

TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 : 황수정

sue100012@naver.com

44 일차 (2018. 04. 30)

목차

- CPU 발전사
- ARM Processor
 - > DSP 와 MAC ?
- General Register 범용 레지스터
- 리눅스 내의 ARM 환경설정
- 학습예제

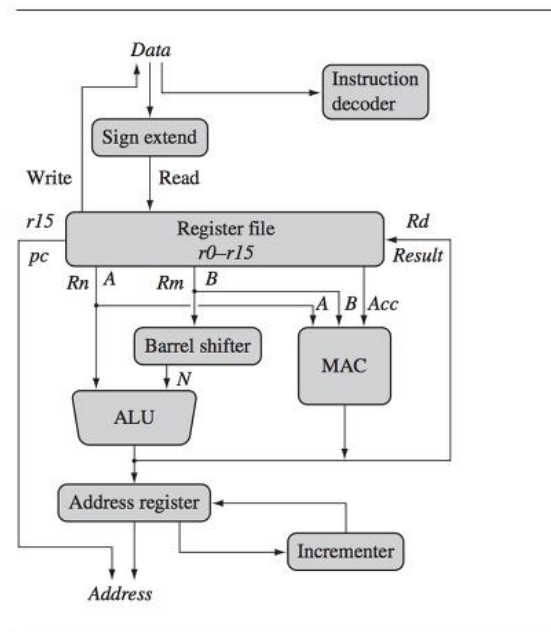
CPU 발전사

CISC → RISC → VLIW → Multi-Core → Heterogeneous

이제는 병렬처리 시대가 아니다. CPU 발전사가 바뀌게 되었다.

- 임베디드 시스템이란? 우리가 사용하는 전자제품을 모두를 말한다.

ARM Processor



ARM core dataflow model.

data bus : 데이터를 가져온다.

address bus : 주소를 가져온다.

address bus 가 data 의 위치를 알려주고 그 위치에 data Bus 가 data 를 갖다 준다.

barrel shifter : 한 개의 연산으로 데이터 워드 내의 다수의 비트를 이동하거나 회전 시킬 수 있는 하드웨어 장치이다. ALU 로 들어가는 한쪽 입력 데이터를 Shift 시키는 역할을 하는데, ARM 의 Barrel Shifter 는 시프트에 소요되는 시간이 Shift 비트수에 상관없이 일정하게 설계되었다.(?)

Register file 이 매우 중요하다.

Instruction decoder : 명령어 실행 전 해석해준다.

ALU(Arithmetic Logic Unit) : 명령어에 해당하는 산술 및 논리 연산을 수행한다.

MAC : 곱셈기이다. 옵션으로 들어있는 하드웨어이고(있으면 DSP 가 되는 것) 곱셈과 덧셈을 최대 4 개까지 동시에 병렬 수행하여 1clock 에 처리 가능하다. 있으면 성능 대폭 증가한다.

- DSP 와 MAC?

보편적으로 ARM 은 MAC 이 없고, MAC 이 있으면 DSP 이다. MAC 이 있으면 연산 클럭 수를 줄여 매우 효율적이다. 이는 원래 곱셈하는데 많은 클럭을 사용하는데 MAC 을 통하면 1 클럭 만에 곱셈 연산을 끝낼 수 있기 때문이다. 곱셈뿐 아니라 덧셈도 동시에 수행하며, 병렬도 수행할 수 있으면서 싱글로 처리하기 때문이다. 그래서 존재 유무에 따라 성능차이가 매우 난다.

예를 들어 $\sin(x) * e^{iy}$ 를 연산한다 가정할 때, 곱하는 각 요소들은 아날로그 함수($\sin(x)$ 는 연속 함수로 테일러급수를 사용)여서 디지털 처리하는 컴퓨터는 동작주파수 f 를 이용해 샘플링을 하여 아날로그적 신호를 디지털 신호로 나타내게 된다. 샘플링 타임이 길면 좋지 않은데, 이는 중간 버려지는 값이 많기 때문이다. 예를 들어 샘플링타임이 0.0001 초로 짧다고 하면 중간에서 일부 손실이 발생하나 2 초보다 의미 있는 값이 된다. 더 값을 살리고 싶다면 샘플링타임을 줄이면 된다. 그러나 단가가 올라간다. 보통 성능 좋은 것이 레이더용이다.

General register 범용레지스터

ARM 에서의 범용 레지스터는 16 개이다. 그러나 한번에 최대 18 개의 레지스터가 활성화될 수 있다. 이는 $r_0 \sim r_{15}$ 의 16 개 data Register, cpsr 와 spsr 의 2 개의 program Status Register 가 존재하기 때문이다.

r_{13} : Stack Pointer(sp), Processor Mode 의 Stack 맨 위 Address Value 를 저장한다.

r_{14} : Link Register(lr), Core 가 sub routine 을 호출할 때 마다 그 return address 를 저장한다.

r_{15} : Program Counter(pc), Processor 가 읽어들이는 다음 Instruction 의 address 를 저장한다.

cpsr : Current Program Status Register. 내부 동작을 모니터링하고 제어하기 위해서 사용한다.

spsr : Saved Program Status Register

➤ <https://blog.naver.com/kojaejung/40108990614> 참조

리눅스 내의 ARM 환경 설정

`sudo apt-get update`

`sudo apt-get install qemu-user-static qemu-system` : qemu 환경을 마련하는 명령어이다.

`sudo apt-get install gcc-arm-linux-gnueabi` : arm 컴파일 환경을 마련하는 명령어이다.

이후에 C 소스 파일을 작성한다.

`arm-linux-gnueabi-gcc -g 소스파일`

`sudo apt-get install gdb-multiarch` : 디버깅할 수 있도록 해주는 환경 설정 명령어이다.

터미널을 2 개 띄운다.

A 터미널에서 아래 명령어를 수행한다.

`qemu-arm-static -g 1234 -L /usr/arm-linux-gnueabi ./a.out`

➤ 이 때, `-g 1234` 를 지우면 값을 확인할 수 있다.

위의 명령어를 입력하게 되면 A 터미널의 커서는 깜박거린다. 이 때, B 터미널에서 아래 명령어를 수행한다.

`gdb-multiarch` : 디버깅 진행이 가능하도록 해준다.

`file a.out`

```
target remote localhost:1234
b main
c
```

이후부터 디버깅을 진행하면 된다. 다른 모든 프로그램도 이러한 방법으로 디버깅을 수행할 수 있다.
이때, info reg 를 입력하면 레지스터 목록들이 나온다. l(L)은 소스를 보여준다.

학습 예제

- add, sub, rsb, and, bic, orr, eor, cmp, teq, tst, mov, mvn 명령어 실습

```
add r3, r1, r2 : r3 = r1 + r2
subgt r3, r1, r2 : (gt 는 접미사로 다른 의미를 가진다.) r3 = r1 - r2
rsble r3, r1, r2 : 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 : r3 = r1 ^ r2
cmp r1, r2 : r1 - r2 하여 비교 후 state flag 업데이트
teq r1, r2 : r1 ^ r2 하여 비교 후 조건 flag 업데이트
tsteq r1, r2 : r1 & r2 하여 비교 후 조건 flag 업데이트
mov r1, r2 : r1 = r2
mvn r1, r2 : r1 = 0xffffffff ^ r2
```

```
void show_reg(unsigned int reg)
{
    int i;

    for(i = 31; i >= 0;)
        printf("%d", (reg>>i--) & 1);
    printf("\n");
}
```

- 이 함수가 사용되는 것은 C 언어에서 2 진법으로 출력하는 방법이 없기 때문이다. 각 비트를 체크하여 2 진법으로 출력해준다.

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

```
int main(void)
```

```
{
```

```
    register unsigned int r0 asm("r0");
```

```
    register unsigned int r1 asm("r1");
```

```
    register unsigned int r2 asm("r2");
```

```
    r1 = 77;
```

```
    r2 = 37;
```

```
    asm volatile("add r0, r1, r2");
```

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

```
    return 0;
```

```
}
```

```
sue100012@sue100012-Z20NH-ASS1B5U:~/project/4/4_30$ qemu-arm-static -L /usr/arm-  
linux-gnueabi ./a.out  
r0 = 114
```

```
(gdb) info reg
```

r0	0x72	114
r1	0x4d	77
r2	0x25	37
r3	0x10120	65516

```

Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file a.out
Reading symbols from a.out...done.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
warning: remote target does not support file transfer, attempting to access file
s from local filesystem.
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
Breakpoint 1 at 0x10440: file add.c, line 9.
(gdb) c
Continuing.
warning: Could not load shared library symbols for 2 libraries, e.g. /lib/libc.s
o.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: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          0xf6ffff014   -150999020
r2          0xf6ffff01c   -150999012
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         0xf6ffeec4    -150999356
r12         0xf6ffef40    -150999232
sp          0xf6ffeec0    0xf6ffeec0
lr          0xf6686d14    -160928492
pc          0x10440      0x10440 <main+8>
cpsr       0x60000010     1610612752
(gdb) █

```

```
(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) l
4      {
5          register unsigned int r0 asm("r0");
6          register unsigned int r1 asm("r1");
7          register unsigned int r2 asm("r2");
8
9          r1 = 77;
10         r2 = 37;
11
12         asm volatile("add r0, r1, r2");
13
(gdb) █
```

```
#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;
}
```

```
sue100012@sue100012-Z20NH-AS51B5U:~/project/4/4_30$ qemu-arm-static -L /usr/arm-
linux-gnueabi ./a.out
r3 = 33
```

```
#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;

```

```

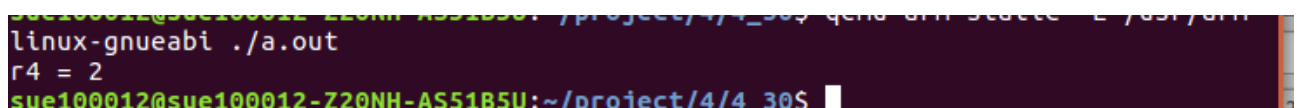
if(r2 <= r1)
    asm volatile("rsble r4, r5, #5");

```

```

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

```



```

sue100012@sue100012-720NH-AS51B5U:~/project/4/4_30$ ./a.out
linux-gnueabi ./a.out
r4 = 2
sue100012@sue100012-720NH-AS51B5U:~/project/4/4_30$

```

```

#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;
}

```

[illegible]

```
#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");
    }

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

[illegible]

➤ biceq 예제 풀이는 다음과 같다.

$$42 \& \sim(2^3 - 1)$$

42 & ~ (7)

42를 2^3 의 배수로 정렬 > 40

```
#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");

```

```

    r5 = 3;

```

```

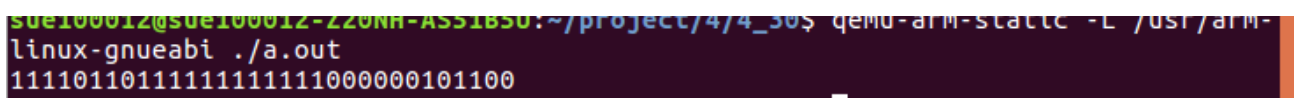
    if(r0 == r1)
    {
        r3 = 44;
        asm volatile("orr r2, r3, r5");
    }

```

```

    show_reg(r2);
    return 0;
}

```



```

sue100012@sue100012-220NH-ASS1850:~/project/4/4_30$ demo-arm-static -L /usr/arm-
linux-gnueabi ./a.out
11110110111111111111000000101100

```

```

#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");

```

```

    if(r0 == r1)
    {
        r0 = 10;
        r3 = 5;
        asm volatile("eors r1, r3, r0");
    }

```

```
}

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

```
linux-gnueabi ./a.out
11110110111111111111000000100100
```

```
#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");

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

    return 0;
}
```

```

r12      0xf6ffef50      -150999216
sp        0xf6ffeed4      0xf6ffeed4
lr        0xf6686d14      -160928492
pc        0x10408 0x10408 <main+8>
cpsr      0x60000010      1610612752
(gdb) n
13      asm volatile("mov r2, #5");
(gdb) disas
Dump of assembler code for function main:
    0x00010400 <+0>:      push    {r11}                ; (str r11, [sp, #-4]!)
    0x00010404 <+4>:      add     r11, sp, #0
    0x00010408 <+8>:      cmp     r0, r1
=> 0x0001040c <+12>:     mov     r2, #5
    0x00010410 <+16>:     cmp     r0, r2
    0x00010414 <+20>:     mov     r3, #0
    0x00010418 <+24>:     mov     r0, r3
    0x0001041c <+28>:     sub     sp, r11, #0
    0x00010420 <+32>:     pop     {r11}                ; (ldr r11, [sp], #4)
    0x00010424 <+36>:     bx      lr
End of assembler dump.
(gdb) info reg
r0          0x1          1
r1          0xf6fff024    -150999004
r2          0xf6fff02c    -150998996
r3          0x10400      66560
r4          0x10428      66600
r5          0x0          0
r6          0x102d8      66264
r7          0x0          0
r8          0x0          0
r9          0x0          0
r10         0xf67fe000    -159391744
r11         0xf6ffeed4    -150999340
r12         0xf6ffef50    -150999216
sp          0xf6ffeed4    0xf6ffeed4
lr          0xf6686d14    -160928492
pc          0x1040c 0x1040c <main+12>
cpsr        0x10         16

```

```

sp      0xf6ffeed4      0xf6ffeed4
lr      0xf6686d14      -160928492
pc      0x10410      0x10410 <main+16>
cpsr    0x10      16
(gdb) n
16      return 0;
(gdb) disas
Dump of assembler code for function main:
   0x00010400 <+0>:      push    {r11}                ; (str r11, [sp, #-4]!)
   0x00010404 <+4>:      add     r11, sp, #0
   0x00010408 <+8>:      cmp     r0, r1
   0x0001040c <+12>:     mov     r2, #5
   0x00010410 <+16>:     cmp     r0, r2
=>  0x00010414 <+20>:     mov     r3, #0
   0x00010418 <+24>:     mov     r0, r3
   0x0001041c <+28>:     sub     sp, r11, #0
   0x00010420 <+32>:     pop     {r11}                ; (ldr r11, [sp], #4)
   0x00010424 <+36>:     bx      lr
End of assembler dump.
(gdb) info reg
r0      0x1      1
r1      0xf6fff024      -150999004
r2      0x5      5
r3      0x10400      66560
r4      0x10428      66600
r5      0x0      0
r6      0x102d8      66264
r7      0x0      0
r8      0x0      0
r9      0x0      0
r10     0xf67fe000      -159391744
r11     0xf6ffeed4      -150999340
r12     0xf6ffef50      -150999216
sp      0xf6ffeed4      0xf6ffeed4
lr      0xf6686d14      -160928492
pc      0x10414      0x10414 <main+20>
cpsr    0x80000010      -2147483632
(gdb)

```

- cmp 후에 cpsr 의 값이 변경되는 것을 확인할 수 있다.

```
#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");
}

```

```

asm volatile("cmp r0, r1");
asm volatile("mov r2, #3");
asm volatile("tsteq r2, #5");

```

```

return 0;
}

```

```

sp      0xf6fffeed4      0xf6fffeed4
lr      0xf6686d14      -160928492
pc      0x10410      0x10410 <main+16>
cpsr    0x10      16
(gdb) n
16      return 0;
(gdb) disas
Dump of assembler code for function main:
0x00010400 <+0>:      push    {r11}          ; (str r11, [sp, #-4]!)
0x00010404 <+4>:      add     r11, sp, #0
0x00010408 <+8>:      cmp     r0, r1
0x0001040c <+12>:     mov     r2, #5
0x00010410 <+16>:     cmp     r0, r2
=> 0x00010414 <+20>:     mov     r3, #0
0x00010418 <+24>:     mov     r0, r3
0x0001041c <+28>:     sub     sp, r11, #0
0x00010420 <+32>:     pop     {r11}          ; (ldr r11, [sp], #4)
0x00010424 <+36>:     bx      lr
End of assembler dump.
(gdb) info reg
r0      0x1      1
r1      0xf6fff024      -150999004
r2      0x5      5
r3      0x10400      66560
r4      0x10428      66600
r5      0x0      0
r6      0x102d8      66264
r7      0x0      0
r8      0x0      0
r9      0x0      0
r10     0xf67fe000      -159391744
r11     0xf6fffeed4      -150999340
r12     0xf6ffef50      -150999216
sp      0xf6fffeed4      0xf6fffeed4
lr      0xf6686d14      -160928492
pc      0x10414      0x10414 <main+20>
cpsr    0x80000010      -2147483632
(gdb)

```

```

#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;
```

```
}
```

```
sue100012@sue100012-Z20NH-A551B5U:~/project/4/4_30$ demo-arm-static -L /usr/arm-
linux-gnueabi ./a.out
r1 = 0xffffffff
sue100012@sue100012-Z20NH-A551B5U:~/project/4/4_30$
```