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

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

ARM 환경설정

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

이후에 C 소스 파일을 작성한다.

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

컴파일을 하려면, 터미널을 2 개 띄운다. 기존 터미널에서 아래 명령어를 수행한다.

qemu-arm-static -g 1234 -L /usr/arm-linux-gnueabi ./a.out (디버그 실행 용)

다른 터미널에서 아래 명령어를 수행한다.

gdb-multiarch (디버그 진입)

file a.out (해당 파일 소스 분석)

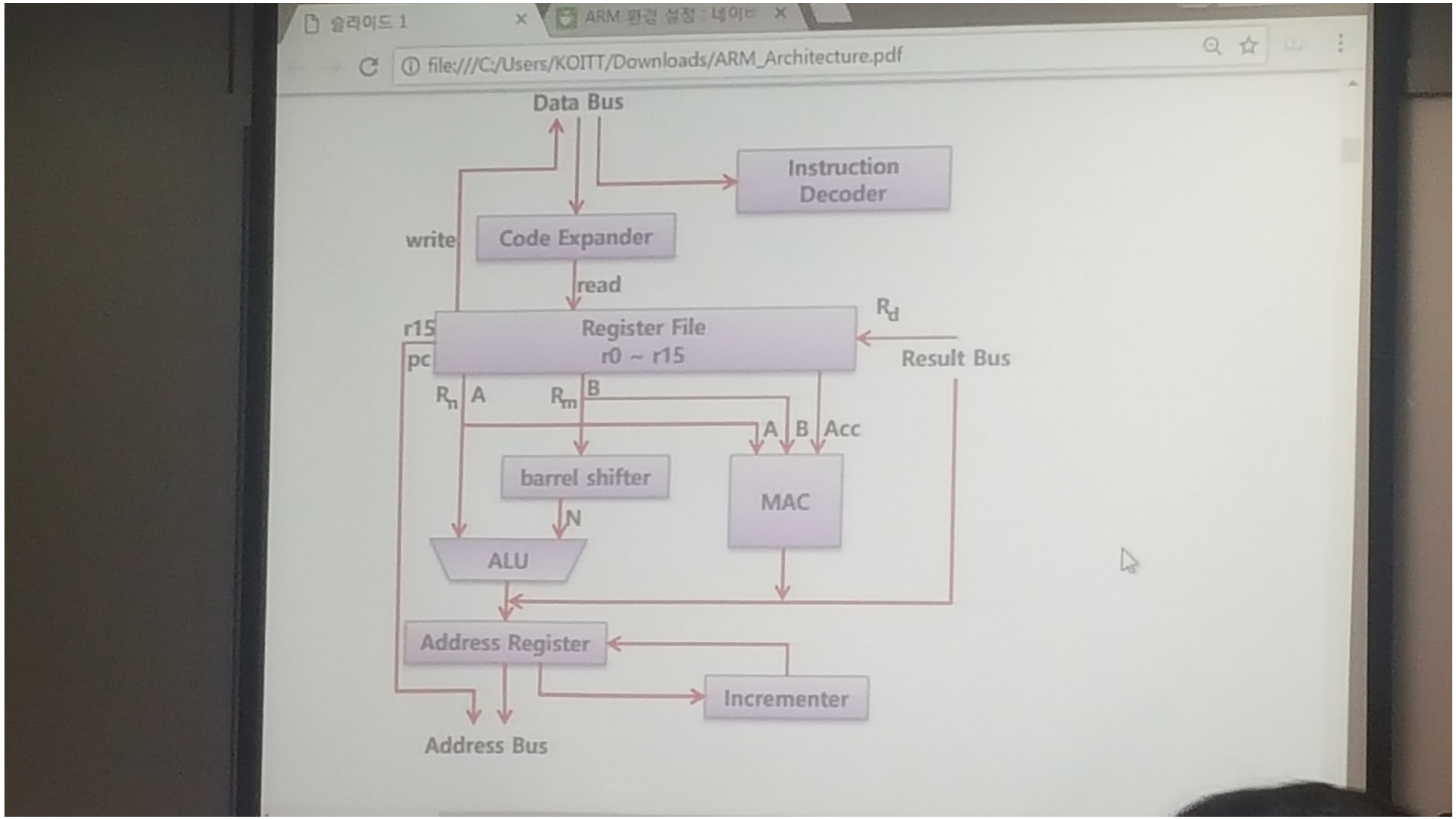
target remote localhost:1234

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.

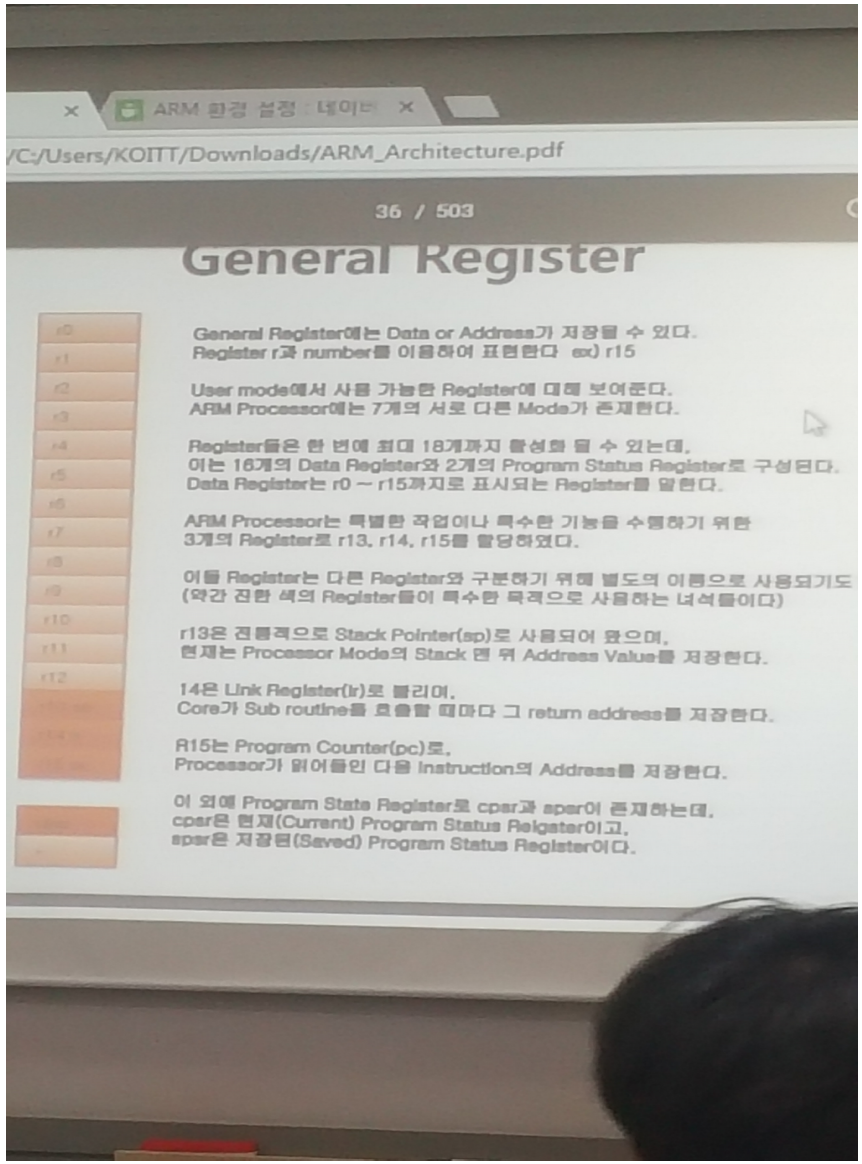
MCU 관련 모터 선정 토크 계산
http://cafe.naver.com/hestit/3109
http://cafe.naver.com/hestit/2266

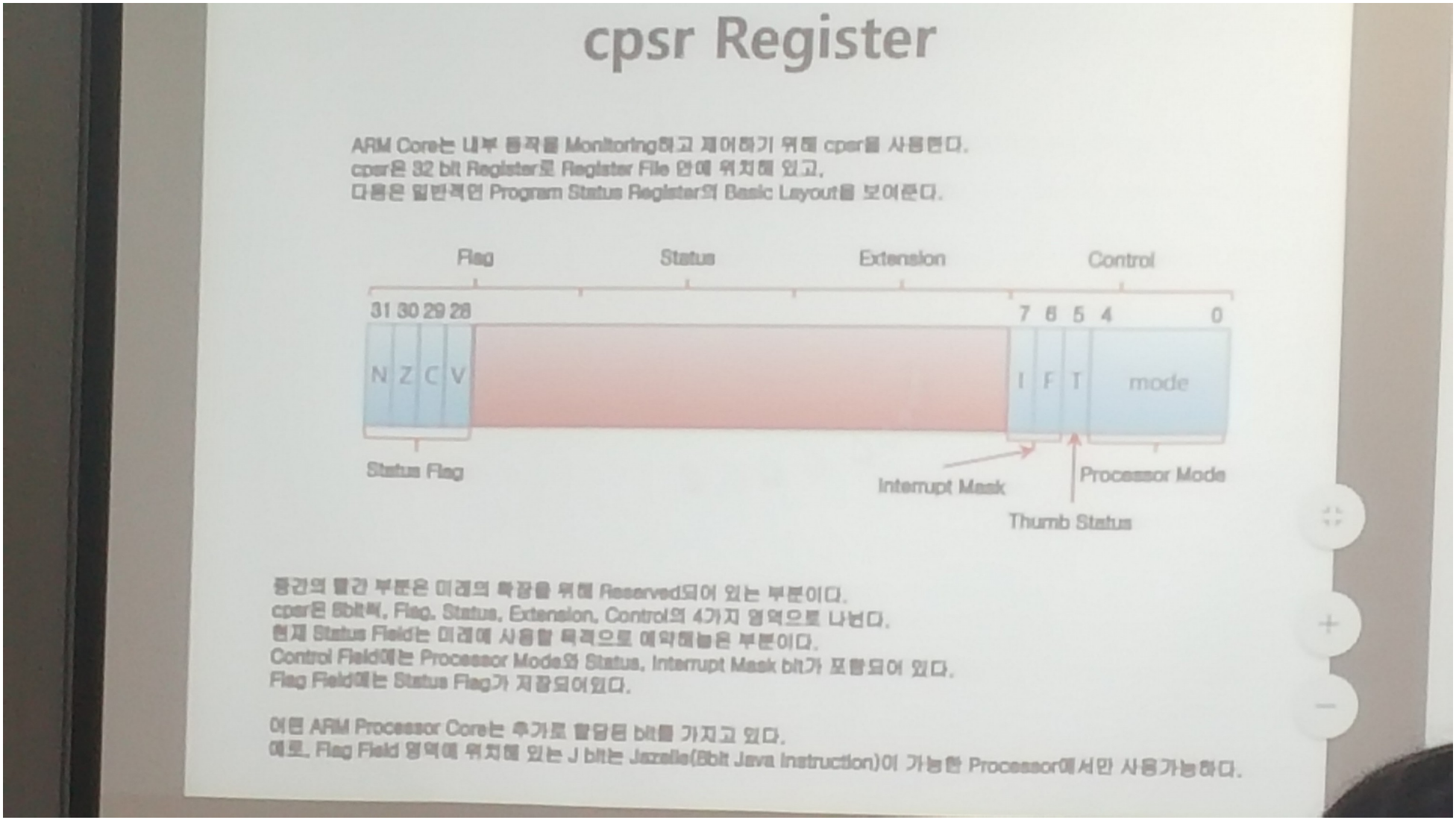


데이터 버스와 어드레스 버스가 중요하다**. code expander** 는 신경쓰지 말아라**.(cpu** 설계 할거 아니면**)**
register file 매우 중요 **(r0 ~ r15** 의 레지스터가 있다**.)**
MAC 이건 곱셈기이다**. MAC** 이 들어있으면 **DSP** 가 되는것이고 없으면 그냥 **arm** 이다**.**
연산 클럭을 줄이는게 핵심이다**.**
보편적으로 곱셈을하면**,** 클럭이 수 클럭 ~ 수십클럭이 소비된다**. (**그 시간에 덧셈을 여러개 할 수 있다**.** 따라서 손해가 심하다**.)**
MAC 이 있으면 곱셈연산을 **1** 클럭 만에 끝내버린다**.(**심지어 곱셈 덧셈을 병렬로 수행하는 것을 **1** 클럭에 끝내버린다**.** 이게 **MAC** 성능차이가 무시무시하다**.)**

sin(x)*e^ix 를 계산한다고 할때
sin(x)는 미분 가능한 연속함수 **(**아날로그 함수**)**
sin(x)는 테일러 급수로 처리한다**.**
e^ix = cos x + i sin x r^2 = x^2+y^2 == cos(r)~~x~~ + sin(r)~~y~~

cpu 는 클록 **(**주파수**)**가 존재한다**.** 클록 사이에 빈 시간에 들어오는 값은 씹는다**. (**샘플링**)**


ARM 환경 설정 : 네이버
/C:/Users/KOITT/Downloads/ARM_Architecture.pdf
36 / 503
General Register
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
General Register에는 Data or Address가 저장될 수 있다.
Register r과 number를 이용하여 표현한다 ex) r15
User mode에서 사용 가능한 Register에 대해 보여준다.
ARM Processor에는 7개의 서로 다른 Mode가 존재한다.
Register들은 한 번에 최대 18개까지 활성화 될 수 있는데,
이는 16개의 Data Register와 2개의 Program Status Register로 구성된다.
Data Register는 r0 ~ r15까지로 표시되는 Register를 말한다.
ARM Processor는 특별한 작업이나 특수한 기능을 수행하기 위한
3개의 Register로 r13, r14, r15를 할당하였다.
이들 Register는 다른 Register와 구분하기 위해 별도의 이름으로 사용되기도
(약간 전환 세의 Register들이 특수한 목적으로 사용하는 녀석들이다)
r13은 전통적으로 Stack Pointer(sp)로 사용되어 왔으며,
현재는 Processor Mode의 Stack 맨 위 Address Value를 저장한다.
14은 Link Register(lr)로 불리며,
Core가 Sub routine를 호출할 때마다 그 return address를 저장한다.
R15는 Program Counter(pc)로,
Processor가 읽어들인 다음 Instruction의 Address를 저장한다.
이 외에 Program State Register로 cpsr과 aper이 존재하는데,
cpsr은 현재(Current) Program Status Reigster이고,
spsr은 저장된(Saved) Program Status Register이다.



cprs -> intel eflags register 와 같음**.**

status register !

satuartion ->> **core** 연산**.**

debug 상에서

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
warning: remote target does not support file transfer, attempting to access files from local filesyst
em.
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0xf67ceb00 in ?? ()
(gdb) b main
Note: breakpoint 1 also set at pc 0x10440.
Breakpoint 2 at 0x10440: file add.c, line 8.
(gdb) c
Continuing.
warning: Could not load shared library symbols for 2 libraries, e.g. /lib/libc.so.6.
Use the "info sharedlibrary" command to see the complete listing.
Do you need "set solib-search-path" or "set sysroot"?

Breakpoint 1, main () at add.c:8
```

```
Breakpoint 1, main () at add.c:8
8           r1=77;
(gdb) info reg
r0             0x1      1
r1             0xf6fff034       -150998988
r2             0xf6fff03c       -150998980
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            0xf6ffeee4       -150999324
r12            0xf6ffef60       -150999200
sp             0xf6ffeee0       0xf6ffeee0
lr             0xf6686d14       -160928492
pc             0x10440  0x10440 <main+8>
cpsr           0x60000010       1610612752
(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)
```

```
(gdb) l
3       int main(void){
4           register unsigned int r0 asm("r0");
5           register unsigned int r1 asm("r1");
6           register unsigned int r2 asm("r2");
7
8           r1=77;
9           r2=37;
10
11          asm volatile("add r0,r1,r2");
12          printf("(asm add(r1,r2))r0 = %d\n",r0);
(gdb)
```

subgt.c

```
(gdb) info reg
r0             0x0        0
r1             0xf6fff153         -150998701
r2             0x0        0
r3             0x0        0
r4             0x0        0
r5             0x0        0
r6             0x0        0
r7             0x0        0
r8             0x0        0
r9             0x0        0
r10            0x20f0c    134924
r11            0x0        0
r12            0x0        0
sp             0xf6ffef70         0xf6ffef70
lr             0x0        0
pc             0xf67ceb00         0xf67ceb00
cpsr           0x10       16
(gdb) c
Continuing.
warning: Could not load shared library symbols for 2 libraries, e.g. /lib/libc.so.6.
Use the "info sharedlibrary" command to see the complete listing.
Do you need "set solib-search-path" or "set sysroot"?

Breakpoint 1, main () at subgt.c:10
10          r1 = 77;
(gdb) info reg
r0             0x1        1
r1             0xf6ffef74         -150999180
r2             0xf6ffef7c         -150999172
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            0xf6ffee24         -150999516
r12            0xf6ffeea0         -150999392
sp             0xf6ffee20         0xf6ffee20
lr             0xf6686d14         -160928492
pc             0x10440    0x10440 <main+8>
cpsr           0x60000010         1610612752
(gdb)
```

cpsr 의 값이 변하는것을 중점적으로 보면 된다**.**

subgt

sub & great then r3 에서 **1** 을 빼서 **33** 이 출력된다**.**

rsble

reverse serve less equal 방향을 뒤집어서 뺄셈 + 작거나 같은경우의 기능을 한다**.**

and
6 번째 자리에 **1** 이 나오는 이유는 **and** 연산이기 때문에

biceq

bit clear equal (동등한 값일 때만**,** 이 **asm** 명령어가 동작되게 하는것 **if** 문과는 좀 중첩적인**.. cmp** 를 쓰면 **if** 가 없어도 되는**..?)**
42 & ~(2^3 -1)
42 & ~(7)
42 를 **2^3** 의 배수로 정렬
따라서 **40** 으로 정렬이 된다**.**

orr(OR 연산**)**

eors (exclusive or)

cmp_mov

tsteq (bit 연산후 **cpsr** 에 전달한다**.)**