Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

#44

강사 : Innova Lee(이 상훈)

학생 : 김 시윤

1.배운내용 복습.

ARM 입문

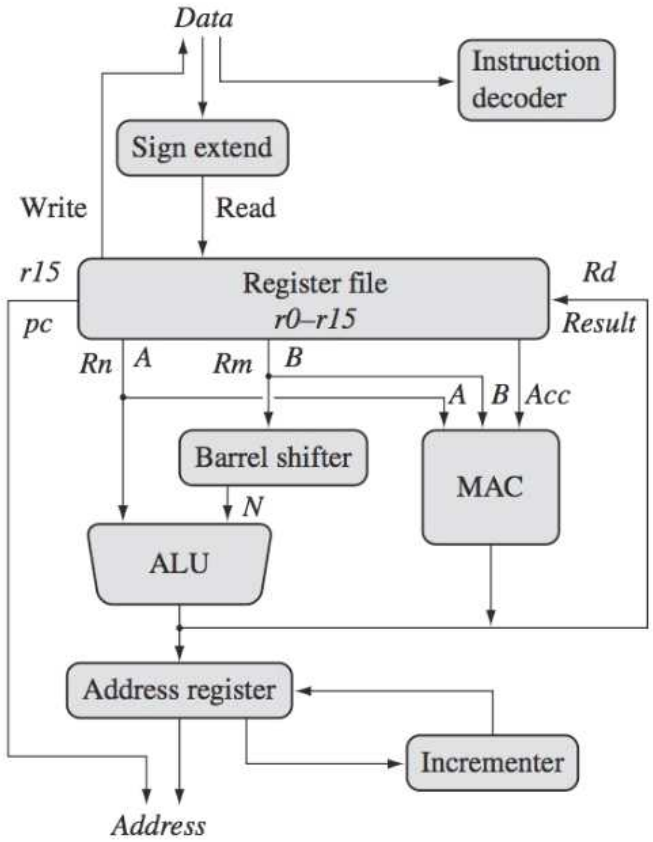

그림1. ARM 구조

Data Bus 는 데이터를 가져오고 Address Bus 는 주소를 가져옴 즉 address Bus 는 Data의 위치를 알려주고 그 위치에 Data Bus가 Data를 갖다 준다. 맥은 곱셈기와 뎃셈기가 합쳐져있다. 맥은 곱셈처리 2개를 1clock에 해결 가능하며, 병렬처리도 4개까지 가능하다.

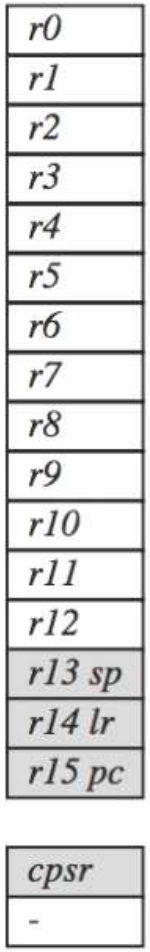

그림2. User모드에서 사용가능한 ARM Register

ARM은 r0 ~ r15 까지 총 16개 레지스터를 갖고 있다.

r13 = sp

r14 = link register 함수 호출할때 복귀주소는 여기 레지스터에 저장

r15 = program counter instruction address 저장

cpsr = current program status register – flag를 통해 현재 current 상태를 나타냄

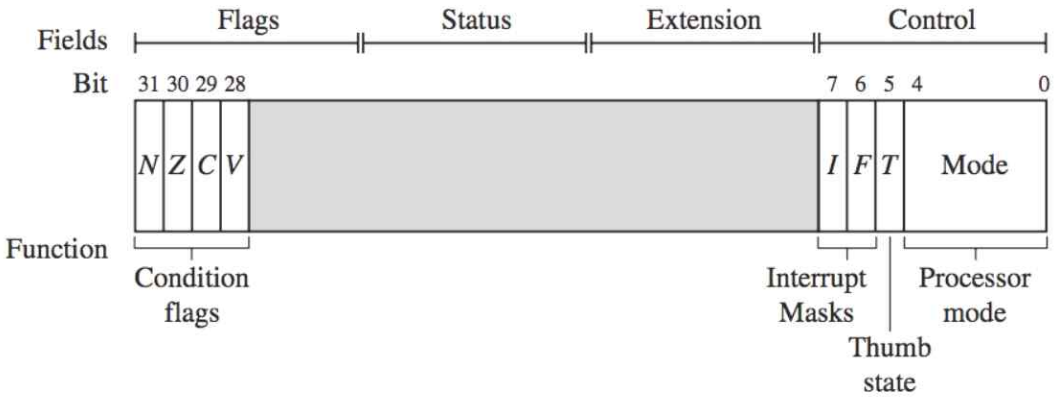

그림3. cpsr

Flag	Flag name	Set when
Q	Saturation	the result causes an overflow and/or saturation
V	oVerflow	the result causes a signed overflow
C	Carry	the result causes an unsigned carry
Z	Zero	the result is zero, frequently used to indicate equality
N	Negative	bit 31 of the result is a binary 1

그림4. status flag

Suffix	Description	flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 | Z=1
GE	Greater or equal	N = V
LT	Less than	N != V
GT	Greater than	Z=0 & N = V
LE	Less than or equal	Z=1 | N=!V
AL	Always	

ARM user 영역의 레지스터를 다루기 위해 우리는 ARM 어셈블리어를 이용하여 디버깅으로 확인하였다.

add.c

```
#include <stdio.h>
int main(void)
{       register unsigned int r0 asm("r0");   //r0변수에 레지스터 r0
        register unsigned int r1 asm("r1");   //r1변수에 레지스터 r1
        register unsigned int r2 asm("r2");   //r2변수에 레지스터 r2
        r1 = 77;
        r2 = 37;

        asm volatile("add r0,r1,r2");   //r0에 r1과 r2를 더해 저장한다.
        printf("r0 = %d\n",r0);
        return 0;
}
```

r1 레지스터에 77저장
r2 레지스터에 37 저장
위 식데로라면 114가 출력되어야 함.

```
siyun@siyun-CR62-6M:~/my_proj/44$ arm-linux-gnueabi-gcc -g add.c
siyun@siyun-CR62-6M:~/my_proj/44$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0 = 114
siyun@siyun-CR62-6M:~/my_proj/44$
```

bic.c

```
#include <stdio.h>
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");
        register unsigned int r1 asm("r1");
        register unsigned int r2 asm("r2");
        register unsigned int r3 asm("r3");
        register unsigned int r4 asm("r4");
        register unsigned int r5 asm("r5");
        r0 = 7;
        r1 = 7;

        if(r0 == r1)
        {
                r3 = 42;
                asm volatile("biceq r2,r3,#7");
                /*여기서 eq 는 Z가 1일 때 동작한다
                if 문을 사용하지 않고 조건을 달수 있는
                arm의 좋은 기능이다. */
        }
                show_reg(r2);
        return 0;
}
```

n 승 단위로 클리어 = 2^n -1
즉 7이면 8 단위로 클리어 시켜라.

여기서 r3은 42
r3을 8의 단위로 나열하면 r2의 값은 40이 된다.

```
siyun@siyun-CR62-6M:~/my_proj/44$ arm-linux-gnueabi-gcc -g bic.c
siyun@siyun-CR62-6M:~/my_proj/44$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
00000000000000000000000000101000
siyun@siyun-CR62-6M:~/my_proj/44$
```

sub.c

```
#include <stdio.h>
int main(void)
{       register unsigned int r0 asm("r0");
        register unsigned int r1 asm("r1");
        register unsigned int r2 asm("r2");
        register unsigned int r3 asm("r3");
        r1 = 77;
        r2 = 37;
        r3 = 34;
        if(r1 > r2)
                asm volatile("subgt r3,r3,#1");
                /* Z=0 & N=V 일 때 동작
                N =0 V=0 Z = 0 이므로 동작한다 */
        printf("r3 = %d\n",r3);
return 0;
}
```

```
siyun@siyun-CR62-6M:~/my_proj/44$ arm-linux-gnueabi-gcc -g subgt.c
siyun@siyun-CR62-6M:~/my_proj/44$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r3 = 33
siyun@siyun-CR62-6M:~/my_proj/44$
```

rsble.c

```
#include <stdio.h>
int main(void)
{       register unsigned int r0 asm("r0");
        register unsigned int r1 asm("r1");
        register unsigned int r2 asm("r2");
        register unsigned int r3 asm("r3");
        register unsigned int r4 asm("r4");
        register unsigned int r5 asm("r5");
        r1 = 77;
        r2 = 37;
        r3 = 34;
        r5 = 3;
        if(r2 <= r1)
                asm volatile("rsble r4, r5, #5");
                /* Z=1 or N !=V 일 때 동작
                sub를 하고 반전을 시킨다 sub를 했을 때
                overflow가 나서 N != V 가 참이되어 동작하는거 같다 */
        printf("r4 = %d\n", r4);
        return 0;
}
```

```
siyun@siyun-CR62-6M:~/my_proj/44$ arm-linux-gnueabi-gcc -g rsble.c
siyun@siyun-CR62-6M:~/my_proj/44$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r4 = 2
siyun@siyun-CR62-6M:~/my_proj/44$
```

/*나중에 디버깅 해보도록 한다 */

and.c

```
#include <stdio.h>
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");
        register unsigned int r1 asm("r1");
        register unsigned int r2 asm("r2");
        register unsigned int r3 asm("r3");
        register unsigned int r4 asm("r4");
        register unsigned int r5 asm("r5");
        r1 = 34;
        r2 = 37;
        r5 = 3;
        asm volatile("and r0,r1,r2");
        show_reg(r0);
        return 0;
}
```

r0 = r1 & r2;
34 & 37
100010 & 100101 = 100000

```
siyun@siyun-CR62-6M:~/my_proj/44$ arm-linux-gnueabi-gcc -g and.c
siyun@siyun-CR62-6M:~/my_proj/44$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
00000000000000000000000000100000
siyun@siyun-CR62-6M:~/my_proj/44$
```

eor.c

```
#include <stdio.h>
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")=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;
        if(r0 == r1)
        {
                r0= 10;
                r3= 5;
                asm volatile("eors r1,r3,r0");
                /* 우리가 알고있는 XOR 연산 정식 명칭은
                EXCLUSSIVE OR 줄여서 eor로 나타냈다.
                r3 과 r0을 eor 하여 r1에 저장한다.*/
        }
        show_reg(r1);
        return 0;
}
```

r3 = 0101
r0 = 1010
r3 ^ r0 = 1111
show_reg 는 비트단위로 나타내기위한 로직

```
siyun@siyun-CR62-6M:~/my_proj/44$ arm-linux-gnueabi-gcc -g eor.c
siyun@siyun-CR62-6M:~/my_proj/44$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
00000000000000000000000000001111
siyun@siyun-CR62-6M:~/my_proj/44$
```

orr.c

```
#include <stdio.h>
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")=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;
        r5 = 3;
        if(r0 == r1)
        {
                r3= 44;
                asm volatile("orr r2,r3,r5");
        }
        show_reg(r2);
        return 0;
}
```

orr 은 or 연산이다.
r5 =3 = 000011
r3 = 44 = 101100
r3 or r5 = 101111
이 출력된다.

mvn.c

```
#include <stdio.h>

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")=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("cmp r0,r1");
        /* CMP = operand1 - operand2; Compare
        즉 r0 - r1 = 0
        status flag 의 Z가 1로 셋된다. */
        asm volatile("mvneq r1,#0");
        /*mov negative 가 cmp에 의해 동작된다.
        r1 에 0이 들어가고 그걸 반전시킨다.
        그러면 모든 비트가 1이 된다. */

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

```
        return 0;
}
```

```
siyun@siyun-CR62-6M:~/my_proj/44$ arm-linux-gnueabi-gcc -g mvn.c
siyun@siyun-CR62-6M:~/my_proj/44$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r1 = 0xffffffff
siyun@siyun-CR62-6M:~/my_proj/44$
```

mov_cmp.c

```
#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("cmp r0,r1");
        asm volatile("mov r2,#5");
        asm volatile("cmp r0,r2");

        return 0;
}
//cmp 할 때 cpsr 값이 바뀜 이유는 status flag 의 Z 의 셋팅이 바뀌어서이다.
```

```
   0x00010434 <+52>:    sub     sp, r11, #8
   0x00010438 <+56>:    pop     {r4, r5, r11}
   0x0001043c <+60>:    bx      lr
End of assembler dump.
(gdb) info register
r0             0x0      0
r1             0x0      0
r2             0x5      5
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x102d8  66264
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef04       -150999292
r12            0xf6ffef80       -150999168
sp             0xf6ffeefc       0xf6ffeefc
lr             0xf6686d14       -160928492
pc             0x10428  0x10428 <main+40>
cpsr           0x60000010       1610612752
(gdb) s
17              return 0;
(gdb) disas
Dump of assembler code for function main:
   0x00010400 <+0>:     push    {r4, r5, r11}
   0x00010404 <+4>:     add     r11, sp, #8
   0x00010408 <+8>:     mov     r0, #0
   0x0001040c <+12>:    mov     r1, #0
   0x00010410 <+16>:    mov     r2, #0
   0x00010414 <+20>:    mov     r3, #0
   0x00010418 <+24>:    mov     r4, #0
   0x0001041c <+28>:    mov     r5, #0
   0x00010420 <+32>:    cmp     r0, r1
   0x00010424 <+36>:    mov     r2, #5
   0x00010428 <+40>:    cmp     r0, r2
=> 0x0001042c <+44>:    mov     r3, #0
   0x00010430 <+48>:    mov     r0, r3
   0x00010434 <+52>:    sub     sp, r11, #8
   0x00010438 <+56>:    pop     {r4, r5, r11}
   0x0001043c <+60>:    bx      lr
End of assembler dump.
(gdb) info register
r0             0x0      0
r1             0x0      0
r2             0x5      5
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x102d8  66264
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef04       -150999292
r12            0xf6ffef80       -150999168
sp             0xf6ffeefc       0xf6ffeefc
lr             0xf6686d14       -160928492
pc             0x1042c  0x1042c <main+44>
cpsr           0x80000010       -2147483632
(gdb)
```

CMP = operand1 - operand2

즉 여기서는 cmp = r0 -r2

0 - 5 = -5

Z bit 는 0으로 셋팅

N 비트는 1로 셋팅(31번째 부호비트 1이기 때문)

따라서 6 -> 8 로 cpsr 이 변하는걸 확인

```
#include <stdio.h>
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")=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("cmp r0,r1");
        asm volatile("mov r2,#3");
        asm volatile("tsteq r2,#5");

        return 0;
}
```

```
pc             0x104f0  0x104f0 <main+36>
cpsr           0x60000010       1610612752
(gdb) disas
Dump of assembler code for function main:
   0x000104cc <+0>:     push    {r4, r5, r11}
   0x000104d0 <+4>:     add     r11, sp, #8
   0x000104d4 <+8>:     mov     r0, #0
   0x000104d8 <+12>:    mov     r1, #0
   0x000104dc <+16>:    mov     r2, #0
   0x000104e0 <+20>:    mov     r3, #0
   0x000104e4 <+24>:    mov     r4, #0
   0x000104e8 <+28>:    mov     r5, #0
   0x000104ec <+32>:    cmp     r0, r1
=> 0x000104f0 <+36>:    mov     r2, #3
   0x000104f4 <+40>:    tsteq   r2, #5
   0x000104f8 <+44>:    mov     r3, #0
   0x000104fc <+48>:    mov     r0, r3
   0x00010500 <+52>:    sub     sp, r11, #8
   0x00010504 <+56>:    pop     {r4, r5, r11}
   0x00010508 <+60>:    bx      lr
End of assembler dump.
(gdb) s
23              asm volatile("tsteq r2,#5");
(gdb) info register
r0             0x0      0
r1             0x0      0
r2             0x3      3
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x10340  66368
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef04       -150999292
r12            0xf6ffef80       -150999168
sp             0xf6ffeefc       0xf6ffeefc
lr             0xf6686d14       -160928492
pc             0x104f4  0x104f4 <main+40>
cpsr           0x60000010       1610612752
(gdb) s
25              return 0;
(gdb) info register
r0             0x0      0
r1             0x0      0
r2             0x3      3
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x10340  66368
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef04       -150999292
r12            0xf6ffef80       -150999168
sp             0xf6ffeefc       0xf6ffeefc
lr             0xf6686d14       -160928492
pc             0x104f8  0x104f8 <main+44>
cpsr           0x20000010       536870928
(gdb)
```

TST = operand1 AND operand2
cmp r0 , r1을 하게 되면 0 - 0 이므로 status flag의 Z 는 1로 셋팅
따라서 6이다
r2에 3을 넣고 r2와 5를 and 연산한다.

011 & 101 = 001 이되어 status flag 의 Z 비트는 0으로 비활성화 되고
앞에 네자리 비트를 읽으면 2가 된다