ARM Architecture

https://youtu.be/KlZuJNx2wlI

IT융합공학부 송경주

ARM 프로세서

- RISC(Reduced Instruction Set Computer)기반의 제품군
- 확장성이 좋아 광범위한 장치에서 사용 가능함
- 비용이 저렴하고 전력소모가 적어 스마트폰 및 태블릿, 임베디드 시스템 등 다양한 장치에서 사용됨
- 스마트폰 및 태블릿의 확산으로 인기가 많아짐
- 간단하고 효율적으로 설계된 명령어 세트가 제공되어 프로그램을 빠르게 실행할 수 있음

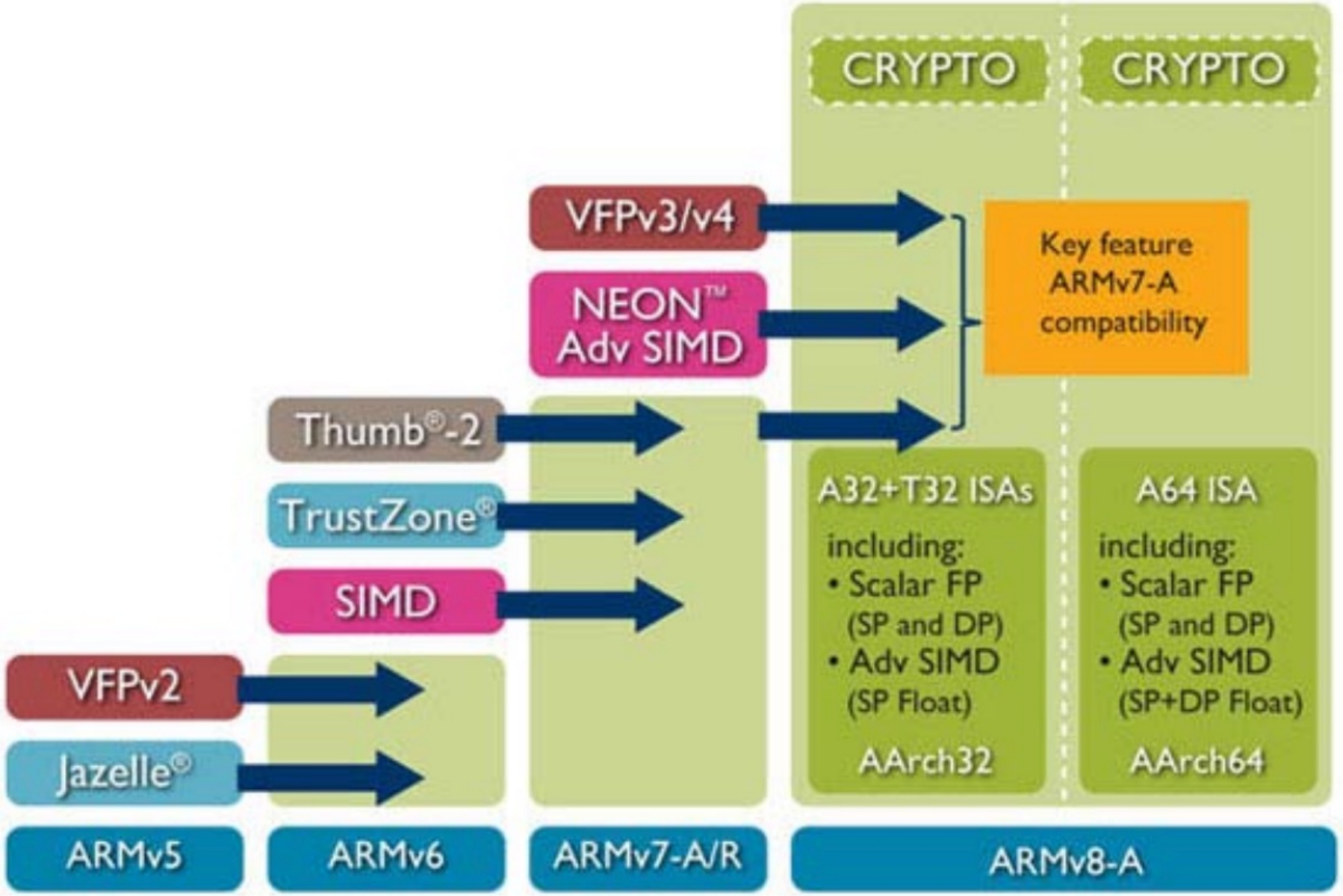
ARM 프로세서

vs CISC(Complex Instruction Set Computer)

- X86과 같은 CISC 프로세서는 단일 명령으로 복잡한 작업을 수행할 수 있으며 상당한 양의 내부 논리를 가진다.
- 반면 ARM 코어는 훨씬 적은 수의 트랜지스터로 실행되며 많은 명령어가 단일 주기에서 실행된다.

- ARM 아키텍처의 단점 : Intel과 같은 아키텍처에서 제공하는 다중 스레딩을 지원하지 않음

ARM 프로세서


CRYPTO
CRYPTO
VFPv3/v4
NEON™
Adv SIMD
Key feature
ARMv7-A
compatibility
Thumb®-2
TrustZone®
SIMD
A32+T32 ISAs
including:
• Scalar FP
(SP and DP)
• Adv SIMD
(SP Float)
AArch32
A64 ISA
including:
• Scalar FP
(SP and DP)
• Adv SIMD
(SP+DP Float)
AArch64
VFPv2
Jazelle®
ARMv5
ARMv6
ARMv7-A/R
ARMv8-A

ARM 프로세서

- Vector Floating Point (VFP) : 스마트폰, 음성 압축 및 압축 해제, 3차원 그래픽 및 디지털 오디오, 프린터, 셋톱 박스 및 자동차 애플리케이션과 같은 광범위한 애플리케이션에 적합한 부동 소수점 연산을 제공
- Jazelle : Java 바이트코드를 기존 ARM 및 Thumb 모드와 함께 세 번째 실행 상태(및 명령어 세트)로 ARM 아키텍처에서 직접 실행할 수 있도록 하는 기술
- Thumb : 32bit ARM프로세서에서 16bit 명령어를 지원하는 기능 (명령어 길이를 줄여 바이너리 크기를 줄이기 위한 방법)
- Thumb-2 : 2003년 발표된 *ARM1156 core* 에서 도입됨, Thumb의 제한된 16비트 명령어 세트를 추가 32비트 명령어로 확장하여 명령어 세트에 더 많은 폭을 제공하여 가변 길이 명령어 세트 생성
- Thumb Execution Environment (ThumbEE) : Thumb2 16/32비트 명령어 세트의 변형, 2005년에 발표되었고 2011년에 사용이 중단됨
- SIMD : Java Virtual Machine(JVM)에 의한 바이트코드 실행의 하드웨어 가속을 위한 아키텍처 지원을 제공
- NEON : Arm Cortex-A 및 Arm Cortex-R 시리즈 프로세서를 위한 고급 Single Instruction Multiple Data (SIMD) 아키텍처의 확장

ARMv8

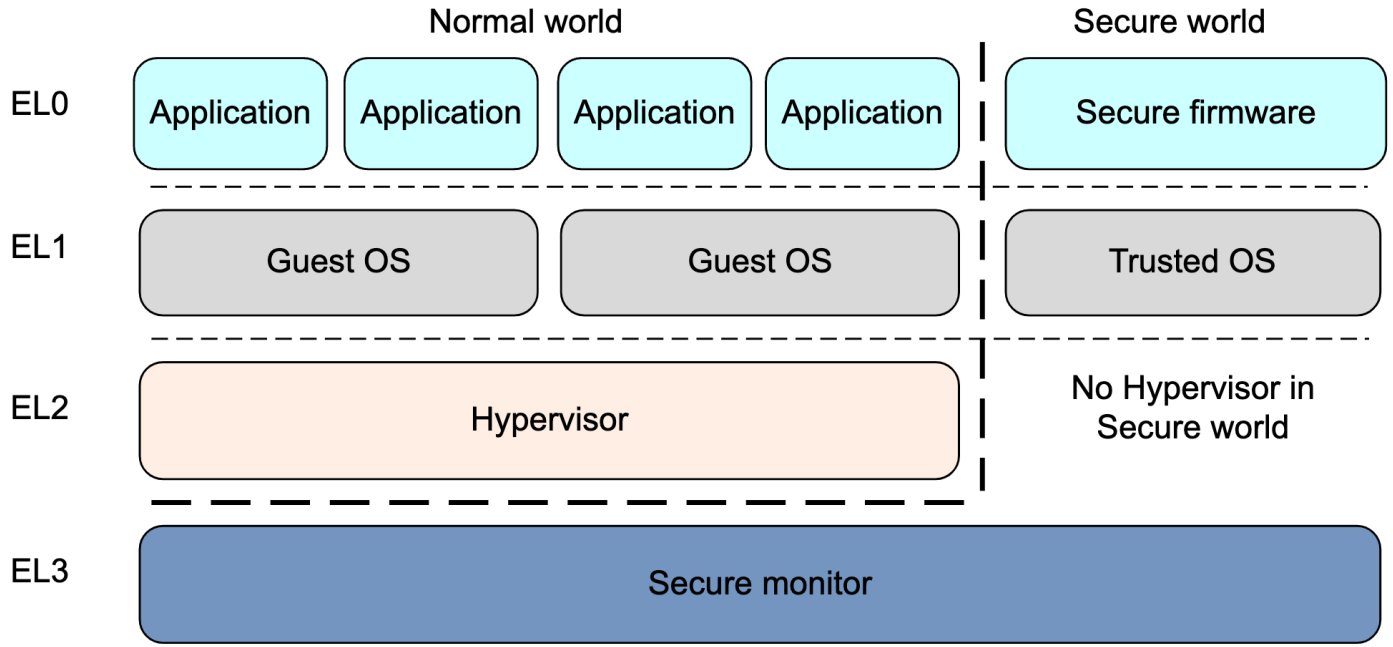
- 2011년 10월 ARM에서 발표한 64-bit 아키텍처
- 32bit 실행과 64bit 실행을 모두 포함
- ARMv8에서는 네 가지 Exception level 중 하나에서 실행됨
- 64-bit 실행 상태 AArch64와 32-bit 실행 상태 AArch32를 전환하며 프로세서를 동작하며 각 실행 상태는 독립적임
- 기존 ARMv7 소프트웨어와의 하위 호환성을 유지하면서 64bit 레지스터로 실행을 수행하는 기능 도입

ARMv8

[Exception levels]

- **EL0** : 일반 사용자 애플리케이션
- **EL1** : OS 커널을 실행할 수 있음
- **EL2** : Non-secure operation의 가상화 지원을 제공, 하이퍼바이저를 실행할 수 있음
- **EL3** : Secure상태와 Non-secure 상태의 두 가지 보안 상태 간 전환 지원 (보안 모니터 실행 가능)

*하이퍼바이저 = 가상 머신 모니터 : 가상 머신(VM)을 생성하고 실행하는 프로세서



ARMv8

- 인터프로세싱

- AArch64와 AArch32의 execute state 사이를 이동하는 방식이며 실행상태에서는 Exception level이 변경될 때만 이동할 수 있다. (즉, exception을 상위 레벨로 가져가거나 하위 레벨로 반환할 때만 execute state가 변경될 수 있음)

[더 높은 Exception level로 예외를 처리할 때]

1. AArch32 에서 AArch64 상태로 변경

[더 낮은 Exception level로 예외를 처리할 때]

1. AArch64 에서 AArch32 상태로 변경

ARMv8

[AArch64]

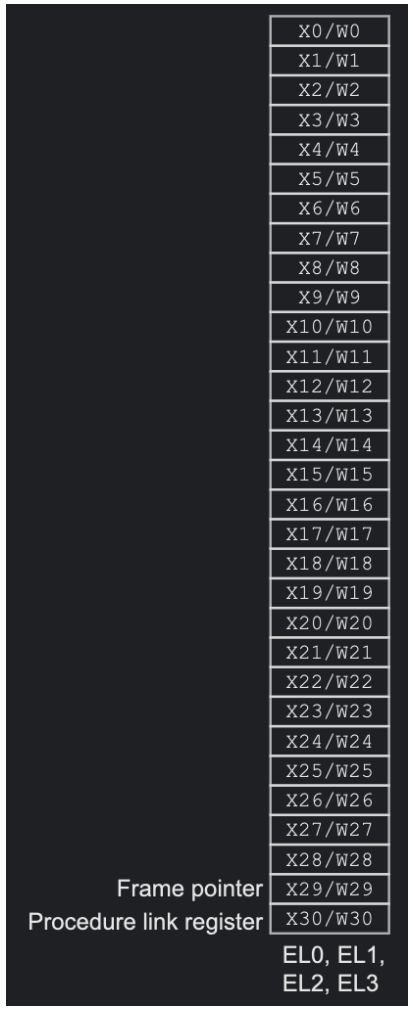
- 31개의 64bit 범용 레지스터(R0-R30)와 64bit 프로그램 카운터(PC), 스택 포인터(SP), 예외 링크 레지스터(ELR)를 사용하는 ARMv8-A 64bit 실행상태
- SIMD 벡터 및 스칼라 부동 소수점 지원을 위한 32개의 128bit 레지스터 (V0-V31) 제공
- 32bit의 고정 길이를 가지며 항상 리틀 엔디안이다.

[AArch32]

- 13개의 32bit 범용 레지스터(R0-R12)와 32bit 프로그램 카운터(PC), 스택 포인터(SP), 링크 레지스터(LR)를 사용하는 ARMv8-A 32bit 실행상태
- 고급 SIMD 벡터 및 스칼라 부동 소수점 지원을 위한 32개의 64bit 레지스터 제공
- ARMv7-A와 같은 이전 32비트 종속 ARM 버전과 역호환 가능

ARMv8

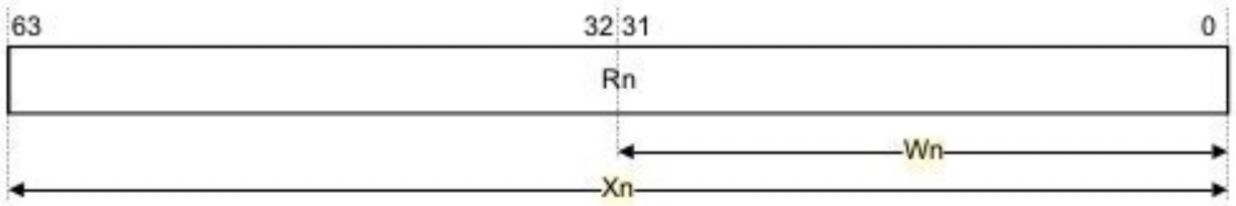
- AArch64 register

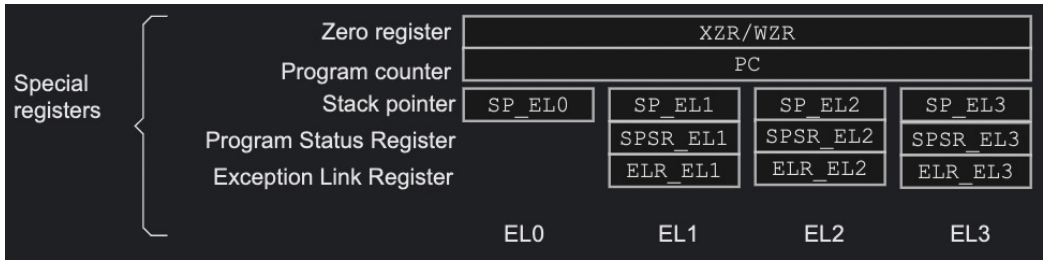


AArch64 범용 레지스터

- AArch64 실행 상태는 항상 모든 예외 수준에서 액세스할 수 있는 31 × 64 bit 범용 레지스터 제공
- 각 레지스터는 64비트의 폭을 가지며 일반적으로 레지스터 X0-X30 라고 지칭함
- AArch64 64비트 범용 레지스터(X0-X30)에 32비트(W0-W30) 형식이 있음(32비트 W 레지스터는 해당 64비트 X 레지스터의 하위 절반을 형성)

  *** W0에 0xFFFFFFFF을 쓰면 X0이 0x00000000FFFFFFFF가 됨**





AArch64 특수 레지스터

Wn	32 bits	General-purpose register: n can be 0-30
Xn	64 bits	General-purpose register: n can be 0-30
WZR	32 bits	Zero register
XZR	64 bits	Zero register
WSP	32 bits	Current stack pointer
SP	64 bits	Current stack pointer

ARMv8

```c
#include <stdio.h>
#include <stdint.h>

extern void test_func(uint8_t *a, uint8_t *b);

int main(int argc, const char * argv[]) {
    uint8_t a[32], b[32];


    for (int i=0; i<32 ; i++){
        a[i] = i;
        b[i] = i*2;
    }

    for(int i=0; i<32 ; i++){
        printf("a=%d\n", a[i]);
        printf("b=%d\n\n", b[i]);
    }

    test_func(a, b);

    printf("====================\n");

    for(int i = 0; i<32; i++)
        printf("%d \n", a[i]);

    return 0;
}
```

```asm
//
//  test.s
//  ARM_practice
//
//  Created by 송경주 on 2023/03/21.
//
.globl test_func
.globl _test_func

test_func:
_test_func:

    LD1 {v0.16b}, [x0]
    LD1 {v1.16b}, [x1]
    MOV x2, #0

LOOP:
    ADD.16b v0, v0, v0
    ADD x2, x2, #1
    CMP x2, #2
    BLT LOOP

    ST1 {v0.16b}, [x0]

    RET
```

```
a=0
a=1
a=2
a=3
a=4
a=5
a=6
a=7
a=8
a=9
a=10
a=11
a=12
a=13
a=14
a=15
a=16
a=17
a=18
a=19
a=20
a=21
a=22
a=23
a=24
a=25
a=26
a=27
a=28
a=29
a=30
a=31
```

```
0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

Q & A