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

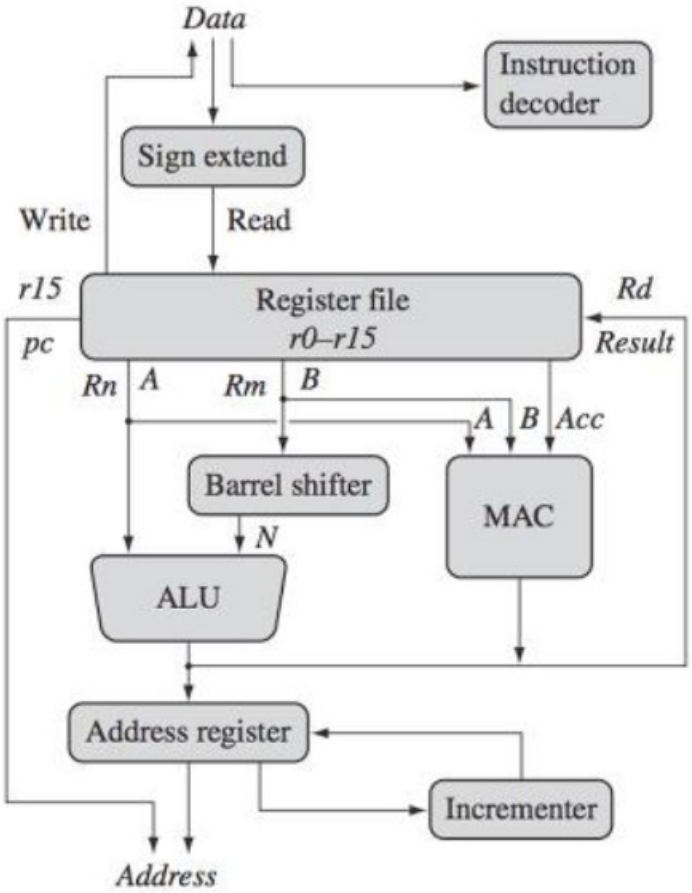
강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

a. arm processor



ARM – 폰노이만 구조(데이터와 명령어가 버스 공유)

<–> 하버드 구조(데이터와 명령어용 버스 분리)

Data

-> Data Bus를 통해 core로 입력

Instruction decoder

-> 명령어 실행 전 해독

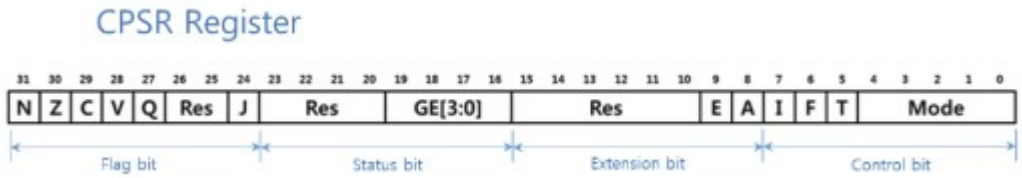
MAC 곱셈기

-> 옵션으로 들어있는 하드웨어이고 곱셈 + 덧셈을

최대 4개까지 동시에 병렬 수행하여 1clock에 처리 가능함

있으면 성능 대폭 증가함

CPSR(Current Program Status Register)



: 현재 프로그램 상태 저장하는 레지스터

arm 어셈블리 함수

* add r3,r1,r2
→ r3 = r1+ r2

* subgt r3,r1,r2
→ r3= r1-r2

* rsble r3,r1,r2 (reversee subtract)
→ r3 =r2 – r1

* and r3,r1,r2
→ r3= r1&r2

* biceq r3,r1,r2
→ r3=r1&~r2

* orr r3,r1,r2
→ r3 = r1|r2

*eors r3,r1,r2 (exclusive or)
→ r3 = r1^r2

*mov r1,r2
→ r1 = r2

*mvn r1,r2
→ r1 = 0xffffffff^r2

*cmp r1,r2
→ r1 – r2 하여 비교후 state flag 업데이트

*cmn r1,r2 (cmp negative)
→ r1 + r2 하여 비교

*tsteq r1,r2
→ r1 & r2 하여 비교 후 조건 flag 업데이트

*teq r1,r2

→ r1 ^ r2 하여 비교 후 조건 flag 업데이트

산술 명령어 뒤에 접미사 gt ge lt le eq 의미 (status flag상태를 보고 실행할지 말지 결정한다)

gt : great than

ge : great equal //크거나 같다

lt : less than

le: less equal // 작거나 같다.

eq : equal

ne : not equal

산술 명령어 뒤에 접미사 s 가 붙을 경우 → 산술 처리 결과에 따라 flag 업데이트 한다.

순서
sudo apt-get update
sudo apt-get install qemu-user-static qemu-system
sudo apt-get install gcc-arm-linux-gnueqbi

이제 arm gcc를 사용할 수 있다.

arm-linux-gnueabi-gcc -g a.c

qemu-arm-static -g 1234 -L /usr/arm-linux-gnueabi ./a.out
(qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out) 는 ./a.out 임

gdb-multiarch

file a.out

target remote localhost:1234

b main

info reg

c

info reg

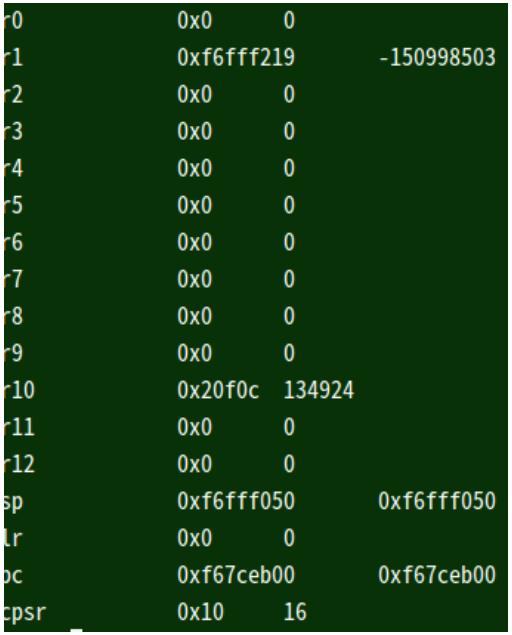
1. subgt

```
#include <stdio.h>

int main(void)
{
   register unsigned int r0 asm("r0"); //레지스터 정의
   register unsigned int r1 asm("r1");
   register unsigned int r2 asm("r2");
   register unsigned int r3 asm("r3");

   r1 = 77;
   r2 = 37;
   r3 = 34;
```

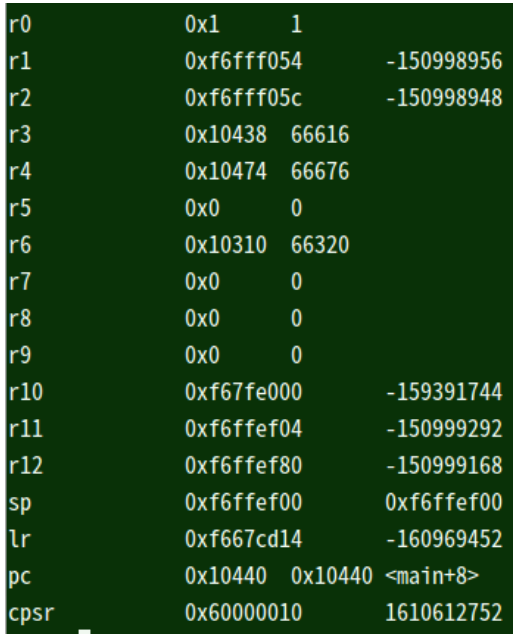
```
  if(r1>r2)1
    asm volatile("subgt r3, r3, #1");

  printf("r3 = %d\n", r3);
  return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/4_30$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r3 = 33
```

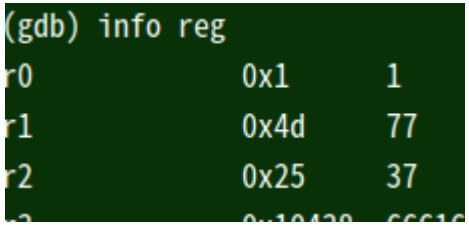
1. main문 시작할때 처음 레지스터 값



2. n, info reg: 초기화 안해주면 쓰레기 값



3. r3까지 값 대입했을 때 값이 레지스터에 저장된것을 볼 수 있다.



2. rsble

```
#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;
}
```

b main

```
(gdb) b main
Breakpoint 1 at 0x10440: file b.c, line 12.
(gdb) info reg
r0             0x0      0
r1             0xf6fff221       -150998495
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             0xf6fff060       0xf6fff060
lr             0x0      0
pc             0xf67ceb00       0xf67ceb00
cpsr           0x10     16
```

c

```
Breakpoint 1, main () at b.c:12
12          r1 = 77;
(gdb) info reg
r0             0x1      1
r1             0xf6fff064       -150998940
r2             0xf6fff06c       -150998932
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            0xf6ffef14       -150999276
r12            0xf6ffef90       -150999152
sp             0xf6ffef08       0xf6ffef08
lr             0xf667ed14       -160961260
pc             0x10440  0x10440 <main+8>
cpsr           0x60000010       1610612752
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/4_30$ qemu-arm-static -L /usr/arm-linux-gnueabi ./b.out
r4 = 2
```

:

3.and

```
#include <stdio.h>

void show_reg(unsigned int reg)
{
   int i;

   for(i=31;i>=0;)
      printf("%d",(reg>>i--) & 1); //   reg 32 비트 부터 1비트까지 해당비트를 체크한다.  32비트의 녀석이 1이 나오는 것을 확인하면 됨.
   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 = 77;
   r2 = 37;
   r5=3;

   asm volatile("and r0, r1, #2");

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

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/4_30$ qemu-arm-static -L /usr/arm-linux-gnueabi ./c.out
00000000000000000000000000000000
```

4.biceq

```
#include <stdio.h>

void show_reg(unsigned int reg)
{
   int i;

   for(i=31;i>=0;)
      printf("%d",(reg>>i--) & 1); // 28분
   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"); // 42 & ~(2^3-1)
                                        // 42 & ~(7) == 42를 2^3의 배수로 정렬, 즉 40임.
   }

   show_reg(r2);
   return 0;
```

```
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/4_30$ qemu-arm-static -L /usr/arm-linux-gnueabi ./d.out
00000000000000000000000000101000
```

5. orr

```
#include <stdio.h>

void show_reg(unsigned int reg)
{
   int i;

   for(i=31;i>=0;)
      printf("%d",(reg>>i--) & 1); // 28분
   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");
  }

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

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/4_30$ qemu-arm-static -L /usr/arm-linux-gnueabi ./e.out
11110110111111111111000001101100
```

6. eors

```
#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");
   }

   show_reg(r1);
   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/4_30$ qemu-arm-static -L /usr/arm-linux-gnueabi ./f.out
00000000000000000000000000001111
```

7. cmp, mov

```
#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;
}
```


8. tsteq : tst는 실제 레지스터를 변화시키는 것이아니라 CPSR값을 변화시킴. "tsteq r2,#5"를 수행하고 나면 결과 값이 -이 아니므로 cpsr의 z비트가 0으로 바뀜

```
#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;
}
```

9. mvneq

```
#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");

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

   return 0;
}
```

디버깅 진행. 이제 arm architecture에 대해 Deep Down 할 수 있는 기점이 마련됐다.