n");
        UART_printf("Exceptions (%s mode)\n", get_current_mode());
        UART_printf("=====\n");
        UART_printf("1. Undefined instruction\n");
        UART_printf("2. Software interrupt\n");
        UART_printf("3. Prefetch abort\n");
        UART_printf("4. Data abort\n");
        UART_printf("5. Quit\n");
        UART_printf("Select one: ");
        ...
    }
}
```

handler.c - Exception Handlers

```
void handle_undefined_instruction(unsigned int *sp,  
    unsigned int spsr)  
{  
    unsigned int pc, instruction;  
  
    ...  
    UART_printf("Undefined instruction: mode=%s,  
pc=%08x, *pc=%08x\n", get_current_mode(), pc,  
instruction);  
    ...  
}  
  
...
```

참고 문헌

1. ARM Limited, ARM[®] Architecture Reference Manual, ARM DDI 0100I, July 2005. (Chapter A2.6)
2. ARM Limited, ARM[®] Architecture Reference Manual ARM[®]v7-A and ARM[®]v7-R Edition, ARM DDI 0406B, July 2009. (Chapter B1.6)