ARM Architecture

교 육 목 표

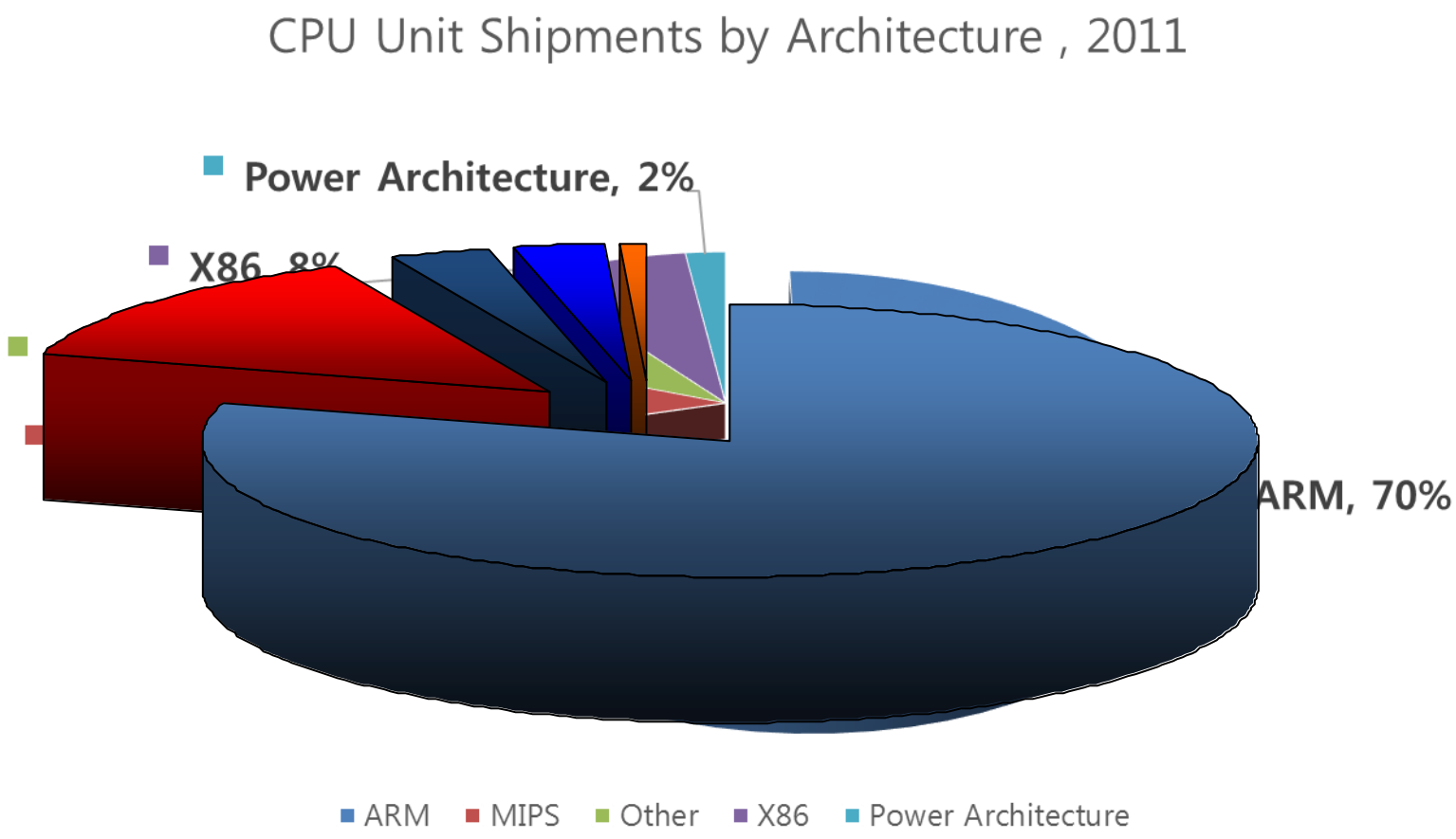
- ARM Architecture에 대하여 이해한다.
- ARM의 Programmer's Model에 대하여 이해한다.
- ARM 명령어의 특징에 대하여 이해한다.
- ARM의 동작 (Operating) 모드에 대하여 이해한다.
- ARM의 레지스터에 대하여 이해한다.
- ARM의 Exception 핸들링 방법을 이해한다.
- 하드웨어적으로 Reset이 발생했을 때 ARM의 동작을 이해한다.

목 차

ARM 소개

- 1990년 설립
  - Acorn Computer에서 독립
  - 12 engineers & CEO
  - Cambridge, UK
- 32-bit RISC Intellectual Property 제공
  - ARM은 silicon을 제조 판매하지는 않는다.
- 1998년 NASDAQ (ARMHY) 과 LSE등록
- 현재 70개 이상의 semiconductor 파트너 보유

ARM의 시장 점유율



Source: IDC, SMART Technology Conference

ARM의 32비트 RISC 프로세서 IP

❖ ARM사는 직접 반도체를 제조하여 판매하는 것이 아니라 설계한 프로세서를 반도체 회사에 Hard Macrocell 또는 Synthesizable core로 제공

❖ 반도체 제조회사 또는 SoC 제조사에서는 ARM사로부터 제공받은 ARM core와 주변장치를 추가하여 SoC를 만들어 사용자에게 판매하거나 자체 제품에서 사용

Hard Macrocell 과 Synthesizable Core

❖ Hard Macrocell

- Layout level로 제공
- 사용자는 core를 변경 불가
- ARM710T, ARM920T 등이 대표적인 예이다.

❖ Synthesizable Core

- RTL(Register Transfer Level)로 제공
- 사용자는 license 조건에 의해 core의 내부 변경 가능
  - 내부 메모리 크기 등
  - 기능은 변경 불가
- ARM7TDMI-S, ARM926EJ-S, ARM1136J-S 등

목 차

❖ ARM Architecture

- ARM core의 기본 구조를 일컫는 말로, 처리되는 데이터의 사이즈, 명령어의 구조, 레지스터 등과 같은 추상적인 구성 및 동작 원리

❖ ARM Architecture의 종류

- v4, v4T, v5TE, v5TEJ, v6, v6T2, v6kz, v6k, v7-A, v7-R, V6-M, v7-M, v7E-M , v8-A등

ARM core 또는 ARM Processor core

❖ ARM core 또는 ARM processor core

- ARM Architecture의 기본 원리를 이용하여 구현한 프로세서의 핵심 부분

❖ ARM processor core의 종류

- ARM7TDMI processor core
- ARM9TDMI processor core
- ARM9E processor core
- ARM10E processor core
- ARM11 processor core
- Cortex processor core

ARM Processor

★

❖ ARM 프로세서(Processor)

- 프로세서 core에 Cache, MMU(Memory Management Unit), Write Buffer, TCM(Tightly Coupled Memory), BIU(Bus Interface Unit)과 같은 주변 회로를 구성한 독립된 형체

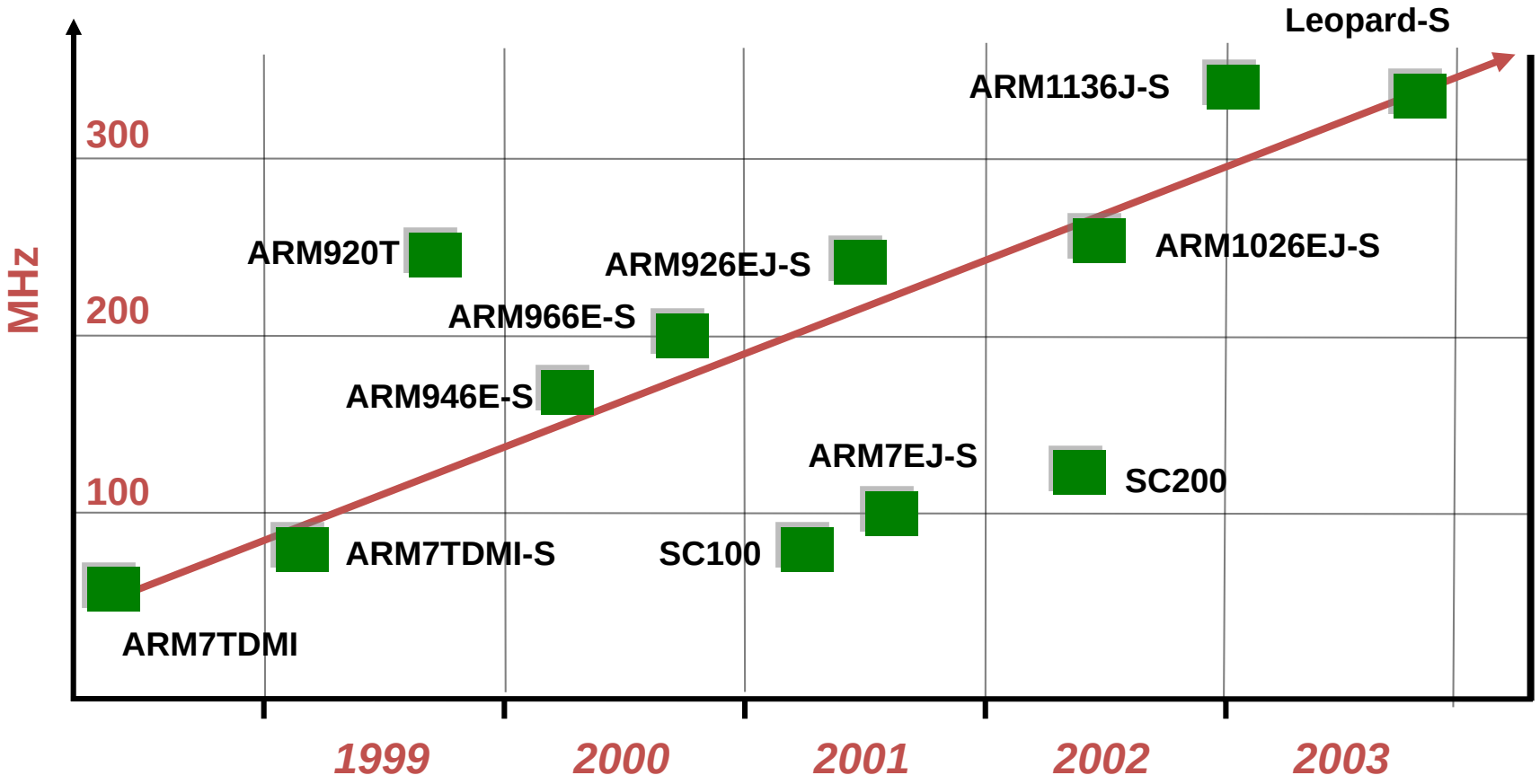
❖ ARM 프로세서의 종류

- ARM920T 프로세서, ARM926EJ프로세서, ARM940T 프로세서, ARM946EJ 프로세서, ARM1020E 프로세서, ARM1136JF-S 프로세서 등

❖ 프로세서 Family

- 동일한 프로세서를 사용한 제품군
- ARM9TDMI core를 사용한 ARM940T나 ARM920T와 같은 프로세서를 ARM9TDMI 프로세서 family라고 부른다.

ARM 프로세서 로드 맵


Leopard-S
ARM1136J-S
300
ARM920T
ARM926EJ-S
ARM1026EJ-S
200
ARM966E-S
ARM946E-S
ARM7EJ-S
SC200
100
ARM7TDMI-S
SC100
ARM7TDMI
MHz
1999
2000
2001
2002
2003

목 차

ARM Architecture의 특징 (1)

- 32비트 RISC Architecture
  - 명령어와 내부 레지스터가 32비트로 구성
  - RISC(Reduced Instruction Set Computer)
    - CISC에 비해 명령어 구조가 간단하고 명령어 수가 적어 보다 빠르고 효율적인 처리가 가능
- 다양한 명령어 지원
  - 32비트 ARM 명령과 16비트 Thumb 명령을 제공
  - 16비트 Thumb 명령
    - 32비트 ARM 명령에 기초하여 16비트 길이로 opcode를 구성
    - 코드 크기를 줄이고, 8 또는 16비트 메모리 인터페이스에서 효과적으로 명령어 처리
  - Jazelle 코어를 확장한 경우 8비트의 Java byte 코드도 실행 가능

ARM Architecture의 특징 (2)

❖ Big/Little Endian 지원

- Endian 방식
  - Endian은 메모리에 여러 바이트의 내용을 저장할 때 상위 바이트 및 하위 바이트의 위치를 결정하는 방식을 말한다.
  - Little-Endian
    - 인텔 계열의 CPU와 같이 하위 바이트를 메모리의 하위 어드레스에 위치하는 방식
  - Big-Endian
    - 모토로라 계열의 CPU와 같이 상위 바이트의 메모리를 하위 어드레스에 놓는 방식
- ARM은 Big-Endian과 Little-Endian을 모두 지원한다.

ARM Architecture의 특징 (3)

❖ Fast Interrupt 지원

- 빠른 인터럽트 처리를 위해 별도의 FAST 인터럽트 방식 제공
  - ARM은 FAST 인터럽트를 위한 별도의 레지스터를 가지고 있어 인터럽트 서비스 루틴 (Interrupt Service Routine)을 작성할 때 레지스터를 저장하고 복구하는 시간을 줄일 수 있다.

목 차

Programmer's Model

❖ Programmer's Model
- 프로그래머가 프로그램을 작성하는데 필요한 각종 정보
  - 여기서 프로그램은 C나 C++이 아닌 어셈블리어를 의미
- Programmer's Model은 ARM의 Architecture를 구분 하는 기준

❖ 프로그래머가 알아야 할 정보 들
- 명령어
- 메모리 구조
- 데이터 구조
- 프로세서의 동작 모드
- 프로세서 내부 레지스터의 구성 및 사용법
- Exception 처리
- 인터럽트 처리

목 차

명령어(Instruction Set)

❖ ARM 프로세서의 명령어

- 32 비트 ARM instruction set
- 16 비트 Thumb instruction set

❖ Jazelle core를 확장한 프로세서

- 8 비트 Java 바이트 명령어

32 비트 ARM 명령어

- 32 비트 ARM 명령어의 특징
  - 모든 ARM 명령어는 조건부 실행이 가능하다.
  - 모든 ARM 명령은 32 비트로 구성되어 있다.
    - Load/Store와 같은 메모리 참조 명령이나 Branch 명령에서는 모두 상대주소(Indirect Address)방식을 사용한다.
    - Immediate 상수는 32 비트 명령어 내에 표시된다.
      - Immediate 상수는 32비트 이하의 상수이다.
  - ARM 명령은 크게 11개의 기본적인 Type으로 구분된다.
- 32 비트의 고정된 명령 길이를 사용하는 이유
  - Pipeline 구성이 용이
  - 명령 디코더의 구현이 쉽다.
  - 고속으로 처리 가능

ARM 명령어 요약

mds ACADEMY

	Instruction Type	Instruction (명령)
1	Branch, Branch with Link	B, BL
2	Data Processing 명령	ADD, ADC, SUB, SBC, RSB, RSC, AND, ORR, BIC, MOV, MVN, CMP, CMN, TST, TEQ
3	Multiply 명령	MUL, MLA, SMULL, SMLAL, SMULL, UMLAL
4	Load/Store 명령	LDR, LDRB, LDRBT, LDRH, LDRSB, LDRSH, LDRT, STR, STRB, STRBT, STRH, STRT
5	Load/Store Multiple 명령	LDM, STM
6	Swap 명령	SWP, SWPB
7	Software Interrupt(SWI) 명령	SWI
8	PSR Transfer 명령	MRS, MSR
9	Coprocessor 명령	MRC, MCR, LDC, STC
10	Branch Exchange 명령	BX
11	Undefined 명령	

16 비트 Thumb 명령어

- Thumb 명령어
  - 32비트의 ARM 명령을 16비트로 재구성한 명령
- Thumb 명령어의 장점
  - 코드의 크기를 줄일 수 있다.
    - ARM 명령의 65%
  - 8비트 나 16비트와 같은 좁은 메모리 인터페이스에서 ARM 명령을 수행할 때 보다 성능이 우수하다.
- Thumb 명령어의 단점
  - 조건부 실행이 안된다.
  - Immediate 상수 값이 표현 범위가 적다.

ARM / Thumb Interwork

❖ ARM state와 Thumb state

- ARM state
  - 32 비트 ARM 명령을 수행하는 상태
  - Reset 및 SWI, IRQ, FIQ, UNDEF, ABORT와 같은 Exception 발생시 프로세서는 무조건 ARM state가 된다.
- Thumb state
  - 16 비트 Thumb 명령을 수행하는 상태

❖ ARM / Thumb Interwork

- ARM state에서 Thumb state로 Thumb state에서 ARM state로 상태가 전환되는 작업
- BX 명령이 의해서 이루어 진다.

Java 명령

❖ Jazelle core가 확장되면 8 비트 Java 명령어 수행 가능

❖ 일반적으로 사용되는 Java 바이트 코드의 95%를 하드웨어로 구현

- 소프트웨어 JVM : 1.0 Caffeinemarks/MHz
- Jazelle 내장된 ARM core : 5.5 Caffeinemarks/MHz

❖ Jazelle core는 12K 보다 작은 gates 수로 구현이 가능하다.

파이프라인

❖ ARM 프로세서 코어는 파이프라인을 지원

- 프로세서의 모든 부분들과 메모리 장치들이 계속적으로 쉬지 않고 동작한다.
- 하나의 명령이 수행되고 있는 동안에 다음에 수행할 명령을 decode 하고, 그 다음에 수행할 명령이 메모리로부터 읽어 온다.

목 차

메모리 구조

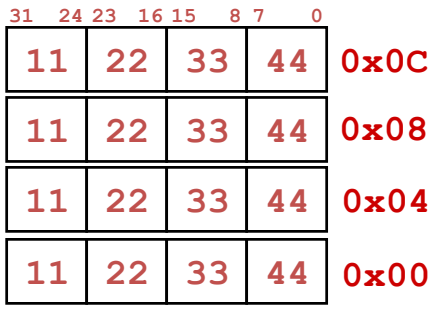
- 메모리는 프로그램과 데이터를 저장하는 공간이다.
- Big/Little Endian 지원
  - ARM core는 Big-Endian 과 Little-Endian을 모두 지원
  - 외부의 핀에 의해서 하드웨어적으로 결정된다.
- ARM 에서 사용 가능한 데이터 Type
  - byte : 8비트
  - Halfword : 16 비트, 2 바이트
  - Word : 32 비트, 4 바이트
- Un-aligned access
  - 기존의 ARM Architecture v4T, v5T, v5TE 등은 지원되지 않는다.
    - Data Abort 발생
  - ARM Architecture v6에서는 지원한다.

Aligned 과 Un-aligned 액세스

mds ACADEMY

❖ 프로세서는 메모리 액세스
- Byte, halfword(2바이트) 또는 word(4바이트) 단위로만 가능

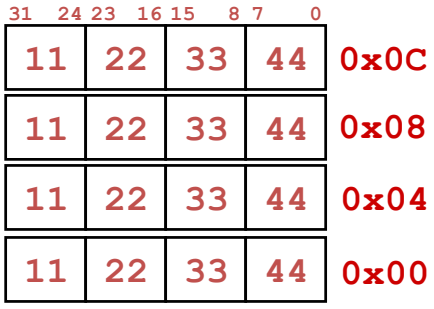
Aligned Access



Word 단위 Access :
0x00, 0x04, 0x08, …

Half-Word Access :
0x00, 0x02, 0x04, 0x06, 0x08, …

Un-aligned Access



Word 단위 Access : → **Abort**
0x01, 0x06, 0x07, …

Half-Word Access : → **Abort**
0x01, 0x03, 0x05 …

Little-Endian 과 Big-Endian

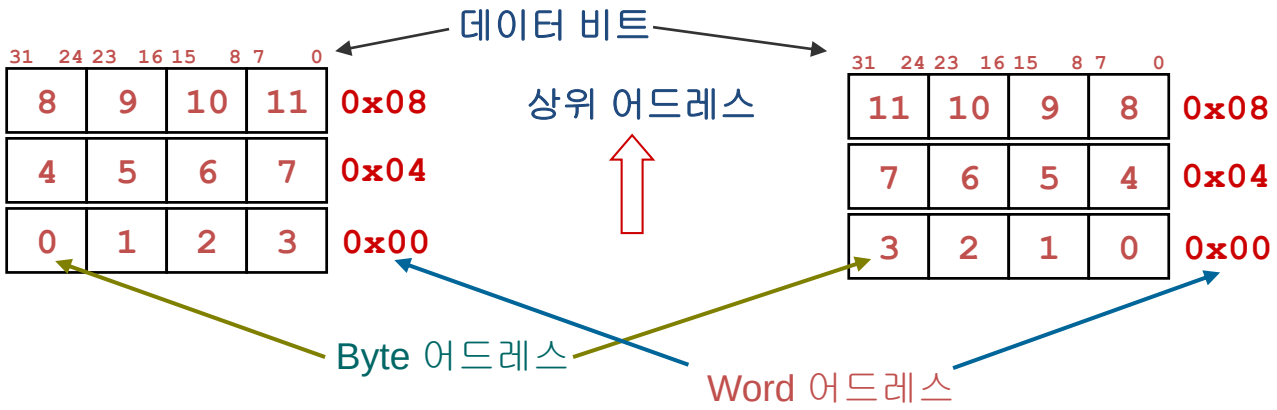
mds ACADEMY

❖ Big-Endian

- 메모리의 하위 어드레스(Byte 어드레스 “0”)에 MSB(Most Significant Byte)가 위치하고 있는 메모리 구조
- MSB는 메모리 데이터의 가장 상위 비트 24 에서 31 까지

❖ Little-Endian

- 메모리의 하위 어드레스(Byte 어드레스 “0”)에 LSB(Least Significant Byte)가 위치하고 있는 메모리 구조
- LSB는 메모리 데이터의 가장 하위 비트 0 에서 7 까지



❑ **0,4,8은 word 어드레스, 0,2,4,6,8,10은 halfword 어드레스, 0,1,2,3,4,…은 byte 어드레스를 나타낸다.**
❑ **word로 메모리를 액세스 하는데 1,2 또는 3 번지를 액세스 하면 un-aligned 액세스가 된다.**

목 차

Operating Mode (1)

❖ Operating 모드는 현재 프로세서가 어떤 권한을 가지고 어떤 종류의 작업을 하고 있는지를 나타낸다.

- User Mode
  - User task 또는 어플리케이션을 수행할 때의 프로세서의 동작 모드
  - ARM 프로세서의 동작 모드 중 유일하게 Un-privileged 모드로 메모리나 I/O 장치와 같은 시스템 자원을 사용하는데 제한이 있다.
- FIQ(Fast Interrupt Request) Mode
  - 빠른 인터럽트의 처리를 위한 프로세서의 동작 모드
- IRQ(Interrupt Request) Mode
  - 일반적으로 사용되는 인터럽트를 처리하기 위한 프로세서의 동작 모드
- SVC(Supervisor) Mode
  - 시스템 자원을 관리할 수 있는 프로세서의 동작 모드
  - Operating 시스템의 커널이나 드라이버를 처리하는 모드
  - Reset이나 소프트웨어 인터럽트(SWI)가 발생하면 ARM은 Supervisor 모드가 된다.

Operating Mode (2)

- Abort Mode
  - 명령이나 데이터를 메모리로부터 읽거나 쓸 때 오류가 발생하면 ARM은 Abort 모드가 된다.
  - Abort는 외부의 메모리 제어기에서 발생된다.
- Undefined Mode
  - ARM이 정의되지 않은 명령을 수행하려고 하면 수행되는 프로세서의 동작 모드
- System Mode
  - User 모드와 동일한 용도로 사용되지만 privilege 모드이다.
  - ARM Architecture v4이후부터 지원된다.

Operating 모드의 변경

❖ Exception

- IRQ, FIQ를 비롯한 대부분의 Operating 모드는 외부에서 발생되는 조건에 의해서 ARM 프로세서가 하드웨어적으로 변경한다.
- 외부에서 발생되는 물리적인 조건에 의해서 정상적인 프로그램의 실행을 미루고 예외적인 현상을 처리하는 것을 Exception이라 한다.
- Operating 모드의 변경은 소프트웨어에 의하여 제어 할 수도 있다.

❖ 시스템 콜

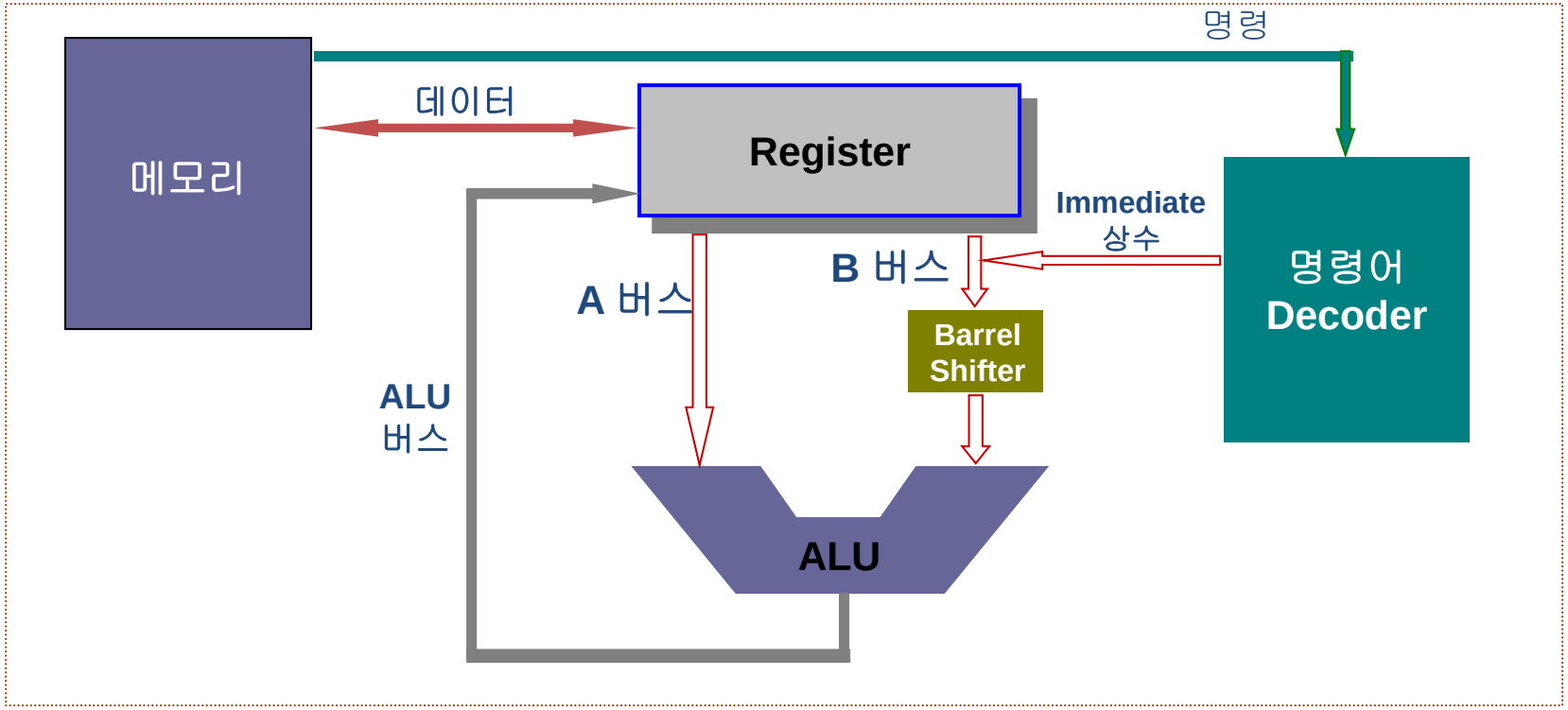
- User 모드에서 수행되는 프로그램에서 Supervisor 모드로 전환하여 시스템 자원을 사용할 수 있게 해주는 인터페이스
- OS에서는 소프트웨어 인터럽트(SWI)를 사용하여 시스템 콜을 구현

목 차

ARM 프로세서의 레지스터

❖ 레지스터

- 프로세서가 작업을 하는데 사용되는 값을 저장하는 공간
- ARM에는 32비트 길이의 37개의 레지스터가 있다.



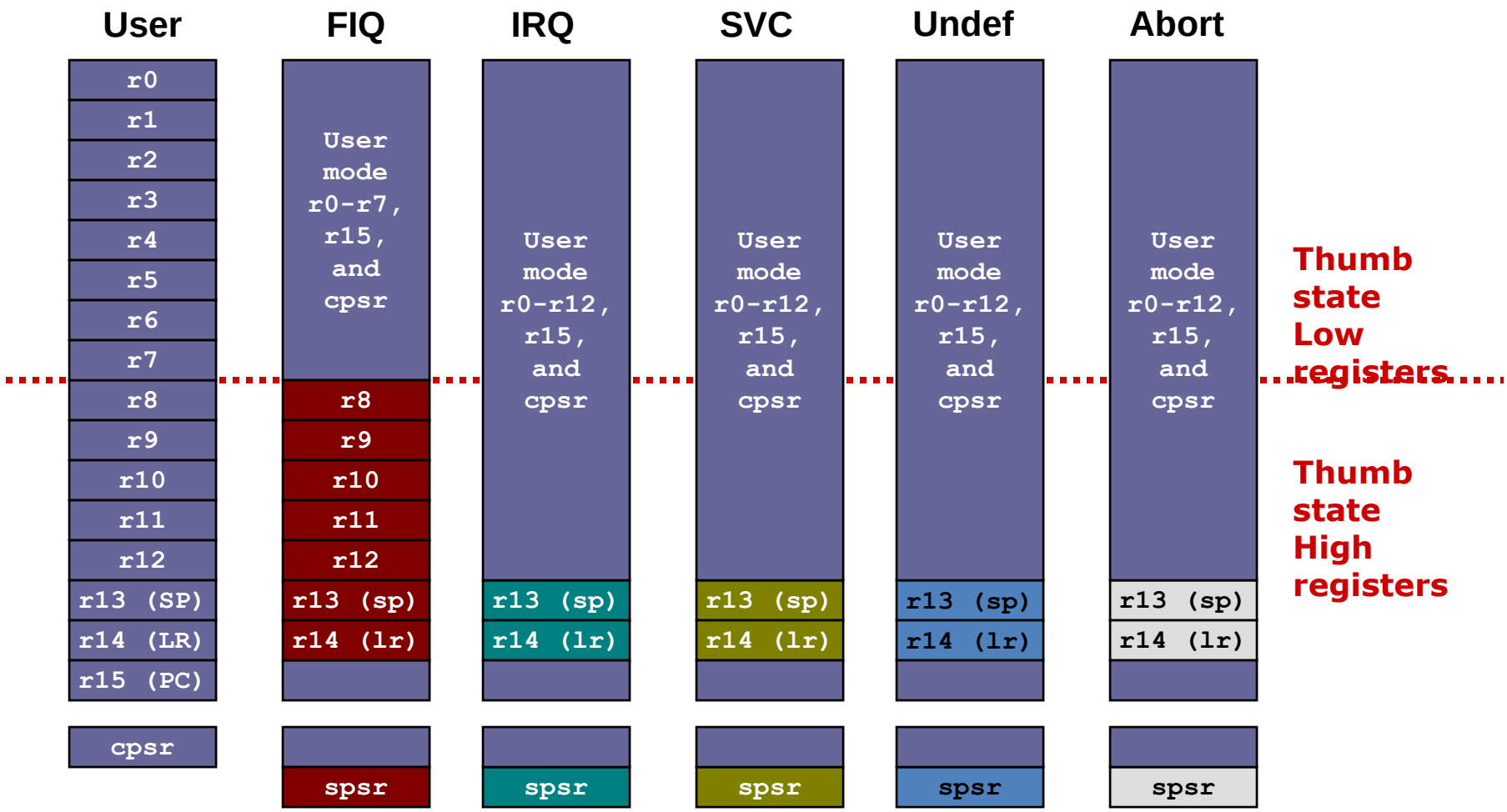
ARM 레지스터 종류

- 30개의 범용(General Purpose) 레지스터
  - 대부분 데이터 연산 등에 사용
  - 프로세서의 동작(Operating) 모드에 따라 사용되는 레지스터가 제한된다.
- 1개의 프로그램 카운터(PC : Program Counter)
  - 프로그램을 읽어올 메모리의 위치를 나타낸다.
- 1개의 CPSR(Current Program Status Register)
  - 프로세서가 수행하고 있는 현재의 동작 상태를 나타낸다.
- 5개의 SPSR(Saved Program Status Register)
  - 이전 모드의 CPSR의 복사본으로
    - Exception이 발생하면 ARM이 하드웨어적으로 이전 모드의 CPSR 값을 각각의 모드에 해당하는 SPSR에 복사
  - User, System 모드를 제외한 모든 privilege 모드에 각각 하나씩 존재

Operating 모드 별 사용 가능한 레지스터

- ARM state의 경우
  - 16개의 범용 레지스터
    - R0에서 R15의 키워드를 사용하여 관리
    - R0에서 R12는 연산 명령과 같은 범용으로 사용
    - R13(SP), R14(LR), R15(PC)는 특수한 목적으로 사용
      - 연산 명령에서 사용 할 수도 있다.
  - 1개의 CPSR
  - Privilege 모드의 경우 각각 1개의 SPSR
- Thumb state의 경우
  - 8개의 범용 레지스터
    - R0에서 R7
  - R13(SP), R14(LR), R15(PC) 레지스터
    - 연산 명령에서 사용 불가
  - 1개의 CPSR
  - Privilege 모드의 경우 각각 1개의 SPSR

Operating 모드 별 레지스터



주의: **System mode**의 경우에는 **User mode**의 레지스터를 사용한다.

Stack Pointer (SP 또는 R13)

- 프로그램에서 사용하는 스택의 위치를 저장하는 레지스터
- 프로세서의 동작 모드마다 별도로 할당된 SP 레지스터를 가지고 있다.
- ARM은 별도의 스택 명령이 없다.
  - Push나 Pop 과 같은 별도의 스택 명령을 제공하지 않는다.
  - LDR/STR, LDM/STM 같은 데이터 전송 명령을 사용하여 스택을 처리

Link Register (LR 또는 R14)

- 서브루틴에서 되돌아 갈 위치 정보를 저장하고 있는 레지스터
  - 서브루틴을 호출하는 BL 명령을 사용하면 PC 값을 자동으로 LR에 저장
  - 되돌아 갈 때는 MOV 명령을 이용하여 LR 값을 PC에 저장

```
MOV PC, LR
```

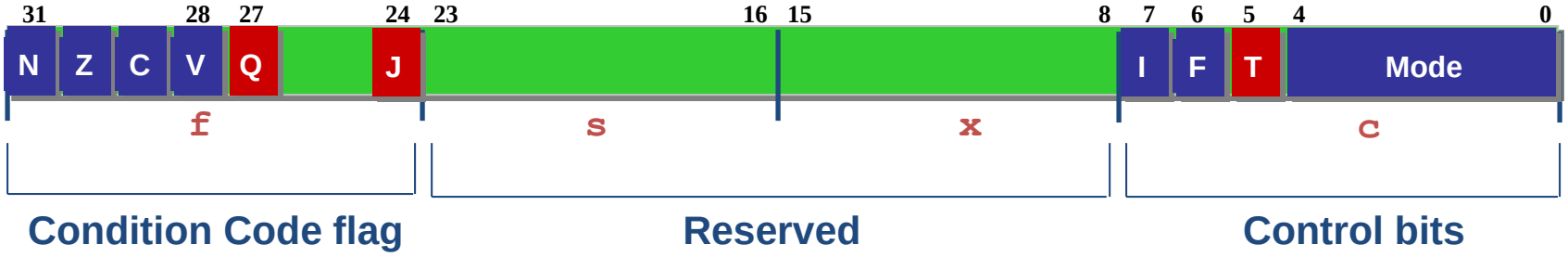
  - ADD나 SUB와 같은 연산 명령을 이용하여 되돌아갈 위치를 조정할 수도 있다.
- 프로세서의 동작 모드마다 별도로 할당된 LR 레지스터를 가지고 있다.

Program Counter (PC 또는 R15)

- 프로그램을 수행하는 위치를 저장하고 있는 레지스터
  - ARM state의 경우 위치 정보 비트 [31:2] 저장
  - Thumb state의 경우 위치 정보 비트 [31:1] 저장
- 다른 범용 레지스터와 마찬가지로 ADD, SUB와 같은 연산 명령을 사용하여 프로그램이 분기할 위치를 조정 가능
- 프로세서의 모든 동작 모드에 대하여 하나만 존재한다.

Program Status Register (PSR)

- ARM의 Program Status Register
  - 1개의 CPSR(Current Program Status Register)
  - 5개의 SPSR(Saved Program Status Register)
- PSR 레지스터의 정보
  - Condition code flag
    - ALU의 연산 결과 정보를 가지는 flag 정보를 가지고 있다.
  - Control bits
    - 프로세서를 제어하기 위한 비트로 구성되어 있다.
  - Reserved



PSR 레지스터 – flag bits

- Flag bits는 ALU를 통한 명령의 실행 결과를 나타내는 부분이다.
  - Negative flag ('N' 비트)
    - ALU 연산 결과 마이너스가 발생한 경우 세트
  - Zero flag ('Z' 비트)
    - ALU 연산 결과 0가 발생한 경우 세트
  - Carry flag ('C' 비트)
    - ALU 연산 결과 자리올림이나 내림이 발생한 경우 세트
    - shift 연산에서 carry가 발생한 경우 세트
  - oVerflow flag ('V' 비트)
    - ALU 연산 결과 overflow가 발생하면 세트
  - Q flag ('Q' 비트)
    - ARM Architecture v5TE/J에서 새롭게 추가된 saturation 연산 명령의 수행 결과 saturation이 발생하면 세트된다.
    - 이 비트는 사용자에 의해서만 클리어된다.
  - 'J' 비트
    - ARM Architecture v5TEJ에서 새롭게 추가된Java 바이트 코드를 수행하는 Jazelle state임을 나타낸다.

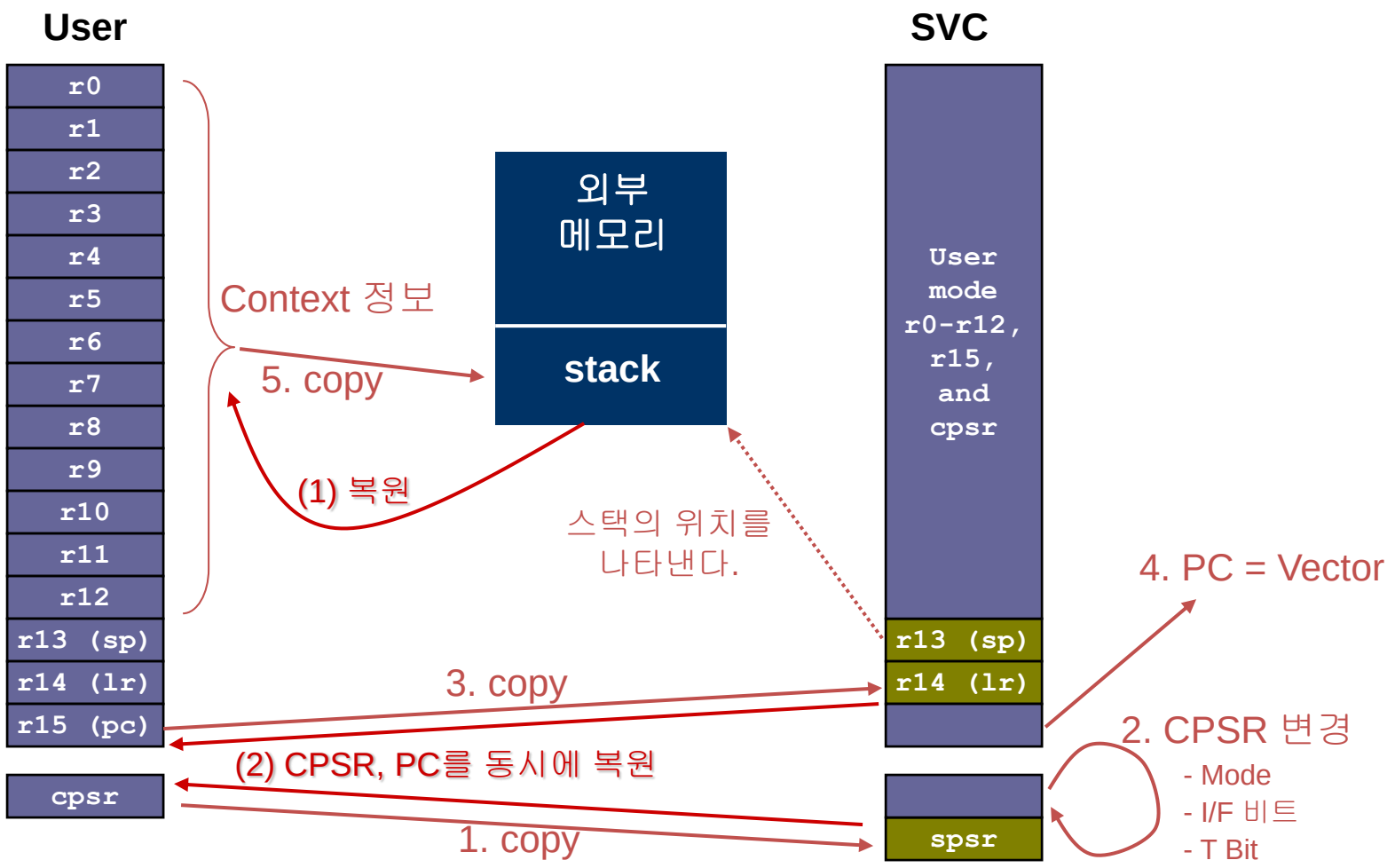
PSR 레지스터 – control bits

❖ 프로세서의 모드, 동작 state 와 인터럽트를 제어

- I/F 비트
  - IRQ('I' 비트) 또는 FIQ('F' 비트)를 disable(set) 또는 enable(clear)
- T 비트
  - ARM Architecture xT 버전인 경우 Thumb state 임을 나타낸다.
  - 상태를 나타낼 뿐 프로세서는 이 비트에 강제로 값을 write 할 수 없다.
  - BX 명령에 의해서만 제어된다
- Mode bits
  - ARM의 7 가지의 동작 모드를 나타낸다.

M[4:0]	Operating Mode	M[4:0]	Operating Mode
10000	**User** 모드	**10111**	**Abort** 모드
10001	**FIQ** 모드	**11011**	**Undefined** 모드
10010	**IRQ** 모드	**11111**	**System** 모드
10011	**SVC** 모드		

Operating 모드의 변환과 레지스터



목 차

Exception

- Exception
  - 외부의 요청이나 오류에 의해서 정상적으로 진행되는 프로그램의 동작을 잠시 멈추고 프로세서의 동작 모드를 변환하고 미리 정해진 프로 그램을 이용하여 외부의 요청이나 오류에 대한 처리를 하도록 하는 것
  - Exception의 예
    - I/O 장치에서 인터럽트를 발생시키면 IRQ Exception이 발생하고, 프로세서는 발생된 IRQ Exception을 처리하기 위해 IRQ 모드로 전환되어 요청된 인터럽트에 맞는 처리 동작 수행
- ARM의 Exception
  - Reset
  - Undefined Instruction
  - Software Interrupt
  - Prefetch Abort
  - Data Abort
  - IRQ(Interrupt Request)
  - FIQ(Fast Interrupt Request)

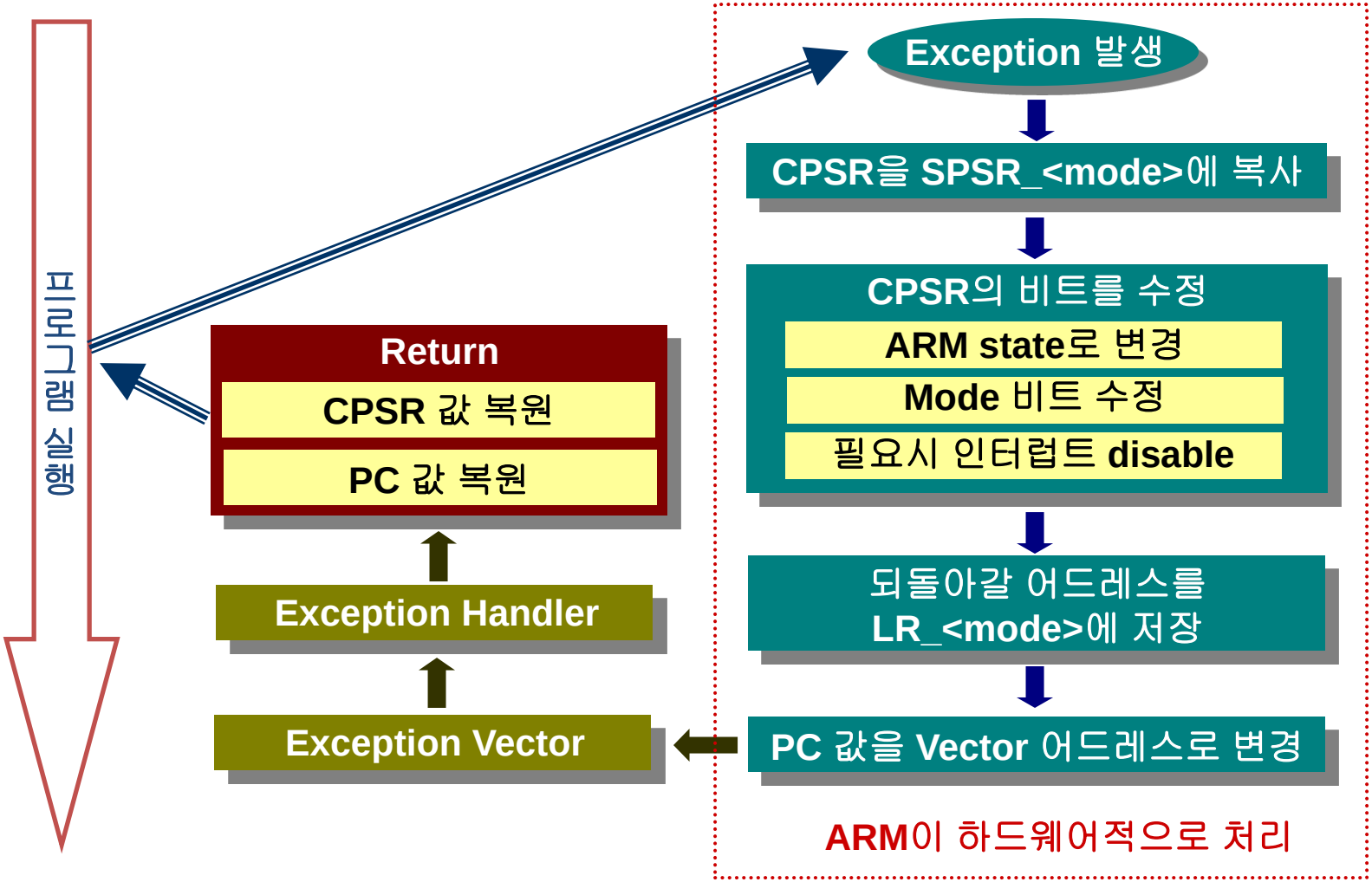
Exception Vector 와 우선순위

- Exception Vector
  - Exception이 발생하면 미리 정해진 어드레스의 프로그램을 수행
  - 미리 정해진 프로그램의 위치를 Exception Vector라 한다.
- Exception Vector Table
  - 발생 가능한 각각의 Exception에 대하여 Vector를 정의해 놓은 테이블
    - 각 Exception 별로 1 word 크기의 ARM 명령어 저장 공간을 가진다.
  - Vector Table에는 Branch 또는 이와 유사한 명령으로 실제 Exception을 처리하기 위한 루틴으로 분기 할 수 있는 명령으로 구성되어 있다.
    - FIQ의 경우는 Vector Table의 맨 상위에 위치하여 분기명령 없이 처리루틴을 프로그램 할 수 있다.
  - ARM은 기본적으로 0x00000000에 Vector Table을 둔다.
    (MMU 제어 프로그램에 의해 위치 변경 가능)
- Exception 우선 순위
  - 동시에 Exception이 발생하는 경우 처리를 위해 우선 순위 지정

Exception Vector Table

Vector Address	Exception	우선순위	동작모드 전환
0x0000 0000	Reset	1 (High)	Supervisor(SVC)
0x0000 0004	Undefined Instruction	6 (Low)	Undefined
0x0000 0008	Software Interrupt(SWI)	6	Supervisor(SVC)
0x0000 000C	Prefetch Abort	5	Abort
0x0000 0010	Data Abort	2	Abort
0x0000 0014	Reserved		
0x0000 0018	IRQ	4	IRQ
0x0000 001C	FIQ	3	FIQ

Exception Handling



목 차

인터럽트 Latency

❖ 인터럽트 Latency

- 외부 장치에서 인터럽트를 요청하여 인터럽트를 처리하기 위한 루틴 (인터럽트 핸들러)로 가는데까지 실제 소요되는 시간
- 시스템 성능의 중요 요소이다.

❖ FIQ 인터럽트 Latency

- 최소 4 사이클
  - 인터럽트 인식 (최소 2사이클) + FIQ Exception Vector로 점프 (2 사이클)
- 최대 28 또는 31 사이클 소요
  - Vector Table에 FIQ 핸들러가 작성되는 경우 28 사이클
  - Vector Table에 Branch 명령이 사용되는 경우 31 사이클

인터럽트 Latency 계산

❖ FIQ 인터럽트 Latency 계산

① I/O 장치가 요청한 인터럽트를 프로세서가 인지하는 가장 긴 시간 3 사이클

② 가장 오랜 사이클이 소요되는 LDM 명령 종료 시간 20 사이클

③ 우선 순위가 높은 Data Abort의 처리 루틴으로 분기하는 시간 3 사이클

④ FIQ Exception Vector로 점프하는데 걸리는 시간 2 사이클

⑤ Exception Vector에 Branch 명령이 사용되는 경우, 인터럽트 처리 루틴으로 점프하는 시간 3 사이클

❖ IRQ 인터럽트 Latency 계산

- FIQ의 인터럽트 Latency와 유사
- IRQ는 FIQ보다 우선 순위가 낮아 지연될 수 있다
- 항상 Branch 명령이 사용된다.

목 차

프로세서의 Reset 과 Start-up 코드

❖ 프로세서에 리셋 신호가 입력되면 실행 중이던 명령을 멈추고,

①SPSR_svc에 CPSR 값을 복사 (일반적으로 의미없음)

②CPSR의 값을 변경

- Mode bit M[4:0]를 Supervisor 모드인 10011b로 변경
- I 비트와 F 비트를 1로 세트하여 인터럽트를 disable
- T 비트를 0으로 클리어하여 ARM state로 변경

③PC 값을 LR_SVC 레지스터에 복사 (일반적으로 의미 없음)

④PC 값을 Reset Vector 어드레스인 0x00000000으로 변경

⑤Reset 핸들러로 분기하여 시스템의 초기화 수행

Reset Handler

- Reset Vector에서 분기된 어셈블리어로 작성된 처리 루틴
- 주요 동작
  - 시스템 초기화 작업 수행
    - 시스템의 clock
    - 메모리 컨트롤러
    - 입출력 포트의 구성
    - MMU 등
  - 인터럽트와 스택 초기화
- 리셋 핸들러의 마지막에서는 main() 함수와 같은 C로 구성된 함수를 호출
- 이 부분을 startup 코드라고 부른다.

Startup 코드

❖ Startup 코드를 작성하기 위해서는

- Programmer's model, 특히 명령어
- 시스템 하드웨어 구조 및 기능에 대한 전반적인 사항

등을 모두 알아야 한다.

❖ Startup 코드의 필수 사항

- Exception Vector Table의 설정
- Reset 핸들러의 구성

목 차

ARM Architecture

- Programmer's Model은 ARM Architecture의 분류 기준
  - Architecture가 동일하면 Programmer's model이 동일하다
    - 프로그램의 호환이 가능하다
  - Programmer's Model
    - 명령어(Instruction Set)
    - 데이터 구조
    - 동작 모드
    - 레지스터의 구조
    - Exception 처리 방식 등
- 동일한 Architecture에도 여러 개의 프로세서가 있을 수 있다
- 동일한 프로세서 Family 라도 Architecture가 서로 다를 수 있다.

ARM Architecture 와 프로세서

mds ACADEMY

Architecture	특 징	프 로 세 서
v1, v2, v3	이전 버전, 현재는 거의 사용 안됨	ARM610, …
v4	System 모드 지원	StrongARM(SA-1110, …)
v4T	v4의 기능과 Thumb 명령 지원	ARM7TDMI, ARM720T ARM9TDMI, ARM940T, ARM920T
v5TE	v4T의 기능 ARM/Thumb Interowrk 개선 CLZ 명령, saturation 명령, DSP Multiply 명령 추가	AEM9E-S, ARM966E-S, ARM946E-S ARM1020E XScale
v5TEJ	v5TE의 기능 Java 바이트 코드 실행	ARM7EJ-S, ARM9EJ-S, ARM1020EJ-S
v6	SIMD 명령(MPEG4 같은 미디어 처리용) Unaligned access 지원	ARM1136EJ-S
v7	v7-A : VFP, NEON, Jazelle RCT, Thumb-2, 13-단계 슈퍼스칼라 파이프라인 v7-R : (FPU) v7-M : Thumb-2 only.	Cortex-A8, Cortex-A9, Cortex-A9 MPCore, Cortex-A12, Cortex-A15, Cortex-R4(F), Cortex-M1, Cortex-M3, Cortex-M4
v8	64비트 명령어 지원	Cortex-A53, Cortex-A57, Cortex-A72

T : Thumb 명령 지원, E : DSP 기능 확장, J : Java 명령 지원, ARMxxx-S : Synthesizable core

ARM 프로세서 Family

구분	ARM7 Family	ARM9 Family	ARM9E Family	ARM11 Family	XScale
CPU Speed (MHz)	33 ~ 72	120 ~ 266	120 ~ 266	300 ~ 400	300 ~ 400
Architecture	V4T	V4T	V5TE	V6	V5TE
Pipeline	3	5	5	8	7~8
버스 구조	Von-Neumann	Harvard	Harvard	Harvard	Harvard

연 습 문 제

- ARM Architecture의 특징 4가지를 설명하라
- Programmer's Model이란 무엇인가 ?
- 32 비트 ARM 명령어가 가지는 특징은 무엇인가 ?
- 16 비트 Thumb 명령어가 가지는 특징은 무엇인가 ?
- ARM 프로세서의 state에 대하여 설명하라
- Little-endian 과 Big-endian의 차이점은 무엇인가 ?
- ARM의 7가지 Operating Mode에 대하여 설명하라.

연 습 문 제

- ❖ 스택 포인터와 링크 레지스터로 사용되는 레지스터는 ?
- ❖ 프로그램 Status 레지스터 (PSR)은 무엇인가 ?
- ❖ Exception 이란 무엇인가 ?
- ❖ Exception Vector에 대하여 설명하라.
- ❖ Exception이 발생하면 누가 어떤 행동을 하는지 설명하라.
- ❖ 인터럽트 Latency란 무엇이며, FIQ의 인터럽트 Latency는 얼마인가 ?
- ❖ 하드웨어적으로 Reset 신호가 구동되면 ARM은 어떤 동작을 하는가 ?

질의 응답