TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/4/30
수업일수	44 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

- asm 디버깅 실행 순서
- ARM Processor
- DSP 와 MAC
- 범용 레지스터 General Register
    - cpsr Register
    - Processor Mode
    - Status Flag
- asm 명령어

asm 디버깅 실행 순서

(사전설치)

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

arm-linux-gnueabi-gcc -g 소스파일

sudo apt-get install gdb-multiarch
```

(gcc 및 결과 값 출력)

터미널을 2개 띄운다.

A 터미널에서 아래 명령어를 수행한다.

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

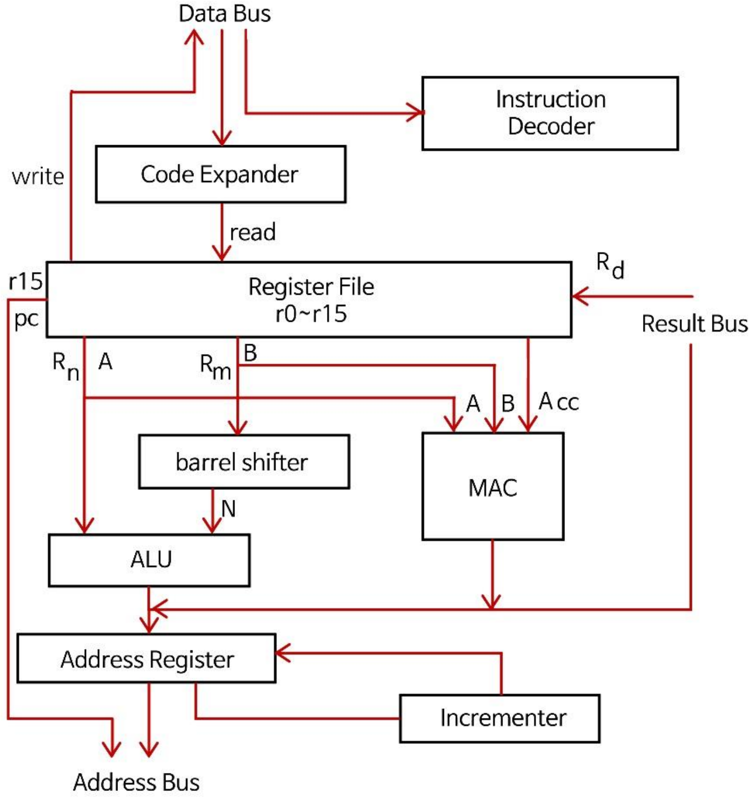
(gdb 실행 순서)

```
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
```

ARM Processer



data bus -데이터를 가져옴.

address bus - 해당 데이터가 메모리에 저장되어야 하는데 address bus 가 메모리를 지정하고 데이터를 보냄,

register file

instruction decoder

barrel shifter - 한 개의 연산으로 데이터 워드 내의 다수의 비트를 이동하거나 회전 시킬 수 있는 하드웨어 장치

ALU - 산술연산, 논리연산 등을 수행

MAC - 곱셈기

DSP 와 MAC

보편적으로 ARM 은 MAC 이 없고, MAC 이 있으면 DSP 이다. MAC 이 있으면 연산 클럭 수를 줄여 효율적이게 된다. 곱셈하는데 수~수십 클럭을 사용하게 되는데 MAC 을 통하여 1클록 만에 곱셈 연산을 끝낼 수 있어 효율적이다.

예를 들어 $\sin(x) * e^{iy}$를 연산한다 할 때 곱하는 각 요소들은 아날로그 함수여서 디지털 처리하는 컴퓨터는 동작주파수 f 를 이용해 샘플링을 하여 아날로그적 신호를 근시적인 디지털 신호로 나타내게 됨. 아두이노만 해도 50MHz 의 샘플링 속도를 가지는데 이를 데이터로 처리하면 매우 많은 데이터양을 pc 가 처리하기엔 너무 벅차고 MAC 이 이를 적은 클럭 수로 병렬처리를 통해 간단히 계산 가능하다.

범용 레지스터 General Register

ARM 에서의 범용 레지스터는 16 개이다. r0~r15(Data Register), cpsr 와 spsr 레지스터(Program Status Register)가 존재. 한번에 최대 18 개의 레지스터가 활성화될 수 있음.

r13 - Stack Pointer(sp), Processor Mode 의 Stack 맨 위 Address Value 를 저장.

r14 - Link Register(lr), Core 가 slub routine 을 호출할 때 마다 그 return address 를 저장.

r15 - Program Counter(pc), Processor 가 읽어들인 다음 Instruction 의 address 를 저장.

cpsr - Current Program Status Register

spsr - Saved Program Status Register

cpsr Register

ARM Core 는 내부 동작을 monitoring 하고 제어하기 위해 cpsr 을 사용. cpsr 은 32bit Rsgister 로 Register File 안에 위치해 있고, 다음은 일반적인 Program Status Register 의 Basic Layout 을 보여줌.

<table>
<tr><td colspan="5">Flag</td><td>Status</td><td>Extension</td><td colspan="4">Control</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td colspan="3" rowspan="2">확장을 위해 reserved 되어있는 부분
최근의 cpsr 레지스터는 꽉 채워져 있게 나옴</td><td>7</td><td>6</td><td>5</td><td>4 0
mode</td></tr>
<tr><td colspan="4">Status Flag</td><td colspan="2">Interrupt Mask</td><td>Thumb status</td><td>Processor Mode</td></tr>
</table>

control 필드에 Processor mode 와 interrupt mask 가 포함 되어있고 flag 필드에는 status flag 가 저장되어 있다.

Processor Mode

어떤 레지스터가 활성화되고 cpsr 레지스터를 access 할 수 있는 권리를 갖게 될지 결정함. 각 Privilege Mode 일 수도 있고 일반 Mode 일 수도 있다.

Mode	Abbreviation	Privilege Function	Mode Bit
Abort	Abt	yes	10111
Fast Interrupt request	Fiq	y	10001
Interrupt request	Irq	y	10010
Supervisor	Svc	y	10011
System	Sys	y	11111
Undefined	Und	y	11011
User	usr	n	10000

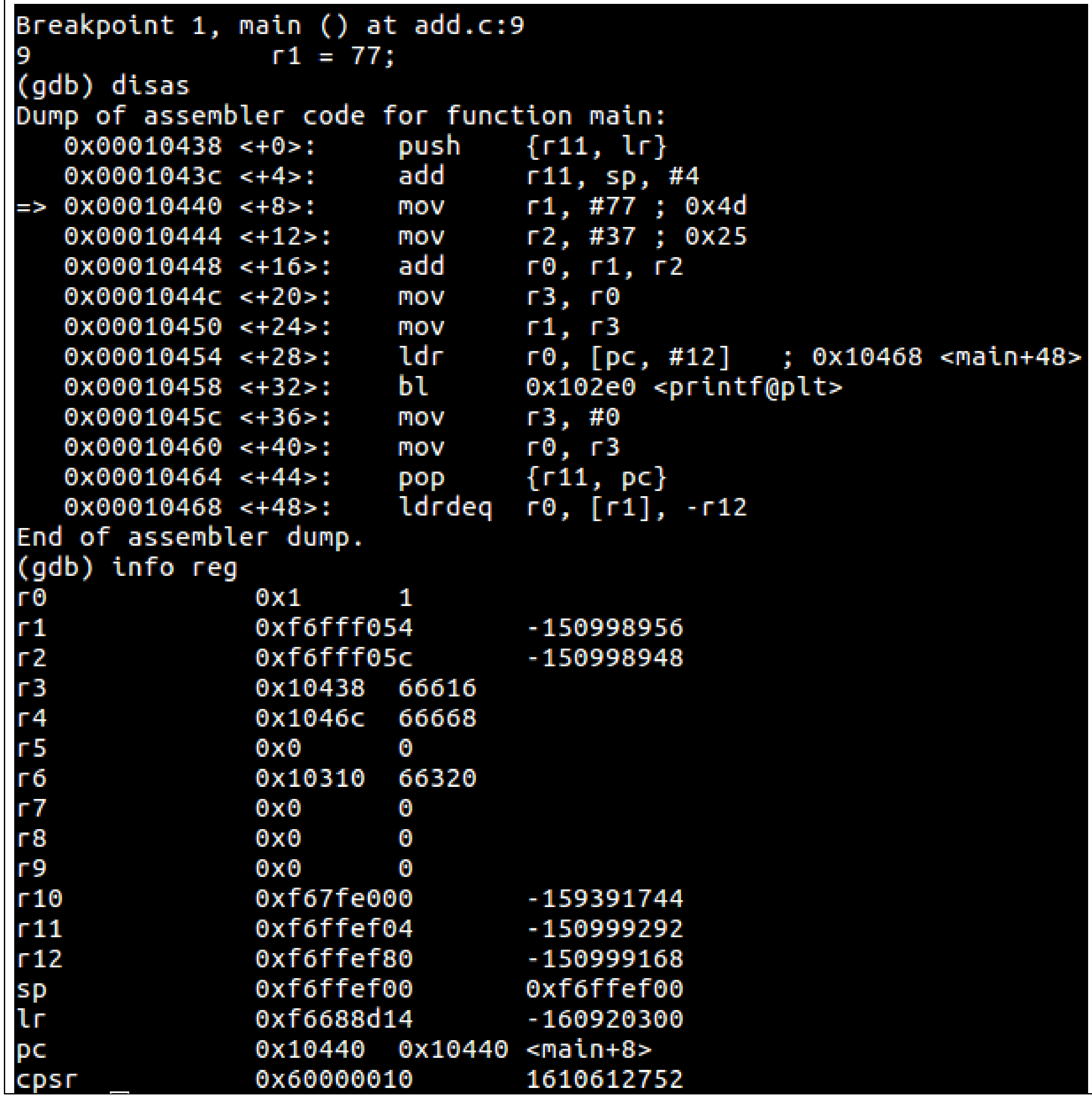
Status Flag

Flag	Flag name	1 로 set 되는 경우
Q	Saturation	Overflow 나 saturation 이 발생하는 경우
V	Overflow	Signed 나 Overflow 가 발생하는 경우
C	Carry	Unsigned Carry 가 발생하는 경우
Z	Zero	Result 가 0 인 경우, 종종 동일함을 표시하기 위해 사용
N	Negative	Result 의 31 번째 bit 가 1 인 경우

add

```
[add.c]
#include<stdio.h>

int main(void)
{
	register unsigned int r0 asm("r0"); //r0라는 변수명으로 진짜 레지스터 r0 를 사용할 것 이다.
	register unsigned int r1 asm("r1");
	register unsigned int r2 asm("r2");

	r1 = 77;
	r2 = 37;

	asm volatile("add r0, r1, r2"); //r1과 r2를 더한 값을 r0에 넣는다.

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

```
~결과
xeno@xeno-NH:~/proj/0430$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0 = 114
```



```
Breakpoint 1, main () at add.c:9
9               r1 = 77;
(gdb) disas
Dump of assembler code for function main:
   0x00010438 <+0>:     push    {r11, lr}
   0x0001043c <+4>:     add     r11, sp, #4
=> 0x00010440 <+8>:     mov     r1, #77 ; 0x4d
   0x00010444 <+12>:    mov     r2, #37 ; 0x25
   0x00010448 <+16>:    add     r0, r1, r2
   0x0001044c <+20>:    mov     r3, r0
   0x00010450 <+24>:    mov     r1, r3
   0x00010454 <+28>:    ldr     r0, [pc, #12]   ; 0x10468 <main+48>
   0x00010458 <+32>:    bl      0x102e0 <printf@plt>
   0x0001045c <+36>:    mov     r3, #0
   0x00010460 <+40>:    mov     r0, r3
   0x00010464 <+44>:    pop     {r11, pc}
   0x00010468 <+48>:    ldrdeq  r0, [r1], -r12
End of assembler dump.
(gdb) info reg
r0             0x1      1
r1             0xf6fff054       -150998956
r2             0xf6fff05c       -150998948
r3             0x10438  66616
r4             0x1046c  66668
r5             0x0      0
r6             0x10310  66320
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef04       -150999292
r12            0xf6ffef80       -150999168
sp             0xf6ffef00       0xf6ffef00
lr             0xf6688d14       -160920300
pc             0x10440  0x10440 <main+8>
cpsr           0x60000010       1610612752
```

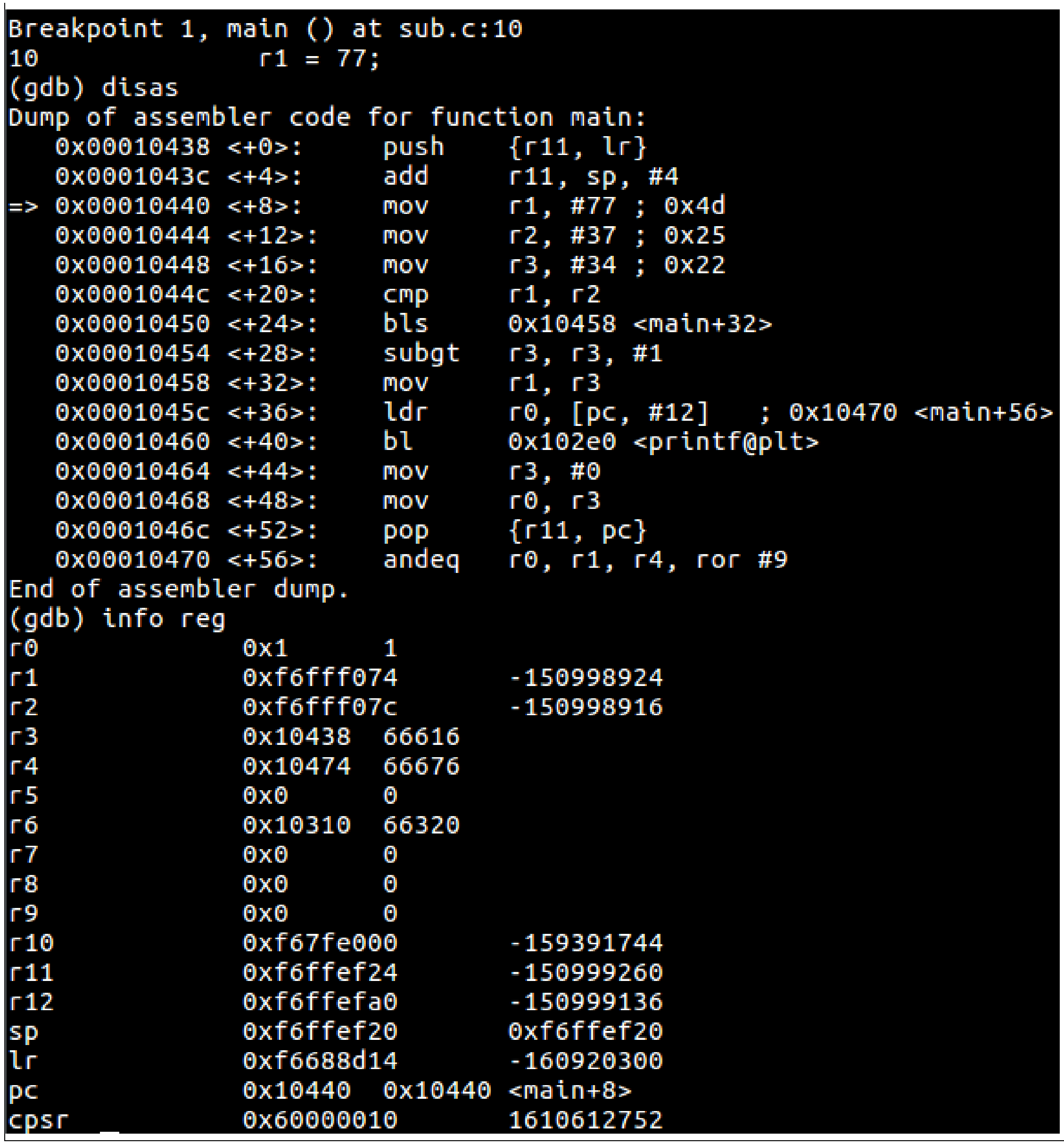
subgt

[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");

        printf("r3 = %d₩n",r3);
        return 0;

}
```

~결과

```
xeno@xeno-NH:~/proj/0430$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r3 = 33
```



```
Breakpoint 1, main () at sub.c:10
10                r1 = 77;
(gdb) disas
Dump of assembler code for function main:
   0x00010438 <+0>:     push    {r11, lr}
   0x0001043c <+4>:     add     r11, sp, #4
=> 0x00010440 <+8>:     mov     r1, #77 ; 0x4d
   0x00010444 <+12>:    mov     r2, #37 ; 0x25
   0x00010448 <+16>:    mov     r3, #34 ; 0x22
   0x0001044c <+20>:    cmp     r1, r2
   0x00010450 <+24>:    bls     0x10458 <main+32>
   0x00010454 <+28>:    subgt   r3, r3, #1
   0x00010458 <+32>:    mov     r1, r3
   0x0001045c <+36>:    ldr     r0, [pc, #12]   ; 0x10470 <main+56>
   0x00010460 <+40>:    bl      0x102e0 <printf@plt>
   0x00010464 <+44>:    mov     r3, #0
   0x00010468 <+48>:    mov     r0, r3
   0x0001046c <+52>:    pop     {r11, pc}
   0x00010470 <+56>:    andeq   r0, r1, r4, ror #9
End of assembler dump.
(gdb) info reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x10438  66616
r4             0x10474  66676
r5             0x0      0
r6             0x10310  66320
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef20       0xf6ffef20
lr             0xf6688d14       -160920300
pc             0x10440  0x10440 <main+8>
cpsr           0x60000010       1610612752
```

r1과 r2를 비교. r1 - r2해서 그 값을 보고 r1이 큰지 r2가 큰지 파악함. if 문을 만족하지 않으면 다음 명령어를 실행하고 만족한다면 다음 명령어를 실행하지 않음.

Sub 는 뺄셈연산을 의미하고 gt 는 greater than 을 의미.

gt 는 Z=

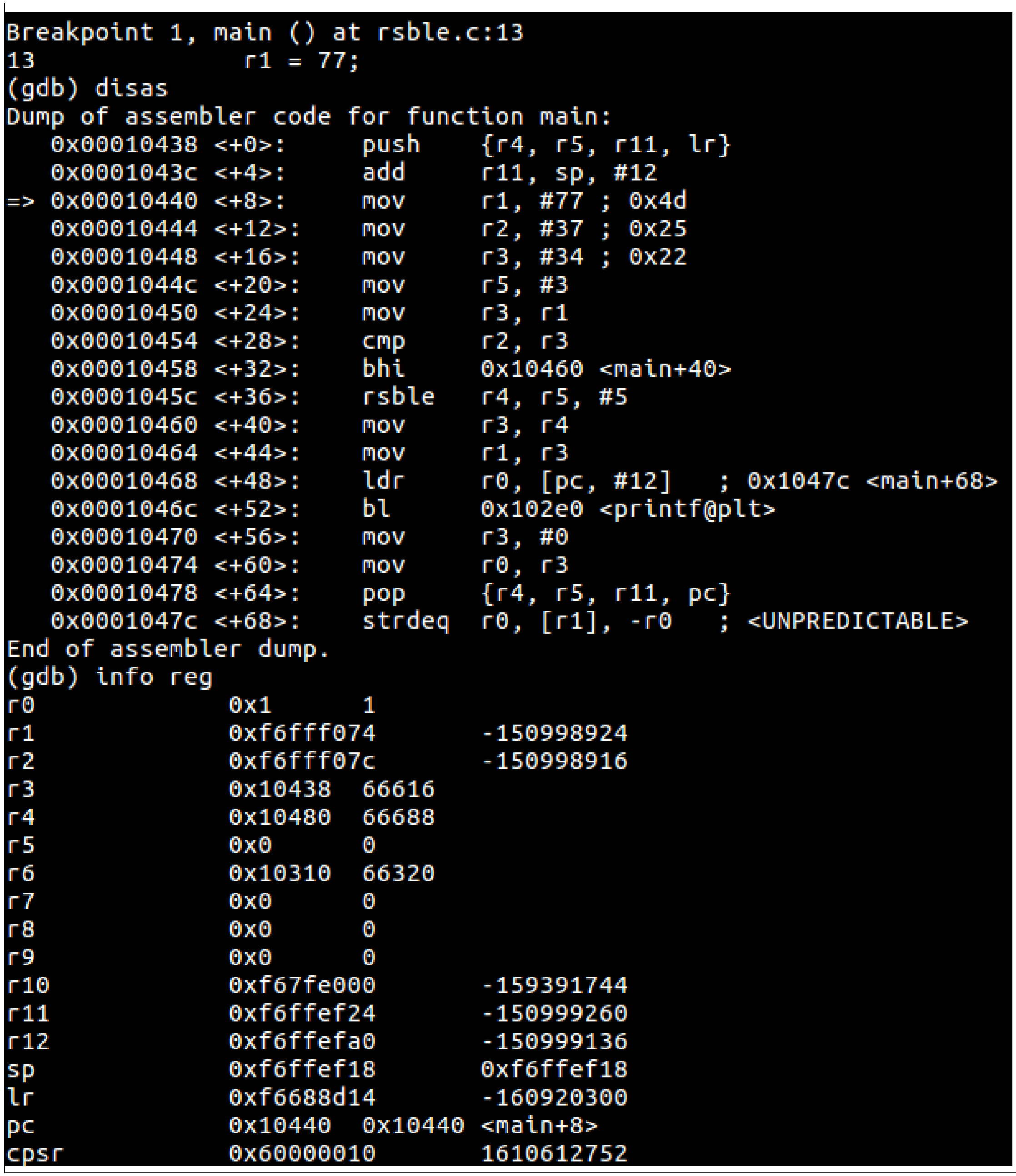
R3에 저장된 값 0x22와 1을 비교하여 더 큰 값을 r3에 저장함.

rsble

: Reverse SuB Less than or Equal

```
[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");

        printf("r4 = %d₩n", r4);
        return 0;
}
```

```
xeno@xeno-NH:~/proj/0430$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r4 = 2
```



```
Breakpoint 1, main () at rsble.c:13
13                r1 = 77;
(gdb) disas
Dump of assembler code for function main:
   0x00010438 <+0>:     push    {r4, r5, r11, lr}
   0x0001043c <+4>:     add     r11, sp, #12
=> 0x00010440 <+8>:     mov     r1, #77 ; 0x4d
   0x00010444 <+12>:    mov     r2, #37 ; 0x25
   0x00010448 <+16>:    mov     r3, #34 ; 0x22
   0x0001044c <+20>:    mov     r5, #3
   0x00010450 <+24>:    mov     r3, r1
   0x00010454 <+28>:    cmp     r2, r3
   0x00010458 <+32>:    bhi     0x10460 <main+40>
   0x0001045c <+36>:    rsble   r4, r5, #5
   0x00010460 <+40>:    mov     r3, r4
   0x00010464 <+44>:    mov     r1, r3
   0x00010468 <+48>:    ldr     r0, [pc, #12]   ; 0x1047c <main+68>
   0x0001046c <+52>:    bl      0x102e0 <printf@plt>
   0x00010470 <+56>:    mov     r3, #0
   0x00010474 <+60>:    mov     r0, r3
   0x00010478 <+64>:    pop     {r4, r5, r11, pc}
   0x0001047c <+68>:    strdeq  r0, [r1], -r0   ; <UNPREDICTABLE>
End of assembler dump.
(gdb) info reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x10438  66616
r4             0x10480  66688
r5             0x0      0
r6             0x10310  66320
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef18       0xf6ffef18
lr             0xf6688d14       -160920300
pc             0x10440  0x10440 <main+8>
cpsr           0x60000010       1610612752
```

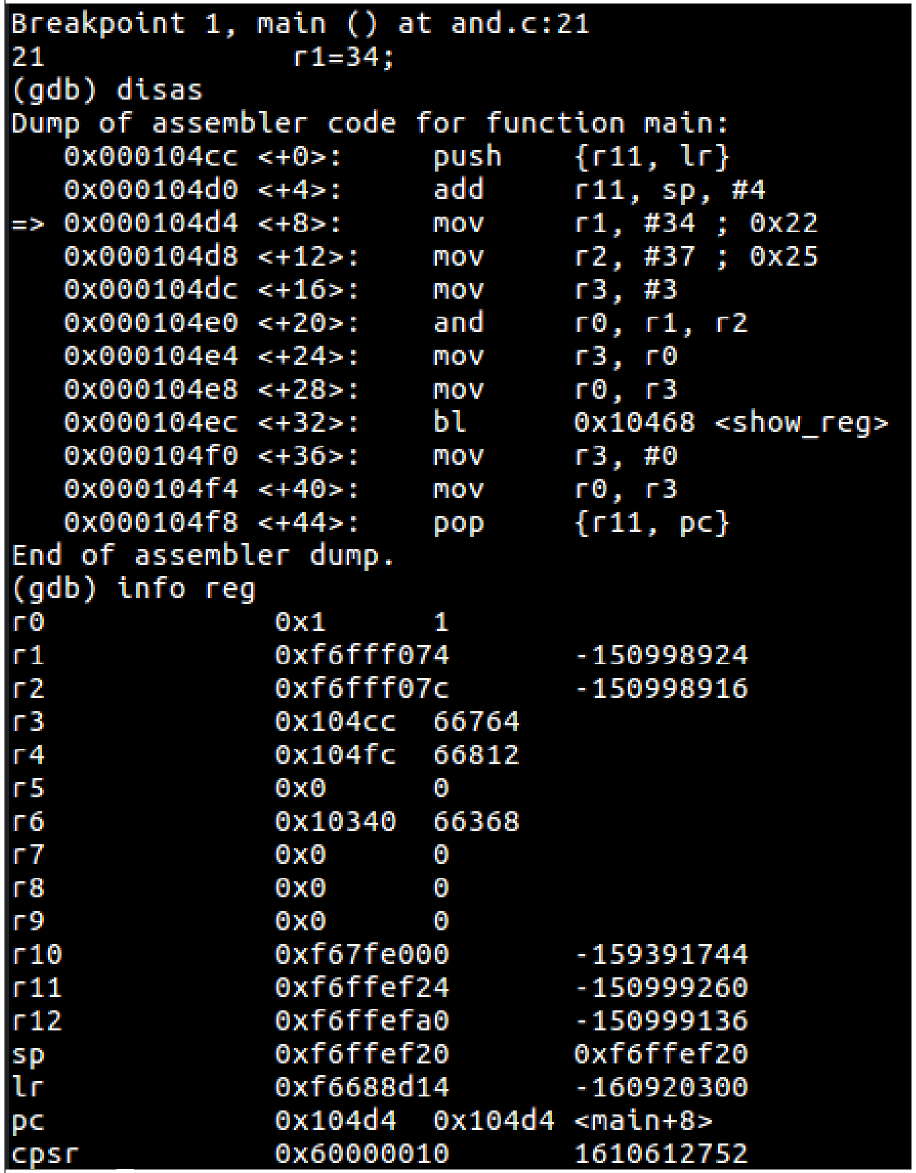
r1, r2, r3, r5값 세팅

r2와 r3값 비교해서 만족하지 않으면 다음 명령어 건너뜀.
rsble 은 5에서 r5레지스터 값을 빼서 r4에 저장시킴

and

```
[and.c]
#include<stdio.h>

void show_reg(unsigned int reg)
{
        int i;
        for(i=31;i>=0;)
                printf("%d", (reg>>i--)&1); //레지스터 값을 출력함.

        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;
        r3 =3;

        asm volatile("and r0, r1, r2"); // r1과 r2를 and 연산 한 값을 r0에 저장함

        show_reg(r0);
        return 0;
}
```

```
xeno@xeno-NH:~/proj/0430$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
00000000000000000000000000100000
```



```
Breakpoint 1, main () at and.c:21
21              r1=34;
(gdb) disas
Dump of assembler code for function main:
   0x000104cc <+0>:     push    {r11, lr}
   0x000104d0 <+4>:     add     r11, sp, #4
=> 0x000104d4 <+8>:     mov     r1, #34 ; 0x22
   0x000104d8 <+12>:    mov     r2, #37 ; 0x25
   0x000104dc <+16>:    mov     r3, #3
   0x000104e0 <+20>:    and     r0, r1, r2
   0x000104e4 <+24>:    mov     r3, r0
   0x000104e8 <+28>:    mov     r0, r3
   0x000104ec <+32>:    bl      0x10468 <show_reg>
   0x000104f0 <+36>:    mov     r3, #0
   0x000104f4 <+40>:    mov     r0, r3
   0x000104f8 <+44>:    pop     {r11, pc}
End of assembler dump.
(gdb) info reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x104cc  66764
r4             0x104fc  66812
r5             0x0      0
r6             0x10340  66368
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef20       0xf6ffef20
lr             0xf6688d14       -160920300
pc             0x104d4  0x104d4 <main+8>
cpsr           0x60000010       1610612752
```

r1과 r2값을 and 연산 시킨 결과를 r0에 넣음.

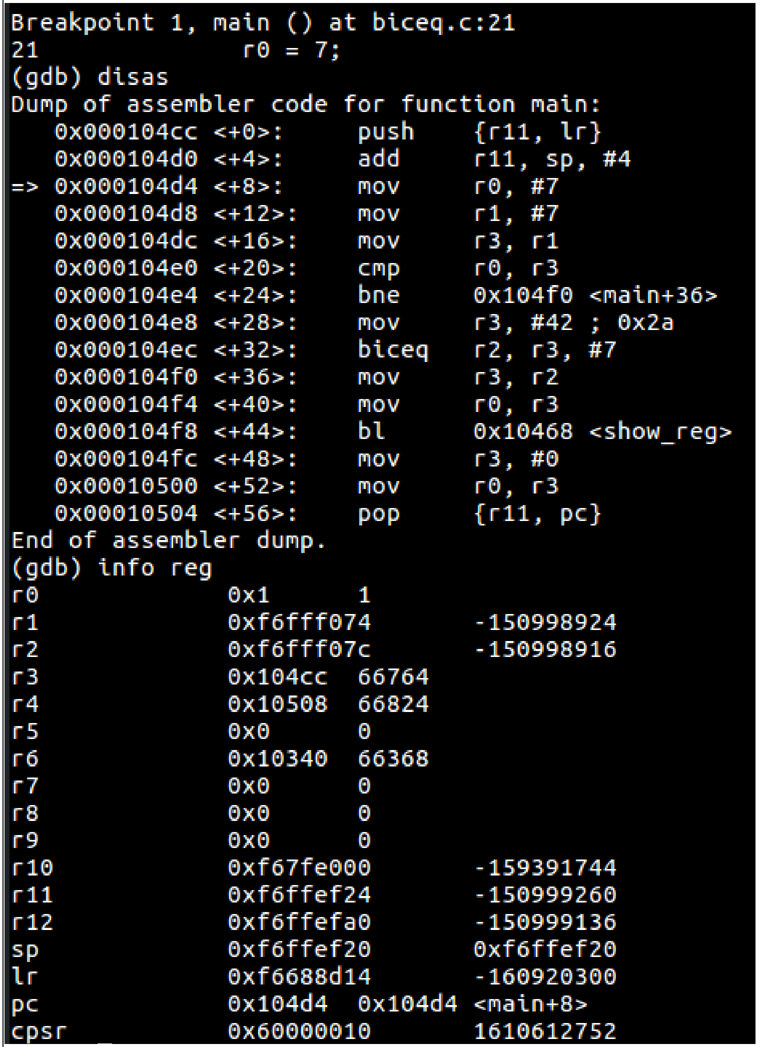
biceq

: BIt Clear(and not) Equal

```
[biceq.c]
#include<stdio.h>
void show_reg(unsigned int reg)
{
        int l;
        for(l=31;l>=0;)
                printf("%d", (reg>>l--)&1); // 레지스터 값 출력

        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) // r0레지스터와 r1레지스터 값이 같으면
        {
                r3 = 42;
                asm volatile("biceq r2, r3, #7");
        }
```

```
	show_reg(r2);
	return 0;
}
```

```
xeno@xeno-NH:~/proj/0430$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
00000000000000000000000000101000
```



```
Breakpoint 1, main () at biceq.c:21
21                r0 = 7;
(gdb) disas
Dump of assembler code for function main:
   0x000104cc <+0>:     push    {r11, lr}
   0x000104d0 <+4>:     add     r11, sp, #4
=> 0x000104d4 <+8>:     mov     r0, #7
   0x000104d8 <+12>:    mov     r1, #7
   0x000104dc <+16>:    mov     r3, r1
   0x000104e0 <+20>:    cmp     r0, r3
   0x000104e4 <+24>:    bne     0x104f0 <main+36>
   0x000104e8 <+28>:    mov     r3, #42 ; 0x2a
   0x000104ec <+32>:    biceq   r2, r3, #7
   0x000104f0 <+36>:    mov     r3, r2
   0x000104f4 <+40>:    mov     r0, r3
   0x000104f8 <+44>:    bl      0x10468 <show_reg>
   0x000104fc <+48>:    mov     r3, #0
   0x00010500 <+52>:    mov     r0, r3
   0x00010504 <+56>:    pop     {r11, pc}
End of assembler dump.
(gdb) info reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x104cc  66764
r4             0x10508  66824
r5             0x0      0
r6             0x10340  66368
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef20       0xf6ffef20
lr             0xf6688d14       -160920300
pc             0x104d4  0x104d4 <main+8>
cpsr           0x60000010       1610612752
```

cmp 연산으로 r0와 r3값이 같은지 판별하고

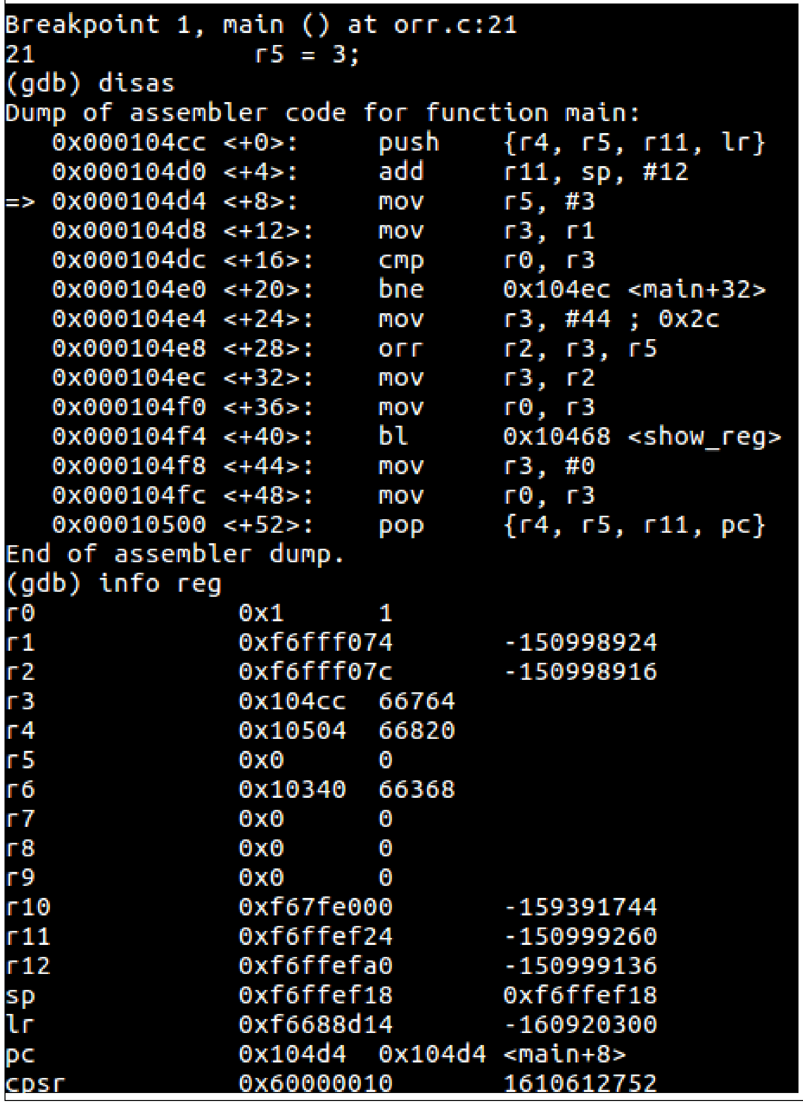
bic'eq'이므로 같으면 이 명령어 수행. r3값 42와 7을 and 시키고 not 연산을 하여 r2에 넣음.

orr

: logic OR

```
[orr.c]
#include<stdio.h>

void show_reg(unsigned int reg)
{
        int l;
        for(l=31;l>=0;)
                printf("%d", (reg>>l--)&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");

        r5 = 3;

        if(r0 == r1)
        {
                r3 = 44;
                asm volatile("orr r2, r3, r5"); //or 연산하여 r2에 넣기
        }

        show_reg(r2);
        return 0;
}
```

```
xeno@xeno-NH:~/proj/0430$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r4 = 2
```



```
Breakpoint 1, main () at orr.c:21
21              r5 = 3;
(gdb) disas
Dump of assembler code for function main:
   0x000104cc <+0>:     push    {r4, r5, r11, lr}
   0x000104d0 <+4>:     add     r11, sp, #12
=> 0x000104d4 <+8>:     mov     r5, #3
   0x000104d8 <+12>:    mov     r3, r1
   0x000104dc <+16>:    cmp     r0, r3
   0x000104e0 <+20>:    bne     0x104ec <main+32>
   0x000104e4 <+24>:    mov     r3, #44 ; 0x2c
   0x000104e8 <+28>:    orr     r2, r3, r5
   0x000104ec <+32>:    mov     r3, r2
   0x000104f0 <+36>:    mov     r0, r3
   0x000104f4 <+40>:    bl      0x10468 <show_reg>
   0x000104f8 <+44>:    mov     r3, #0
   0x000104fc <+48>:    mov     r0, r3
   0x00010500 <+52>:    pop     {r4, r5, r11, pc}
End of assembler dump.
(gdb) info reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x104cc  66764
r4             0x10504  66820
r5             0x0      0
r6             0x10340  66368
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef18       0xf6ffef18
lr             0xf6688d14       -160920300
pc             0x104d4  0x104d4 <main+8>
cpsr           0x60000010       1610612752
```

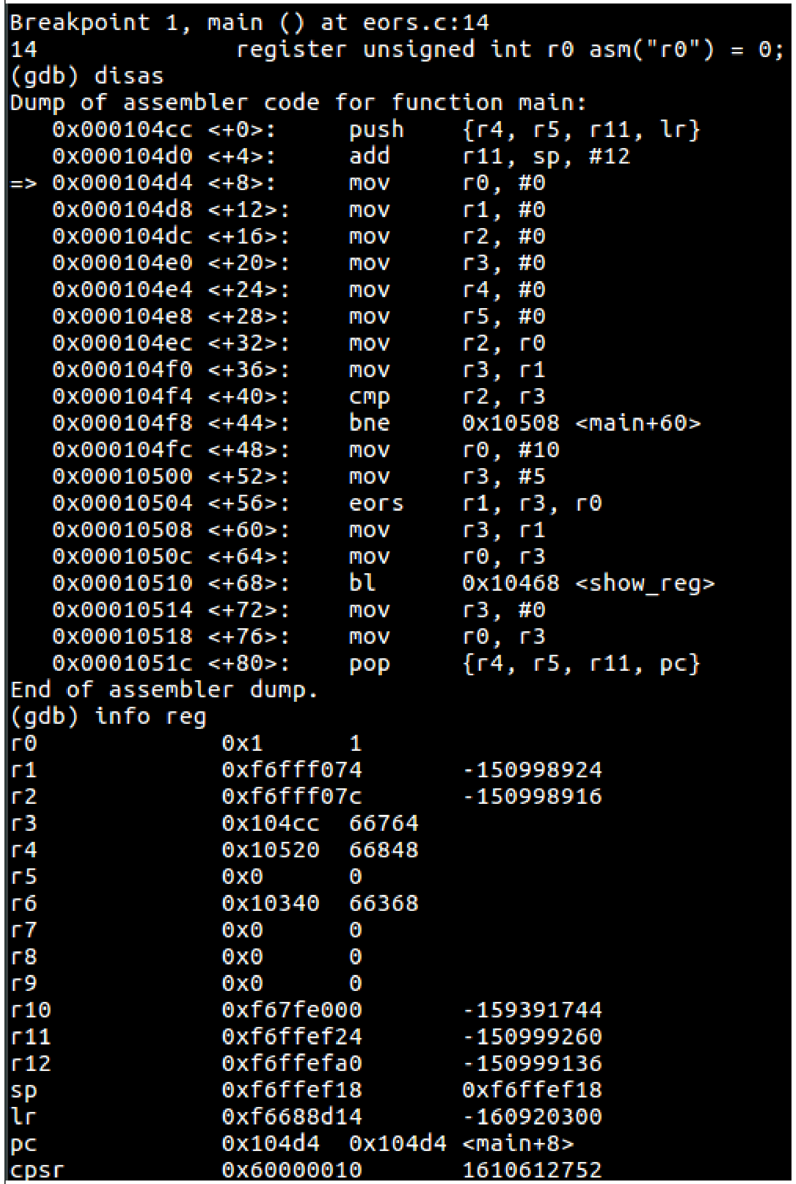
r3와 r5를 OR 연산 시켜서 결과 값을 r2에 넣음.
44와 3을 or 연산 시키면 101100 | 11 == 101111

eors

: Exclusive OR

```
[eors.c]
#include<stdio.h>

void show_reg(unsigned int reg)
{
        int l;
        for(l=31;l>=0;)
                printf("%d", (reg>>l--)&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"); //xor 연산  1111
        }
        show_reg(r1);
        return 0;
}
```

```
xeno@xeno-NH:~/proj/0430$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
00000000000000000000000000001111
```



```
Breakpoint 1, main () at eors.c:14
14              register unsigned int r0 asm("r0") = 0;
(gdb) disas
Dump of assembler code for function main:
   0x000104cc <+0>:     push    {r4, r5, r11, lr}
   0x000104d0 <+4>:     add     r11, sp, #12
=> 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>:    mov     r2, r0
   0x000104f0 <+36>:    mov     r3, r1
   0x000104f4 <+40>:    cmp     r2, r3
   0x000104f8 <+44>:    bne     0x10508 <main+60>
   0x000104fc <+48>:    mov     r0, #10
   0x00010500 <+52>:    mov     r3, #5
   0x00010504 <+56>:    eors    r1, r3, r0
   0x00010508 <+60>:    mov     r3, r1
   0x0001050c <+64>:    mov     r0, r3
   0x00010510 <+68>:    bl      0x10468 <show_reg>
   0x00010514 <+72>:    mov     r3, #0
   0x00010518 <+76>:    mov     r0, r3
   0x0001051c <+80>:    pop     {r4, r5, r11, pc}
End of assembler dump.
(gdb) info reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x104cc  66764
r4             0x10520  66848
r5             0x0      0
r6             0x10340  66368
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef18       0xf6ffef18
lr             0xf6688d14       -160920300
pc             0x104d4  0x104d4 <main+8>
cpsr           0x60000010       1610612752
```

r0~r5 까지 0으로 세팅

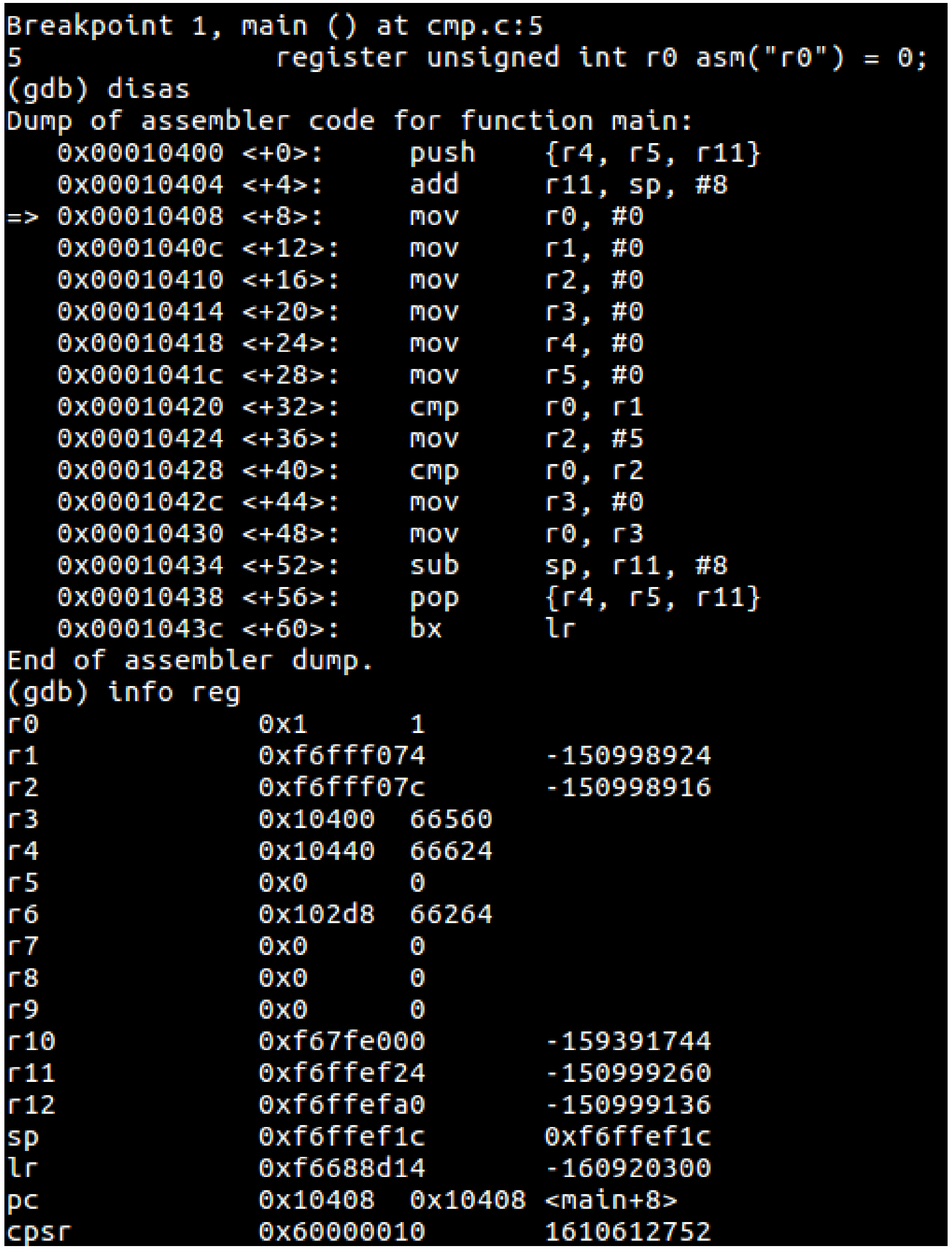
r2와 r3가 같으므로 r0와 r3를 xor 시켜 r1에 저장.
5^10 = 1111

cmp

```
[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;
}
```

~결과

r0에서 r1을 빼서 0이 나오면 같은 값, 양수일 경우 r0가 더 크고 음수일 경우 r1이 더 크다고 판단한다.

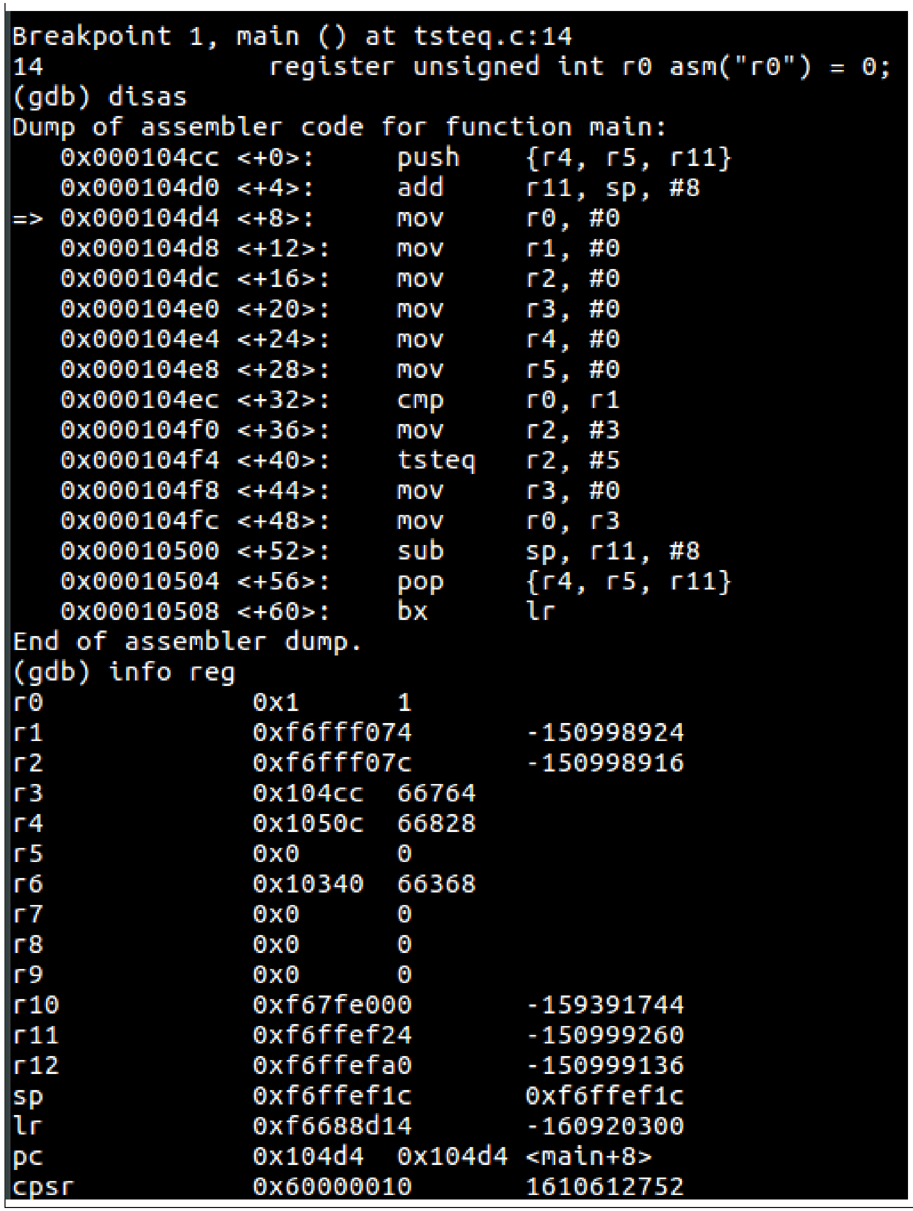


```
Breakpoint 1, main () at cmp.c:5
5               register unsigned int r0 asm("r0") = 0;
(gdb) disas
Dump of assembler code for function main:
   0x00010400 <+0>:     push    {r4, r5, r11}
   0x00010404 <+4>:     add     r11, sp, #8
=> 0x00010408 <+8>:     mov     r0, #0
   0x0001040c <+12>:    mov     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 reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x10400  66560
r4             0x10440  66624
r5             0x0      0
r6             0x102d8  66264
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef1c       0xf6ffef1c
lr             0xf6688d14       -160920300
pc             0x10408  0x10408 <main+8>
cpsr           0x60000010       1610612752
```

tsteq

: Test equal

```
[tsteq.c]
#include<stdio.h>

void show_reg(unsigned int reg)
{
        int l;
        for(l=31;l>=0;)
                printf("%d", (reg>>l--)&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"); //3이랑 5를 and 시킴 , cpsr 값이 6에서 2로 바뀜 제로플래그가 꺼졌다는 뜻.

        return 0;
}
```



```
Breakpoint 1, main () at tsteq.c:14
14              register unsigned int r0 asm("r0") = 0;
(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) info reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x104cc  66764
r4             0x1050c  66828
r5             0x0      0
r6             0x10340  66368
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef1c       0xf6ffef1c
lr             0xf6688d14       -160920300
pc             0x104d4  0x104d4 <main+8>
cpsr           0x60000010       1610612752
```

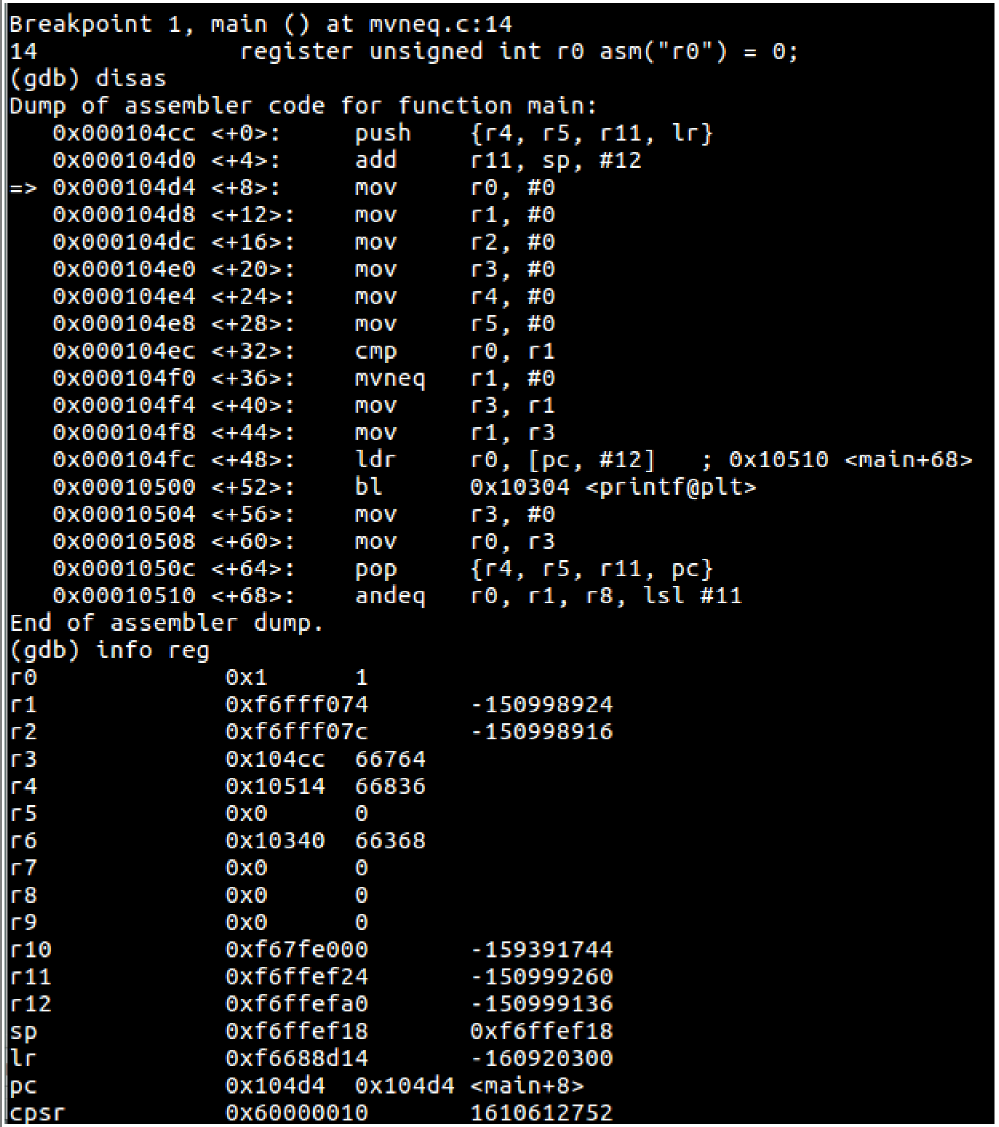
r0와 r1이 같을 경우
r2와 5를 and 연산한 뒤 버린다.

mvneq

: mov not equal

```
[mvneq.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");
        asm volatile("mvneq r1, #0"); // xor,

        printf("r1 = 0x%x₩n",r1);

        return 0;
}
```

xeno@xeno-NH:~/proj/0430$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out r1 = 0xffffffff



```
Breakpoint 1, main () at mvneq.c:14
14              register unsigned int r0 asm("r0") = 0;
(gdb) disas
Dump of assembler code for function main:
   0x000104cc <+0>:     push    {r4, r5, r11, lr}
   0x000104d0 <+4>:     add     r11, sp, #12
=> 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>:    mvneq   r1, #0
   0x000104f4 <+40>:    mov     r3, r1
   0x000104f8 <+44>:    mov     r1, r3
   0x000104fc <+48>:    ldr     r0, [pc, #12]   ; 0x10510 <main+68>
   0x00010500 <+52>:    bl      0x10304 <printf@plt>
   0x00010504 <+56>:    mov     r3, #0
   0x00010508 <+60>:    mov     r0, r3
   0x0001050c <+64>:    pop     {r4, r5, r11, pc}
   0x00010510 <+68>:    andeq   r0, r1, r8, lsl #11
End of assembler dump.
(gdb) info reg
r0             0x1      1
r1             0xf6fff074       -150998924
r2             0xf6fff07c       -150998916
r3             0x104cc  66764
r4             0x10514  66836
r5             0x0      0
r6             0x10340  66368
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0xf6ffef24       -150999260
r12            0xf6ffefa0       -150999136
sp             0xf6ffef18       0xf6ffef18
lr             0xf6688d14       -160920300
pc             0x104d4  0x104d4 <main+8>
cpsr           0x60000010       1610612752
```

r0와 r1이 같을 경우
0을 r1으로 mov 한 뒤 not 연산을 시킴

Table 4.1. Location of ARM instructions

Mnemonic	Brief description	Page	Architecture[1]
ADC, ADD	Add with carry, Add	ADD, SUB, RSB, ADC, SBC, and RSC	All
AND	Logical AND	AND, ORR, EOR, and BIC	All
B	Branch	B and BL	All
BIC	Bit clear	AND, ORR, EOR, and BIC	All
BKPT	Breakpoint	BKPT	5
BL	Branch with link	B and BL	All
BLX	Branch, link and exchange	BLX	5T[2]
BX	Branch and exchange	BX	4T[b]
CDP, CDP2	Coprocessor data operation	CDP, CDP2	2, 5
CLZ	Count leading zeroes	CLZ	5
CMN, CMP	Compare negative, Compare	CMP and CMN	All
EOR	Exclusive OR	AND, ORR, EOR, and BIC	All
LDC, LDC2	Load coprocessor	LDC, STC	2, 5
LDM	Load multiple registers	LDM and STM	All
LDR	Load register	ARM memory access instructions	All

Mnemonic	Brief description	Page	Architecture[1]
MAR	Move from registers to 40-bit accumulator	*MAR, MRA*	XScale[3]
MCR, MCR2, MCRR	Move from register(s) to coprocessor	*MCR, MCR2, MCRR*	2, 5, 5E[4]
MIA, MIAPH, MIAxy	Multiply with internal 40-bit accumulate	*MIA, MIAPH, and MIAxy*	XScale
MLA	Multiply accumulate	*MUL and MLA*	2
MOV	Move	*MOV and MVN*	All
MRA	Move from 40-bit accumulator to registers	*MAR, MRA*	XScale
MRC, MRC2	Move from coprocessor to register	*MRC, MRC2*	2, 5
MRRC	Move from coprocessor to 2 registers	*MRRC*	5E[d]
MRS	Move from PSR to register	*MRS*	3
MSR	Move from register to PSR	*MSR*	3
MUL	Multiply	*MUL and MLA*	2
MVN	Move not	*MOV and MVN*	All
ORR	Logical OR	*AND, ORR, EOR, and BIC*	All
PLD	Cache preload	*PLD*	5E[d]
QADD, QDADD, QDSUB, QSUB	Saturating arithmetic	*QADD, QSUB, QDADD, and QDSUB*	5ExP[5]
RSB, RSC, SBC	Reverse sub, Reverse sub with carry, Sub with carry	*ADD, SUB, RSB, ADC, SBC, and RSC*	All

Mnemonic	Brief description	Page	Architecture[1]
SMLAL	Signed multiply-accumulate (64 <= 32 x 32 + 64)	*UMULL, UMLAL, SMULL and SMLAL*	M[6]
SMLALxy	Signed multiply-accumulate (64 <= 16 x 16 + 64)	*SMLALxy*	5ExP[e]
SMLAWy	Signed multiply-accumulate (32 <= 32 x 16 + 32)	*SMLAWy*	5ExP[e]
SMLAxy	Signed multiply-accumulate (32 <= 16 x 16 + 32)	*SMLAxy*	5ExP[e]
SMULL	Signed multiply (64 <= 32 x 32)	*UMULL, UMLAL, SMULL and SMLAL*	M[f]
SMULWy	Signed multiply (32 <= 32 x 16)	*SMULWy*	5ExP[e]
SMULxy	Signed multiply (32 <= 16 x 16)	*SMULxy*	5ExP[e]
STC, STC2	Store coprocessor	*LDC, STC*	2, 5ExP[e]
STM	Store multiple registers	*LDM and STM*	All
STR	Store register	*ARM memory access instructions*	All
SUB	Subtract	*ADD, SUB, RSB, ADC, SBC, and RSC*	All
SWI	Software interrupt	*SWI*	All
SWP	Swap registers and memory	*SWP*	3
TEQ, TST	Test equivalence, Test	*TST and TEQ*	All
UMLAL, UMULL	Unsigned MLA, MUL (64 <= 32 x 32 (+ 64))	*UMULL, UMLAL, SMULL and SMLAL*	M[f]