TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 윤연성
whatmatters@naver.com

sudo apt-get update
sudo apt-get install qemu-user-static qemu-system
sudo apt-get install gcc-arm-linux-gnueabi

이후에 아무런 C 소스 파일을 작성한다.

arm-linux-gnueabi-gcc -g 소스파일

sudo apt-get install gdb-multiarch

터미널을 2 개 띄운다.
A 터미널에서 아래 명령어를 수행한다.

qemu-arm-static -g 1234 -L /usr/arm-linux-gnueabi ./a.out

B 터미널에서 아래 명령어를 수행한다.

gdb-multiarch

file a.out

target remote localhost:1234

b main

c

이후부터 디버깅을 진행하면 된다.
다른 모든 프로그램도 이러한 방법으로 디버깅을 수행할 수 있다.
이제 ARM Architecture 에 대해 Deep Down 할 수 있는 기점이 마련되었다.

```
#include <stdio.h>

int main(void)
{
	register unsigned int r0 asm("r0");
	register unsigned int r1 asm("r1");
	register unsigned int r2 asm("r2");

	r1 = 77;
	r2 = 37;

	asm volatile("add r0,r1,r2");

	printf("r0 = %d\n", r0);
	return 0;
}
```

42 & ~(2^3 -1)
42 & ~(7)

42 를 2^3 의 배수로 정렬
40

bind: Address already in use
qemu: could not open gdbserver on port 1235
이라고 나오면 2 번쨰 터미널에서 스톱한거 q 로 꺼야됨

```
asm volatile("mvneq r1, #0");
```

#0 을
r1 에 넣는거
mvn 연산 eq 조건

mov

mvn, mov, add, sub
cmp, tst, teq, and
orr, eor, bic

arm 은 무조건 4byte (32bit) 단위 동작

4bit, 28bit

mov 28bit 0xfff ffff ---→ 0x400 0000

ARM 의 레지스터 구성

-총 37 개의 32bit 레지스터로 구성

-범용 레지스터 30 + 상태 레지스터 6 + PC

-모드별로 17 개의 레지스터를 볼 수 있음(r0~r15, CPSR (+ SPSR)

이렇게 구성되어 있으며, 범용 레지스터가 30 개라는 의미는 잠시후 나오는 그림에서 알아보자.

이번엔 레지스터의 용도

PC

- 프로그램을 읽어올 메모리의 위치를 나타낸다.

- 32 비트 ARM 모드에서는 [31:2]에 읽어올 어드레스를 저장한다.

- 16 비트 Thumb 모드에서는 [31:1]에 읽어올 어드레스를 저장한다.

- PC 는 모든 동작모드에서 공유하여 사용할 수 있다.

CPSR

- 현재 동작중인 프로세서의 상태를 나타낸다.

-모든 동작모드에서 공유하여 사용할 수 있다.

SPSR

- 이전 동작모드의 CPSR 의 정보를 복사하여 저장한다.
- 표현되는 정보는 CPSR 과 같다.
- User, System 모드를 제외한 5 개 동작모드 전환시 이전 동작모드의 CPSR 저장.
- 각 모드별 1 개씩 할당하여 사용

-r0 ~ r12 : 일반적인 레지스터

-r13 :스택 포인터(sp) - 스택의 최상단을 가리키고 있어 스택 관리

-r14 : 링크 레지스터(link) - 함수를 만나 분기 했을 때 복귀 주소 저장

-r15 : 프로그램 카운터(PC) - 수행 명령어 주소 관리

-CPSR : 현재 모드의 상태 레지스터

ADD R0, R1, R2 = 무조건 R1 + R2 하여 R0 에 저장

```
asm volatile("add r0,r1,r2");
```

SUBGT R3, R3, #1 = 크다(Great then)조건이면 R3 - 1 을 R3 에 저장

```
if(r1 > r2)
    asm volatile("subgt r3,r3,#1");
```

RSBLES R4, R5, #5 = 작거나 같다(Less Equal)조건이면 R5 - 5 를 R4 에 저장하고 연산결과에 따라 CPSR 플래그 세트

```
if(r2 <= r1)
 asm volatile("rsble r4, r5, #5");
```

AND R0, R1, R2 =무조건 R1 AND R2 하여 R0 에 저장

```
asm volatile("and r0, r1, r2");
```

BICEQ R2, R3, #7 = 같다(Equal) 조건이면 R3 에서 7 에 해당하는 비트를 클리어 하여 R2 에 저장

```
if(r0 == r1)
 {
r3 = 42;
  asm volatile("biceq r2, r3, #7");
 }
```

EORS R1, R3, R0 = R3 과 R0 를 XOR 하여 R1 에 저장하고 연산결과에 따라 CPSR 플래그 설정

CMP R0, R1 = 무조건 R0 와 R1 을 산술적으로 비교하여 그 결과에 따라 CPSR 의 플래그를 설정

TSTEQ R2, #5 = EQ(Equal : 같다) 조건이면 R2 와 #5 를 비교 연산한 후 결과로 CPSR 의 플래그 설정

MOV R0, R1 = 무조건 R1 의 값을 R0 에 대입한다. R0 := R1

MOVEQ R2, #5 = EQ(Equal : 같다) 조건이면 5 를 R2 에 대입한다

```
asm volatile("cmp r0, r1");
sm volatile("mov r2, #3");
asm volatile("tsteq r2, #5");
asm volatile("cmp r0, r1");
asm volatile("mvneq r1, #0");
```