

I DSP,Xilinx zynq FPGA,MCU 및  
Xilinx

zynq FPGA 프로그래밍 전문가 과정

강사 -INNOVA LEE( 이상훈 )

[Gccompil3r@gmail.com](mailto:Gccompil3r@gmail.com)

학생 - 윤지완

[Yoonjw789 @naver.com](mailto:Yoonjw789@naver.com)

```

#include<stdio.h>
int main(void)
{
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;

```

```

asm volatile("mov r2,#3");

```

```

asm volatile("mov r3, #7");
asm volatile("mul r1,r2,r3");
printf("r1= %d\n",r1);

```

```

return 0;
}

```

```

ynjw375812@ynjw375812-L20NH-ASS1850:~$ demu-arm-static -L /usr/arm-linux-gnueabihf
./a.out
r1= 21

```

```

lr      0xf6686d14      -160928492
pc      0x10458  0x10458  <main+32>
cpsr    0x60000010      1610612752
(gdb) si
14      asm volatile("mov r3, #7");
(gdb) info reg
r0      0x0      0
r1      0x0      0
r2      0x3      3
r3      0x0      0
r4      0x0      0
r5      0x0      0
r6      0x10310  66320
r7      0x0      0
r8      0x0      0
r9      0x0      0
r10     0xf67fe000      -159391744
r11     0xf6ffef14      -150999276
r12     0xf6ffef90      -150999152
sp      0xf6ffef08      0xf6ffef08
lr      0xf6686d14      -160928492
pc      0x1045c  0x1045c  <main+36>
cpsr    0x60000010      1610612752
(gdb)

```

```

r1 0x10080d14 -160928492
c 0x1045c 0x1045c <main+36>
psr 0x60000010 1610612752
gdb) si
5 asm volatile("mul r1,r2,r3");
gdb) info reg
0 0x0 0
1 0x0 0
2 0x3 3
3 0x7 7
4 0x0 0
5 0x0 0
6 0x10310 66320
7 0x0 0
8 0x0 0
9 0x0 0
10 0xf67fe000 -159391744
11 0xf6ffef14 -150999276
12 0xf6ffef90 -150999152
p 0xf6ffef08 0xf6ffef08
r 0xf6686d14 -160928492
c 0x10460 0x10460 <main+40>
psr 0x60000010 1610612752
gdb)

```

```

(gdb) info reg
0 0x0 0
1 0x15 21
2 0x3 3
3 0x7 7
4 0x0 0
5 0x0 0
6 0x10310 66320
7 0x0 0
8 0x0 0
9 0x0 0
10 0xf67fe000 -159391744

```

```

#include<stdio.h>
int main(void)
{
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;

```

```

asm volatile("mov r2,#3");
asm volatile("mov r3,#7");
asm volatile("mov r4, #33");
asm volatile("mla r1,r2,r3,r4");
printf("r1= %d\n",r1);

```

```

return 0;
}

```

```

ynjw375812@ynjw375812-Z20NH-AS51B5U:~$ qemu-arm
./a.out
r1= 54

```

```

(gdb) info reg
r0                0x0          0
r1                0x0          0
r2                0x3          3
r3                0x0          0
r4                0x0          0
r5                0x0          0
r6                0x10310      66320
r7                0x0          0
r8                0x0          0
r9                0x0          0

```

```

(gdb) stepi
14      asm volatile("mov r4, #33");
(gdb) info reg
r0                0x0          0
r1                0x0          0
r2                0x3          3
r3                0x7          7
r4                0x0          0
r5                0x0          0

```

```

15      asm volatile("mla r1,r2,r3,r4");
(gdb) info reg
r0          0x0      0
r1          0x0      0
r2          0x3      3
r3          0x7      7
r4          0x21     33
r5          0x0      0

```

```

6      printf("r1= %d\n",r1);
(gdb) info reg
0          0x0      0
1          0x36     54
2          0x3      3
3          0x7      7
4          0x21     33
5          0x0      0

```

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

```
int main(void)
```

```
{
```

```
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;
```

```
asm volatile("mov r2,#0x44,8");
```

```
asm volatile("mov r3,#0x200");
```

```
//asm volatile("mov r4, #33");
```

```
asm volatile("umull r0,r1,r2,r3");
```

```
printf("r1r0= 0x%x%x\n",r1,r0);
```

```
return 0;
```

```
}
```

```
r2=0x4400 00 00 00
```

```
r3=0x      2 00
```

---

```
0x88000000000000
```

```
r1=상위 비트  r0=하위 비트
```

```
ynjw375812@ynjw375812-Z20NH-AS51B5U:~$ qemu-arm-static -L /usr/arm-linux-gnueabi  
./a.out  
r1r0= 0x880
```

**r2\*r3 결과값을 r0,r1 에 나눠서 넣어준다.32 비트를 넘어가기 때문에**

```
#include<stdio.h>  
int main(void)  
{  
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;
```

```
asm volatile("mov r2,#0x44,8");  
asm volatile("mov r3,#0x200");  
//asm volatile("mov r4, #33");  
asm volatile("umull r0,r1,r2,r3");
```

```
printf("r1r0= 0x%x %08x\n",r1,r0);
```

```
return 0;  
}
```

**뒷자리를 표현하기 위해 %08x 를 추가했다.**

```
#include<stdio.h>
int main(void)
{
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;
```

```
asm volatile("mov r0,#0xf");
asm volatile("mov r1,#0x1");
asm volatile("mov r2, #0x44,8");
asm volatile("mov r3,#0x200");
asm volatile("umlal r0,r1,r2,r3");
```

```
printf("r1r0= 0x%x%x\n",r1,r0);
```

```
return 0;
}
```

r1=상위 비트,r0=하위 비트

umlal:동시다발적으로 곱셈과 덧셈을 해준다.

```
ynjw375812@ynjw375812-Z20NH-AS51B5U:~$ qemu-arm-static -L /usr/arm-linux-gnue
./a.out
r1r0= 0x89f
```

```
#include<stdio.h>
unsigned int arr[5]={1,2,3,4,5};
```

```
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 unsigned int 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=arr;
```

```
//r1=arr[0]
```

```
asm volatile("mov r2, #0x8");
```

```
asm volatile("ldr r0,[r1,r2]");
```

```
printf("r0= %u\n",r0);
```

```
return 0;
```

```
}
```

```
ynjw375812@ynjw375812-ZZ0NH-ASS1B5U:~$ qemu-arm-static -L /usr/arm-linux-gnueabihf ./a.out
r0= 3
```

```
#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];
asm volatile("mov r0 , #61");
asm volatile("strb r0,[r1]");//레지스터에서 메모리로 집어넣는다(strb)

```

```

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

```

```

return 0;
}

```

```

ynjw375812@ynjw375812-Z20NH-AS51B5U:~$ qemu-arm-static -L /usr/arm-linux-gnueabi
./a.out
test= hello=RM

```

r1 에다가 test[5]에 있는 A 를 넣고 #61 번은 아스키코드로 보면 “=”이  
기때문에 STRB 를 통해 r0 에 있는 아스키코드 번호를 r1 에 넣고 test  
를 출력해보면 A 대신 “=”가 바뀐것을 알 수 있다.

```

#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]!");//레지스터에서 메모리로 집어넣는다
(strb)

```

```

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

```

```

return 0;
}

```

```

ynjw375812@ynjw375812-Z20NH-AS51B5U:~$ qemu-arm-static -L /usr/arm-linux-gnueab
./a.out
test= helloARM, r1=ARM

```

여기서 중요한것은 !이다.test 는 그대로 나오는것은 당연하나 왜 r1 이 ARM 이 출력이 되냐, 그 이유는 r2 는 0x5 까지 쉬프트를 동작하며 r2 의 0x5 쉬프트 된것을 r1 값을 fix 시킨다.

```

#include<stdio.h>

```

```
unsigned int arr[5]={1,2,3,4,5};
```

```
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 unsigned int * 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=arr;  
asm volatile("mov r2 , #0x4");  
asm volatile("ldr r0,[r1],r2");//레지스터에서 메모리로 집어넣는다  
(strb)
```

```
printf("r0= %u, r1=%u\n",r0,*r1);
```

```
return 0;  
}
```

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

```
int main(void)
```

```
{  
    int i;  
    unsigned int test_arr[7]={0};
```

```

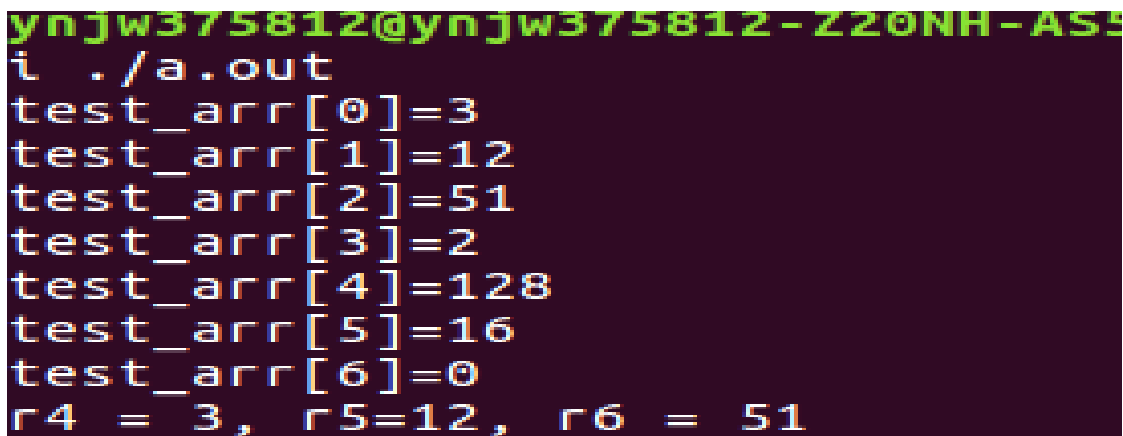
register unsigned int *r0 asm("r0")=0;
register unsigned int r1 asm("r1")=0;
register unsigned int r2 asm("r2")=0;
register unsigned int r3 asm("r3")=0;
register unsigned int r4 asm("r4")=0;
register unsigned int r5 asm("r5")=0;
register unsigned int r6 asm("r6")=0;

```

```

r0=test_arr;
asm volatile("mov r1 , #0x3\n"
             "mov r2,r1,lsr #2\n"
             "mov r4 , #0x2\n"
             "add r3,r1,r2,lsr 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;

```



A terminal window with a dark background and green text. The output of the program is displayed, showing the values of the test\_arr array and the register values r4, r5, and r6.

```

ynjw375812@ynjw375812-220NH-AS5
i ./a.out
test_arr[0]=3
test_arr[1]=12
test_arr[2]=51
test_arr[3]=2
test_arr[4]=128
test_arr[5]=16
test_arr[6]=0
r4 = 3, r5=12, r6 = 51

```

```

#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
End of assembler dump.

```

bl 의 복귀주소는 lr 에 저장한다.

arm 은 register 로 값을 저장하고 보내는 특징이 있다.

```
#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;
}
```

```
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] ; 0xffffffffe4
0x0001049c <+20>:     mov     r3, #3
0x000104a0 <+24>:     str     r3, [r11, #-24] ; 0xffffffffe8
0x000104a4 <+28>:     mov     r3, #4
0x000104a8 <+32>:     str     r3, [r11, #-20] ; 0xffffffffec
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] ; 0xffffffffec
0x000104cc <+68>:     ldr     r1, [r11, #-24] ; 0xffffffffe8
> 0x000104d0 <+72>:     ldr     r0, [r11, #-28] ; 0xffffffffe4
0x000104d4 <+76>:     bl      0x10438 <my_func>
0x000104d8 <+80>:     str     r0, [r11, #-8]
0x000104dc <+84>:     ldr     r1, [r11, #-8]
```

```
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
```

처음에 lr 은 bl 이 실행이 끝나고 복귀주소(0xf6ffef14)를 저장한다.

그리고 r11 에 sp+4bit 를 저장하고 sp 를 32bit 를 빼준다.

그리고 r3 레지스터를 이용해 각 n1~n5 까지의 값들을 4bit 간격으로 값을 넣어주고있다.

그리고 0x000104b8 <+48>: str r3, [r11, #-12]

0x000104bc <+52>: ldr r3, [r11, #-12]

이 부분은 쓸때없는 작업을 하는것으로 보여진다. 6을넣고 빼고 하는 쓸때없는 동작. 그리고 r3에 저장되어있던 6을 sp(stack pointer)에 저장을 한다. 이 작업을 하는것을 맨 마지막에 적어놨다. 그 다음 ldr을 이용해 메모리에 있던 2~5의 값들을 r0~r3까지의 레지스터에 차곡차곡 넣어두고 있다 왜 r0~r3까지만 사용하고 나머지 레지스터를 쓰지않는 이유는 arm 특유의 방식으로 인자를 4개까지만 레지스터의 저장하고 나머지 인자들은 sp에 저장하는 방식이다. 이 방식은 intel과 차이점이 있다.이 동작이 끝나고 bl을 통해서 res=my\_func(n1,n2,n3,n4,n5)가 동작을 하고 아래 부분으로 들어간다.

```
Dump of assembler code for function my_func:
=> 0x00010438 <+0>:      push    {r11}                ; (str r11, [sp, #-4]!)
    0x0001043c <+4>:      add     r11, sp, #0
    0x00010440 <+8>:      sub     sp, sp, #20
    0x00010444 <+12>:     str     r0, [r11, #-8]
    0x00010448 <+16>:     str     r1, [r11, #-12]
    0x0001044c <+20>:     str     r2, [r11, #-16]
    0x00010450 <+24>:     str     r3, [r11, #-20] ; 0xffffffffec
    0x00010454 <+28>:     ldr     r2, [r11, #-8]
    0x00010458 <+32>:     ldr     r3, [r11, #-12]
    0x0001045c <+36>:     add     r2, r2, r3
    0x00010460 <+40>:     ldr     r3, [r11, #-16]
    0x00010464 <+44>:     add     r2, r2, r3
    0x00010468 <+48>:     ldr     r3, [r11, #-20] ; 0xffffffffec
    0x0001046c <+52>:     add     r2, r2, r3
    0x00010470 <+56>:     ldr     r3, [r11, #4]
    0x00010474 <+60>:     add     r3, r2, r3
    0x00010478 <+64>:     mov     r0, r3
    0x0001047c <+68>:     sub     sp, r11, #0
    0x00010480 <+72>:     pop     {r11}                ; (ldr r11, [sp], #4)
    0x00010484 <+76>:     bx     lr
```

<bl my\_func 내부의 모습>

이 코드에서 중요한 부분은 r0~r3까지는 인자를 받는 레지스터들이다. 그래서 데이터를 받을때 저장할 때 이것들로만 사용을 했지만 숫자의 갯수가 4개가 넘어가니 sp(stack pointer)에다가 저장하는 모습을 보여주고 있다.함수의 리턴값은 언제나 r0에 저장이 된다.데이터를 주거나 받거나 하지 않을때는 r4~r6까지주로 사용 하면된다.r7는 system call 역할이다.

