TI DSP, MCU 및 Xilinx Zynq FPGA

프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 하성용
accept0108@naver.com

45 일차

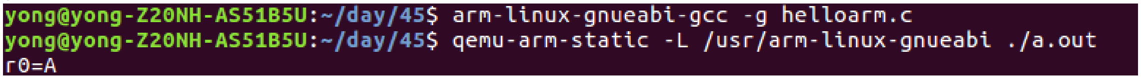
실행 -
```
arm-linux-gnueabi-gcc -g (소스파일)
qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
```

gdb -
```
arm-linux-gnueabi-gcc -O0 -g (소스파일)
qemu-arm-static -g 1234  -L /usr/arm-linux-gnueabi ./a.out
```

새 터미널열고
```
gdb-multiarch
file a.out
tar rem localhost:1234
b main 혹은 b *(해당 주소)
c
```

helloarm.c

```
#include<stdio.h>

char test[] = "HelloARM";

void show_reg(unsigned int reg)
{
        int i;

        for(i=31; i>=0;)
                printf("%d",(reg>>i--)&1);
        printf("\n");
}

int main(void)
{
        register unsigned int r0 asm("r0")=0;
        register char *r1 asm("r1")=NULL;
        register unsigned int *r2 asm("r2")=NULL;
        register unsigned int r3 asm("r3")=0;
        register unsigned int r4 asm("r4")=0;
        register unsigned int r5 asm("r5")=0;

        r1=test;

        asm volatile("ldreqb r0, [r1, #0x5]");

        printf("r0=%c\n",r0);

        return 0;
}
```

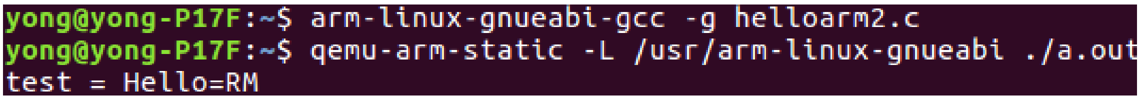


```
yong@yong-Z20NH-AS51B5U:~/day/45$ arm-linux-gnueabi-gcc -g helloarm.c
yong@yong-Z20NH-AS51B5U:~/day/45$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0=A
```

//
대문자 A 가 나오는 이유는 시작주소에서 1 바이트씩 5 개(5 바이트) 지나가서
0(h) – 1(e) – 2(l) – 3(l) – 4(o) –
5(A) → 대문자 A 가 출력
그래서 r0 에 5 가 들어감

helloarm2.c

```
#include<stdio.h>

char test[] = "HelloARM";

void show_reg(unsigned int reg)
{
        int i;

        for(i=31; i>=0;)
                printf("%d",(reg>>i--)&1);
        printf("\n");
}

int main(void)
{
        register unsigned int r0 asm("r0")=0;
        register char *r1 asm("r1")=NULL;
        register unsigned int *r2 asm("r2")=NULL;
        register unsigned int r3 asm("r3")=0;
        register unsigned int r4 asm("r4")=0;
        register unsigned int r5 asm("r5")=0;

        r1=&test[5];                            // r1=test;

        asm volatile("mov r0, #61");
        asm volatile("strb r0, [r1]");  // strb r0, [r1,#5]

        printf("test = %s\n", test);

        return 0;
}
```

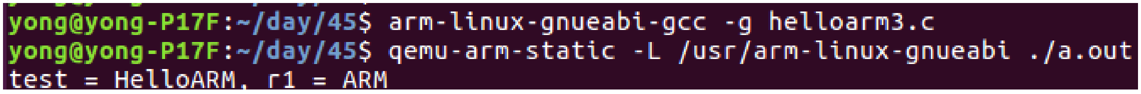


```
yong@yong-P17F:~$ arm-linux-gnueabi-gcc -g helloarm2.c
yong@yong-P17F:~$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test = Hello=RM
```

//
ldr 의 반대
레지스터에서 메모리로 가는것
61 은 아스키코드에서 ‘=’
r0 을 r1 으로 집어넣을것
r1 은 대문자 A
아스키코드에 =으로바뀐다는거
helloARM → hello=RM 으로 바뀜

helloarm3.c // !(느낌표) 옵션

```
#include<stdio.h>

char test[] = "HelloARM";

void show_reg(unsigned int reg)
{
		int i;

		for(i=31; i>=0;)
				printf("%d",(reg>>i--)&1);
		printf("\n");
}

int main(void)
{
		register unsigned int r0 asm("r0")=0;
		register char *r1 asm("r1")=NULL;
		register unsigned int *r2 asm("r2")=NULL;
		register unsigned int r3 asm("r3")=0;
		register unsigned int r4 asm("r4")=0;
		register unsigned int r5 asm("r5")=0;

		r1=test;

		asm volatile("mov r2, #0x5");
		asm volatile("ldr r0, [r1,r2]!");

		printf("test = %s, r1 = %s\n", test,r1);

		return 0;
}
```



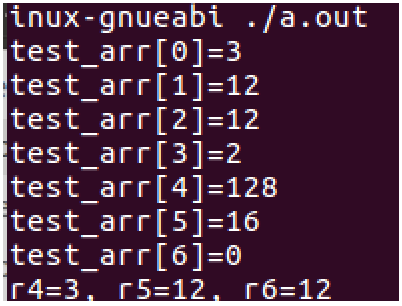
```
yong@yong-P17F:~/day/45$ arm-linux-gnueabi-gcc -g helloarm3.c
yong@yong-P17F:~/day/45$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test = HelloARM, r1 = ARM
```

//
! (느낌표) 가 하는일
이동하는데까지 값을 갱신시키라는뜻
r1 은 시작주소인데 r2
hello 를 넘기고 대문자 A 부터 시작하니까
A 부터 출력

sti=스토어 멀티플의 약자

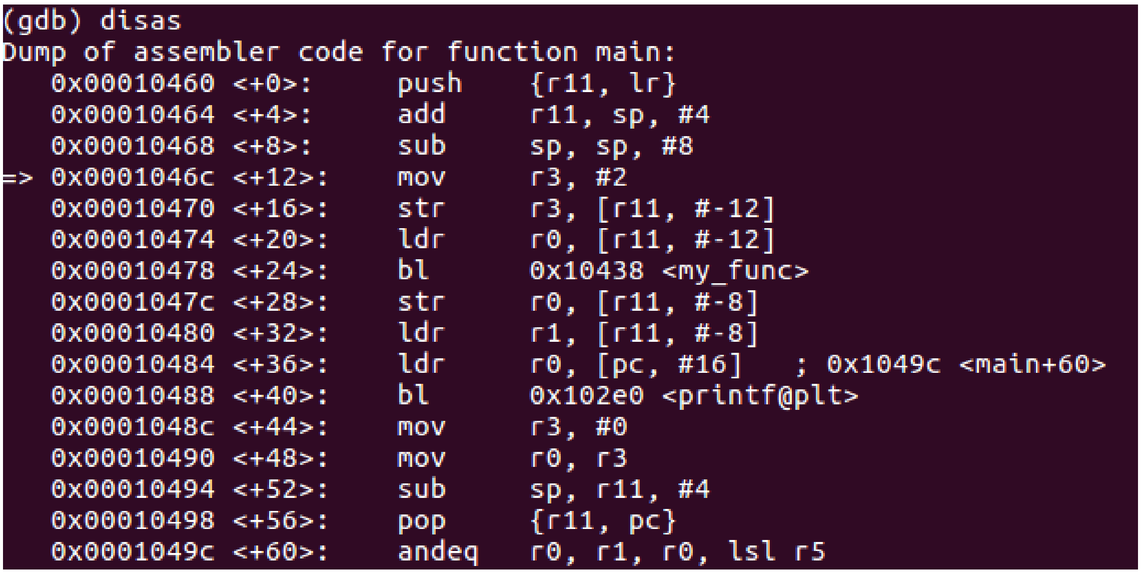
```
#include<stdio.h>

int main(void)
{
		int i;
		unsigned int test_arr[7]={0};

		register unsigned int *r0 asm("r0")=0;
		register unsigned int r1 asm("r1")=0;
		register unsigned int r2 asm("r2")=0;
		register unsigned int r3 asm("r3")=0;
		register unsigned int r4 asm("r4")=0;
		register unsigned int r5 asm("r5")=0;
		register unsigned int r6 asm("r6")=0;
```

```
        r0=test_arr;

        asm volatile("mov r1, #0x3\n"
                                "mov r2, r1, lsl #2\n"
                                "mov r4, #0x2\n"
                                "add r3, r1, lsl r4\n"
                                "stmia r0!,{r1,r2,r3}\n"
                                "str r4, [r0]\n"
                                "mov r5, #128\n"
                                "mov r6, r5, lsr #3\n"
                                "stmia r0, {r4,r5,r6}\n"
                                "sub r0, r0, #12\n"
                                "ldmia r0,{r4,r5,r6}");

        for(i=0;i<7;i++)
                printf("test_arr[%d]=%d\n",i,test_arr[i]);

        printf("r4=%u, r5=%u, r6=%u\n",r4,r5,r6);
        return 0;
}
```



```
inux-gnueabi ./a.out
test_arr[0]=3
test_arr[1]=12
test_arr[2]=12
test_arr[3]=2
test_arr[4]=128
test_arr[5]=16
test_arr[6]=0
r4=3, r5=12, r6=12
```

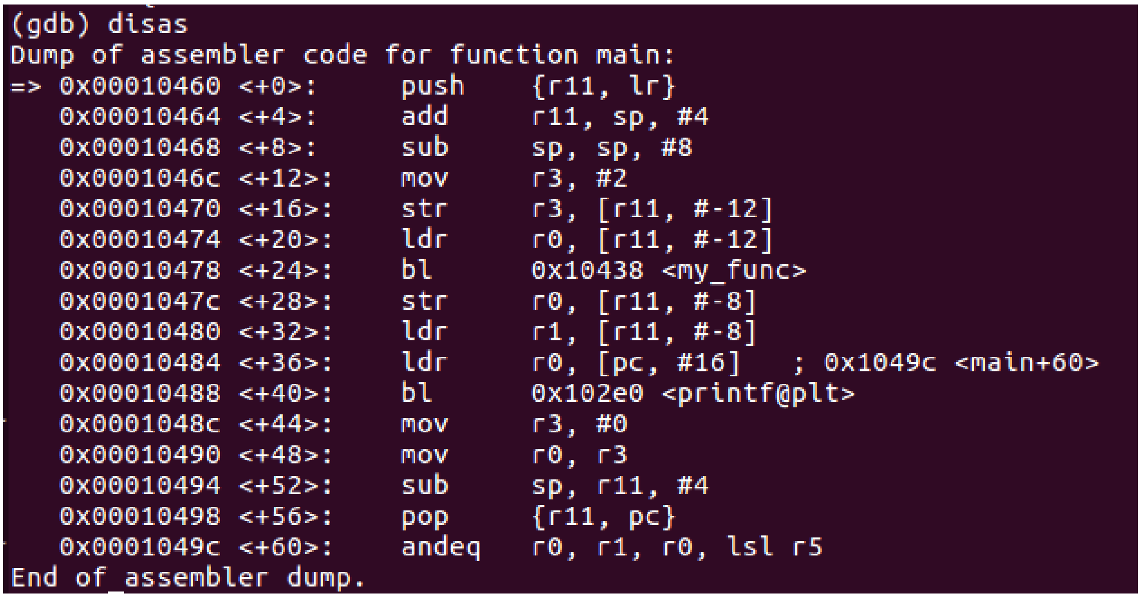
```
#include<stdio.h>

int my_func(int num)
{
        return num *2;
}

int main(void)
{
        int res, num=2;
        res=my_func(num);
        printf("res=%d\n",res);
        return 0;
}
```



```
(gdb) disas
Dump of assembler code for function main:
   0x00010460 <+0>:     push    {r11, lr}
   0x00010464 <+4>:     add     r11, sp, #4
   0x00010468 <+8>:     sub     sp, sp, #8
=> 0x0001046c <+12>:    mov     r3, #2
   0x00010470 <+16>:    str     r3, [r11, #-12]
   0x00010474 <+20>:    ldr     r0, [r11, #-12]
   0x00010478 <+24>:    bl      0x10438 <my_func>
   0x0001047c <+28>:    str     r0, [r11, #-8]
   0x00010480 <+32>:    ldr     r1, [r11, #-8]
   0x00010484 <+36>:    ldr     r0, [pc, #16]   ; 0x1049c <main+60>
   0x00010488 <+40>:    bl      0x102e0 <printf@plt>
   0x0001048c <+44>:    mov     r3, #0
   0x00010490 <+48>:    mov     r0, r3
   0x00010494 <+52>:    sub     sp, r11, #4
   0x00010498 <+56>:    pop     {r11, pc}
   0x0001049c <+60>:    andeq   r0, r1, r0, lsl r5
```

bl 이란 명령어는→ call 과 같은 역할을함

첫번째로 가기위해선
b *(주소)
b *0x00010460
--
gdb-multiarch
file a.out
tar rem localhost:1234
b *0x00010460
c



```
(gdb) disas
Dump of assembler code for function main:
=> 0x00010460 <+0>:     push    {r11, lr}
   0x00010464 <+4>:     add     r11, sp, #4
   0x00010468 <+8>:     sub     sp, sp, #8
   0x0001046c <+12>:    mov     r3, #2
   0x00010470 <+16>:    str     r3, [r11, #-12]
   0x00010474 <+20>:    ldr     r0, [r11, #-12]
   0x00010478 <+24>:    bl      0x10438 <my_func>
   0x0001047c <+28>:    str     r0, [r11, #-8]
   0x00010480 <+32>:    ldr     r1, [r11, #-8]
   0x00010484 <+36>:    ldr     r0, [pc, #16]   ; 0x1049c <main+60>
   0x00010488 <+40>:    bl      0x102e0 <printf@plt>
   0x0001048c <+44>:    mov     r3, #0
   0x00010490 <+48>:    mov     r0, r3
   0x00010494 <+52>:    sub     sp, r11, #4
   0x00010498 <+56>:    pop     {r11, pc}
   0x0001049c <+60>:    andeq   r0, r1, r0, lsl r5
End of assembler dump.
```

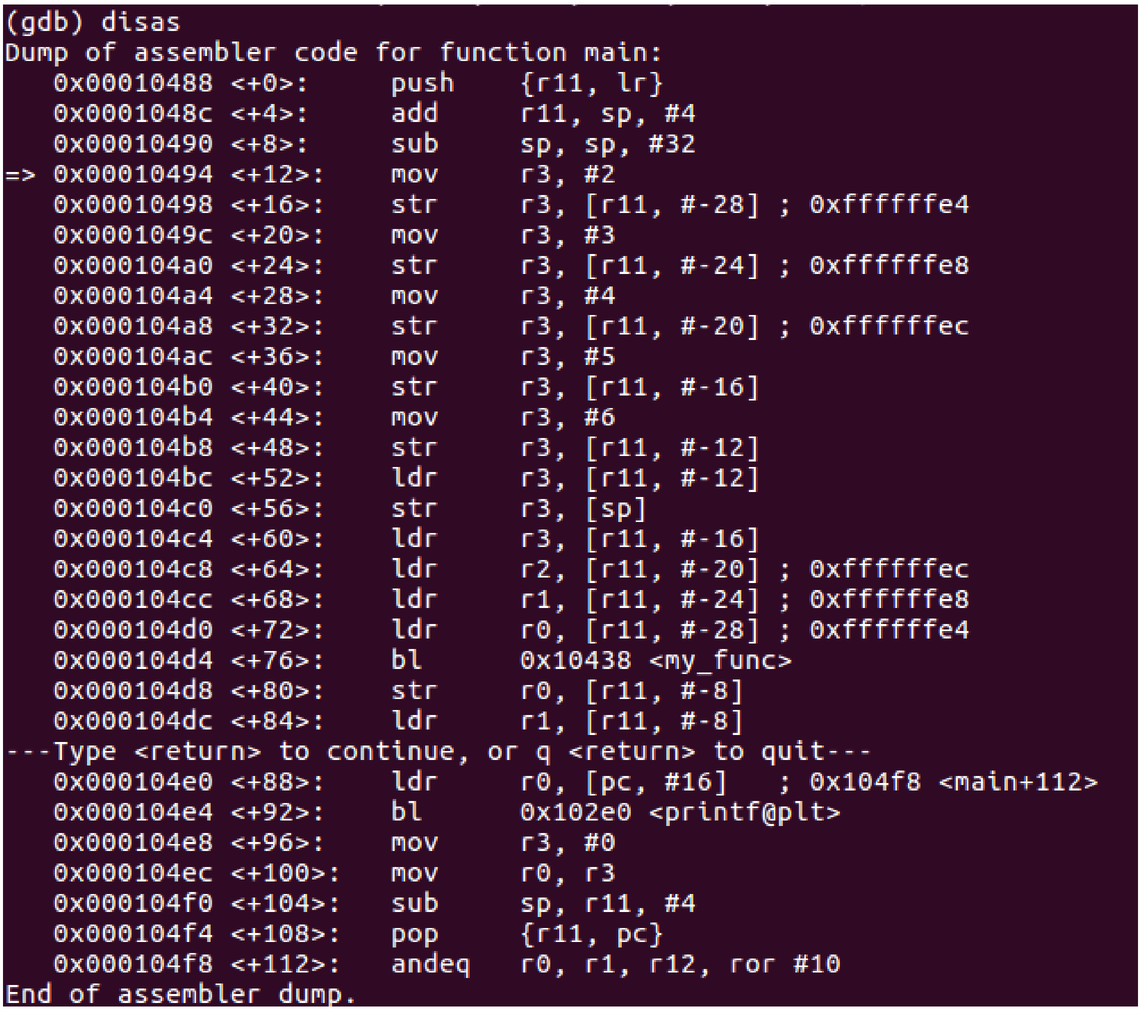
복귀주소를 lr 에 저장

//
인텔은 함수의 인자를 스택에 전달
암은 함수의 인자를 레지스터에 전달

arm_func2.c
#include<stdio.h>

int my_func(int n1, int n2, int n3, int n4, int n5)
{

```
        return n1+n2+n3+n4+n5;
}

int main(void)
{
        int res, n1=2, n2=3, n3=4, n4=5, n5=6;
        res=my_func(n1,n2,n3,n4,n5);
        printf("res=%d\n",res);
        return 0;
}
```



```
(gdb) disas
Dump of assembler code for function main:
   0x00010488 <+0>:     push    {r11, lr}
   0x0001048c <+4>:     add     r11, sp, #4
   0x00010490 <+8>:     sub     sp, sp, #32
=> 0x00010494 <+12>:    mov     r3, #2
   0x00010498 <+16>:    str     r3, [r11, #-28] ; 0xffffffe4
   0x0001049c <+20>:    mov     r3, #3
   0x000104a0 <+24>:    str     r3, [r11, #-24] ; 0xffffffe8
   0x000104a4 <+28>:    mov     r3, #4
   0x000104a8 <+32>:    str     r3, [r11, #-20] ; 0xffffffec
   0x000104ac <+36>:    mov     r3, #5
   0x000104b0 <+40>:    str     r3, [r11, #-16]
   0x000104b4 <+44>:    mov     r3, #6
   0x000104b8 <+48>:    str     r3, [r11, #-12]
   0x000104bc <+52>:    ldr     r3, [r11, #-12]
   0x000104c0 <+56>:    str     r3, [sp]
   0x000104c4 <+60>:    ldr     r3, [r11, #-16]
   0x000104c8 <+64>:    ldr     r2, [r11, #-20] ; 0xffffffec
   0x000104cc <+68>:    ldr     r1, [r11, #-24] ; 0xffffffe8
   0x000104d0 <+72>:    ldr     r0, [r11, #-28] ; 0xffffffe4
   0x000104d4 <+76>:    bl      0x10438 <my_func>
   0x000104d8 <+80>:    str     r0, [r11, #-8]
   0x000104dc <+84>:    ldr     r1, [r11, #-8]
---Type <return> to continue, or q <return> to quit---
   0x000104e0 <+88>:    ldr     r0, [pc, #16]   ; 0x104f8 <main+112>
   0x000104e4 <+92>:    bl      0x102e0 <printf@plt>
   0x000104e8 <+96>:    mov     r3, #0
   0x000104ec <+100>:   mov     r0, r3
   0x000104f0 <+104>:   sub     sp, r11, #4
   0x000104f4 <+108>:   pop     {r11, pc}
   0x000104f8 <+112>:   andeq   r0, r1, r12, ror #10
End of assembler dump.
```

레지스터연산은 1 클록에 끝나고 메모리클록은 수십클록이 걸릴수도있음

안에서 호출할때 파라미터는 레지스터로 전달되는데
단 4 개가넘어가면 스택을 쓰게된다

함수의 리턴값은 r0 에 저장된다
함수가 호출되고 r0 값을 보려고하면 함수의 리턴값이 보이게된다(주의)
함수에 인자전달하지않는이상 r0, r3 을 사용하지않는게 좋다