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 assembly

< 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

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 // 같다.

ne : not equal

b: byte

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

*** ! 느낌표 들어간 문장 이해하기

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

arm 어셈블리 함수 오늘 추가 분

***mov r0, #0xff, 8 : lotation shift**
→ arm은 32bit 단위이다. 0Xff = 0x00 00 00 ff
→ 우측 8비트 시프트 시킨다. 0x 00 00 00 00 //ff 돌아서 상위 비트로 물린다. 즉 0x ff 00 00 00 00

***r1, lsl r2 : logical shift leg**
→ r1 * 2^r2

***r1, asr , r2**
→ r1 / 2^r2

***mul r1,r2,r3**
→ r1 = r2 * r2

***mla r1,r2,r3,r4 : dsp는 곱셈과 덧셈을 동시에 진행 할 수 있다. r2와 r3를 곱하고 거기에 r4를 더해라. 이 모두가 1 clock에 끝난다.**
→ r1 = r2 * r3 + r4

***umull r0,r1,r2,r3 : Unsigned Long 곱하기, 32 비트 피연산자 및 64 비트 결과**
→ r1 r0 = r2*r3

***umlal r0,r1,r2,r3**
→ r1 r0 = r2*r3 + (r0, r1) → r1 = 상위비트 8bit+ r1, r0 = 하위8비트 + r0

*ldr r0,[r1,#0x4] : load 레지스터 (reg → mem)
→ r0 = *(r1 + 4byte)

*ldreqb r0,[r1,#0x5] : eq 는 z플래그 확인
→ r0 = *(r1 + 5 byte)

*ldr r0, [r1],r2 : r1은 주소이다. r2는 byte크기이다.
→ r0 = *r1 , r1 = r1 +r2

*str : store 레지스터 (mem → reg)

*strb r0,[r1] : b는 byte
→ *r1 = r0

*stmia r0,{r1,r2,r3} : store multiful , r0 memory 에 r1, r2, r3 을 순서대로 집어 넣어라. r0은 포인터(memory)

→ r0 = r1

*mrs r0 cpsr : cpsr, sqrs의 값을 arm 의 범용 레지스터로 읽어 온다.
→ r0= cpsr , 특수 인터럽트 플래그 비트 레지스터에 접근이 가능하다. msr은 그 반대

*msr cpsr r0 : arm의 값을 cpsr, sqrs 의 레지스터에 쓴다.
→ cpsr = r0 , r0값을 cpsr reg로 전달

mrs 없으면 cpsr값에 접근할 수 없다. Cpsr레지를 mrs와 같은 별도 명령어로 관리를한다. → 키보드나 마우스를 사용도 인터럽트인데, 사용자가 임의로 cpsr레지를 건드리면 인터럽트 반응이 안되기 때문에 아무나 움직이지 못하도록 운영체제가 관리하는 별도의 instruction을 만들어 둔것.

***어셈블리 사용하는 이유??**
하나의 예로 인터럽트 껐다 켰다 하는 것은 운영체제가 한다. 이때 사용하는 프로그래밍은 어셈블리로만 가능하다.

***ldr , *str**

어셈블리어 레벨에서 볼 때 **load store register는** 핵심이다. **load store register 가** 존재하는 이유가 뭘까?

intel은 memory to memory 연산이 가능하다. arm은 불가능하기 때문에 memory에 있는 정보를 레지스터에 가져오고 다시 메모리에 집어 넣어야 한다.

→ load는 레지스터에 가져오는 오는 것이고, store는 레지스터에 때려 넣는 것이다. Ok?

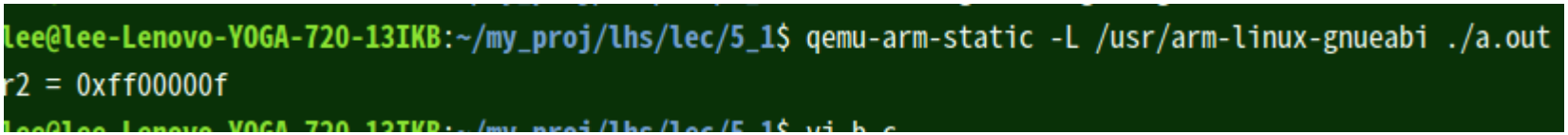
mov : 같다. 복사

```
#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("mov r0, #0xff, 8");   // (r0 = (0xff  >> 8) = 0xff000000   →  0xff >> 8 하면 0x00 이 되지만, 사라지는게 아니라 32비트 맨 앞으로 온다고 생각.

   printf("r0 = 0x%x\n",r0);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0 = 0xff000000
```

Add : 더하기

```
#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("mov r0, #0xff, 8"); //  r0 = 0x ff 00 00 00
   asm volatile("mov r1, #0xf");     //  r1 =  0x 00 00 00 0f
   asm volatile("add r2,r1,r0");     //  r2 =   r0+r1 = 0x ff 00 00 0f

   printf("r2 = 0x%x\n",r2);

   return 0;
}
```



lsl1 : 2^x 승수 곱하기

```
#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("mov r1, #7"); // r1 = 7
   asm volatile("mov r2, #3"); // r2 = 3  // lsl = logical shift leg
   asm volatile("add r0,r1,r2,lsl #7");    // add r0,r1,r2,lsl #7  →  r0 = r1 + (r2 lsl #7) = 101  + 11 00 00 00 0  = 110000101

   printf("r0 = 0x%x\n",r0);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0 = 0x187
```

lsl2

```
#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("mov r1, #7");   // r1 = 7
   asm volatile("mov r2, #3");   // r2 = 3
   asm volatile("mov r3, #2");   // r3 = 2
   asm volatile("add r0,r1,r2,lsl r3");  //  7+3*2^2 = 19

   printf("r0 = 0x%x\n",r0);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0 = 0x13
```

lsl3

```
#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("mov r1, #2");
   asm volatile("add r0,r1,r1,lsl #2"); // r0 = 2 + 2 * 2^2 = 10

   printf("r0 = 0x%x\n",r0);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0 = 0xa
```

asr : 2^x 승수 나누기

```
#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("mov r1, #32");
   asm volatile("add r0,r1,asr #2"); // r0 = 32 /  2^2

   printf("r0 = 0x%x\n",r0);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0 = 0x8
```

asr2

```
#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("mov r1, #32");
   asm volatile("add r0,r1,asr #2"); // r0 = 32 / 2^2 = 8
   asm volatile("mrs r0,cpsr");
   show_reg(r0);
   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
01100000000000000000000000010000
```

: 현재 레지스터들 값이 나온다.
17분

P/t $cpsr

p/x $cpsr

n

p r0

p/x r0

cpsr레지는
mrs 가 하는 역할은 cpsr 레지의 값을 r0레지에 집어넣는 일을 했다. 특수한 레지스터에 접근이 가능하다.
이때 떠올라야 하는 일 : Pushf 와 같다는 것. 스택에 집어 넣는 일과 같은 일을함.)

mrs : 인터럽트를 껐다 켰다 할 때 사용할 수 있음.

```
(gdb) si
22          asm volatile("mrs r0,cpsr");
(gdb) info reg
r0             0x8      8
r1             0x20     32
r2             0x0      0
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            0xf6ffef34       -150999244
r12            0xf6ffefb0       -150999120
sp             0xf6ffef28       0xf6ffef28
lr             0xf667cd14       -160969452
pc             0x104f4  0x104f4 <main+40>
cpsr           0x60000010       1610612752
(gdb) si
23          show_reg(r0);
(gdb) info reg
r0             0x60000010       1610612752
r1             0x20     32
r2             0x0      0
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            0xf6ffef34       -150999244
r12            0xf6ffefb0       -150999120
sp             0xf6ffef28       0xf6ffef28
lr             0xf667cd14       -160969452
pc             0x104f8  0x104f8 <main+44>
cpsr           0x60000010       1610612752
```

***mul : 곱하기 → dsp명령어로 사용**

```
#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("mov r2, #3");
   asm volatile("mov r3, #7");
   asm volatile("mul r1,r2,r3"); //r1 = r2*r3

   printf("r1 = %d\n", r1);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r1 = 21
```

*mla : dsp는 곱셈과 덧셈을 동시에 진행 할 수 있다. r2와 r3를 곱하고 거기에 r4를 더해라 → 1 clock에 끝난다.

```
#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("mov r2, #3");
   asm volatile("mov r3, #7");
   asm volatile("mov r4, #33");
   asm volatile("mla r1,r2,r3,r4");  // r1= r2*r3 + r4

   printf("r1 = %d\n", r1);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r1 = 54
```

***umull** : Unsigned Long 곱하기, 32 비트 피연산자 및 64 비트 결과

```
#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;  // 0x 00 00 00 ff
   asm volatile("mov r2, #0x44,8");       //r2 = 0x44 <<8 ( 00 00 00 00) =   100 0100 00 00 00 00
   asm volatile("mov r3, #0x200");        //r3 =  0x200                      =         0010 00 00 00 00
   asm volatile("umull r0,r1,r2,r3");     // 32비트인 것을 두개 곱함. 32비트를 넘어가니 두개 r0 r1 레지로 표현한것.

   printf("r1r0 = 0x%x%08x\n",r1,r0);
//
   return 0;
```

```
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r1r0 = 0x880
```

*umlal : mla 는 곱하고 더하기

```
#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("mov r0, #0xf");
   asm volatile("mov r1, #0x1");
   asm volatile("mov r2, #0x44,8"); //
   asm volatile("mov r3, #0x200");
   asm volatile("umlal r0,r1,r2,r3"); //  r1 이 상위 r0이 하위, 즉= 44 *2 = 0x8800  r1은 88+ 1이고 ,  r0 는 00+ f

   printf("r1r0 = 0x%x%x\n",r1,r0);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r1r0 = 0x89f
```

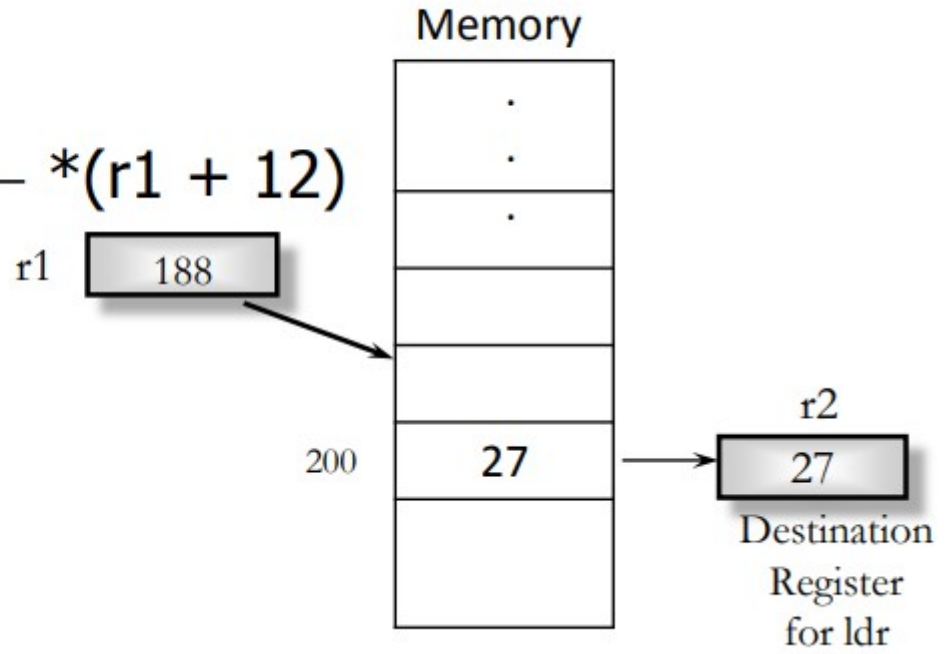
***ldr : load reg, memory → register**

```
#include <stdio.h>

unsigned int arr[5] = {1,2,3,4,5};

int main(void)
{
   register unsigned int r0 asm("r0") = 0;
   register unsigned int *r1 asm("r1") = NULL;
   register unsigned int *r2 asm("r2") = NULL;
   register unsigned int r3 asm("r3") = 0;
   register unsigned int r4 asm("r4") = 0;
   register unsigned int r5 asm("r5") = 0;

    r1 = arr;   // 배열의 주소값 셋팅

   asm volatile("mov r2, #0x8"); // r2=8
   asm volatile("ldr r0,[r1,r2]"); //[r1,r2]  r1은 시작주소이다. r1에서 8바이트 이동하면 3이다, 이제 3 을 r0로 가져오라는 뜻. \

   printf("f0 = %u\n",r0);   //r0 오타임.
   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
f0 = 3
```

Example:

ldr r2, [r1, #12] @ r2 ← *(r1 + 12)



ldr r2, [r1, #12]! @ r1 ← r1 + 12 then r2 ← *r1

ldr r2, [r0, r1]! @ r0 ← r0 + r1 then r2 ← *r0

ldr r2, [r0, r1, lsl #2]! @ r0 ← r0 + r1*4 then r2 ← *r0

*ldr2

```
#include <stdio.h>

unsigned int arr[5] = {1,2,3,4,5};

int main(void)
{
   register unsigned int r0 asm("r0") = 0;
   register unsigned int *r1 asm("r1") = NULL;
   register unsigned int *r2 asm("r2") = NULL;
   register unsigned int r3 asm("r3") = 0;
   register unsigned int r4 asm("r4") = 0;
   register unsigned int r5 asm("r5") = 0;

    r1 = arr;  //r1 =  arr의 시작주소

   asm volatile("ldr r0,[r1,#0x4]"); // r0 = *(r1  + 4 byte)

   printf("f0 = %u\n",r0);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
f0 = 2
```

***ldreqb: eq 는 z플래그 확인(사실 초기에 z 0으로 초기화 되었기 때문에 그냥 쓴것 b는 byte임.**

```
#include <stdio.h>

char test[] = "HelloARM";

int main(void)
{
   register unsigned int r0 asm("r0") = 0;
   register char *r1 asm("r1") = NULL;
   register unsigned int *r2 asm("r2") = NULL;
   register unsigned int r3 asm("r3") = 0;
   register unsigned int r4 asm("r4") = 0;
   register unsigned int r5 asm("r5") = 0;

    r1 = test;

   asm volatile("ldreqb r0,[r1,#0x5]");

   printf("f0 = %c\n",r0);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
f0 = A
```

***strb : ldr의 반대 즉, register→ memory**

```
#include <stdio.h>

char test[] = "HelloARM";

int main(void)
{
   register unsigned int r0 asm("r0") = 0;
   register char *r1 asm("r1") = NULL;
   register unsigned int *r2 asm("r2") = NULL;
   register unsigned int r3 asm("r3") = 0;
   register unsigned int r4 asm("r4") = 0;
   register unsigned int r5 asm("r5") = 0;

   r1 = &test[5];

   asm volatile("mov r0, #61");   // 61이 ASCII 코드로  =  임.
   asm volatile("strb r0,[r1]");   // *r1 = r0

   printf("test = %s\n",test);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test = Hello=RM
```

***위의 strb과 ldr 비교해보기**

ldr4.c

```
#include <stdio.h>

char test[] = "HelloARM";

int main(void)
{
   register unsigned int r0 asm("r0") = 0;
   register char *r1 asm("r1") = NULL;
   register unsigned int *r2 asm("r2") = NULL;
   register unsigned int r3 asm("r3") = 0;
   register unsigned int r4 asm("r4") = 0;
   register unsigned int r5 asm("r5") = 0;

   r1 = test;

   asm volatile("mov r2,#0x5");
   asm volatile("ldr r0, [r1,r2]!");   // !가 붙으면, r1에 r2 만큼 이동한 곳에 r1을 픽스 시키라는 뜻이다.

   printf("test = %s, r1 = %s\n",test,r1);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test = HelloARM, r1 = ARM
```

*ldr5

```
#include <stdio.h>

unsigned int arr[5] = {1,2,3,4,5};

int main(void)
{
   register unsigned int r0 asm("r0") = 0;
   register unsigned int *r1 asm("r1") = NULL;
   register unsigned int *r2 asm("r2") = NULL;
   register unsigned int r3 asm("r3") = 0;
   register unsigned int r4 asm("r4") = 0;
   register unsigned int r5 asm("r5") = 0;

   r1 = arr;  // r1에는 arr의 시작주소가 들어간다.

   asm volatile("mov r2,#0x4");
   asm volatile("ldr r0, [r1],r2");    // 따로 동작함. r0에는 r1의 시작주소가 들어간다. 즉 1이 들어가는 것이고, r1에는 r1의 4byte지난 것이 들어간다.

   printf("r0 = %u, r1 = %u\n",r0,*r1);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
r0 = 1, r1 = 2
```

***stmia : reg → mem , stack**

```
#include <stdio.h>

int main(void)
{
   int i;
   unsigned int test_arr[5] = {0};

   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;

   r0 = test_arr;  // r1에는 arr의 시작주소가 들어간다.

   asm volatile("mov r1,#0x3"); // r1=3
   asm volatile("mov r2,r1,lsl #2");  // r2에 r1*2^2 =12
   asm volatile("mov r4, #0x2"); // r4= 2
   asm volatile("add r3,r1,r2,lsl r4"); //   r3= 3+ 48 = 51
   asm volatile("stmia r0,{r1,r2,r3}"); // store multiful : store = r0 memory 에 r1, r2, r3 을 순서대로 집어 넣어라.

   for(i=0;i<5;i++)
   printf("test_arr[%d] = %d\n",i,test_arr[i]);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test_arr[0] = 3
test_arr[1] = 12
test_arr[2] = 51
test_arr[3] = 0
test_arr[4] = 0
```

***stmia2:**

```
#include <stdio.h>     //→ 1_2.3 녹음 30분

int main(void)
{
   int i;
   unsigned int test_arr[5] = {0};

   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;

   r0 = test_arr;  // r1에는 arr의 시작주소가 들어간다.

   asm volatile("mov r1,#0x3"); // r1=3
   asm volatile("mov r2,r1,lsl #2");  // r2에 r1*2^2 =12
   asm volatile("mov r4, #0x2"); // r4= 48
   asm volatile("add r3,r1,r2,lsl r4"); // r3에  값을 증가 후에 값을 넣어라
   asm volatile("stmia r0!,{r1,r2,r3}"); // store multiful : store = reg 에있는 것을 memory로
   asm volatile("str r4, [r0]");

   for(i=0;i<5;i++)
   printf("test_arr[%d] = %d\n",i,test_arr[i]);

   return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test_arr[0] = 3
test_arr[1] = 12
test_arr[2] = 51
test_arr[3] = 2
test_arr[4] = 0
```

***stmia2 위와 같은 코드 volatile선언 여러번사용 안함.**

```
int main(void)
{
   int i;
   unsigned int test_arr[5] = {0};

   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;

   r0 = test_arr;  // r1에는 arr의 시작주소가 들어간다.

   asm volatile("mov r1,#0x3\n"
            "mov r2,r1,lsl #2\n"
            "mov r4,#0x2\n"
            "add r3,r1,r2,lsl r4\n"
```

```
        "stmia r0!,{r1,r2,r3}\n"
        "str r4,[r0]");
  for(i=0;i<5;i++)
  printf("test_arr[%d] = %d\n",i,test_arr[i]);

  return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test_arr[0] = 3
test_arr[1] = 12
test_arr[2] = 51
test_arr[3] = 2
test_arr[4] = 0
```

*

```
#include <stdio.h>

int main(void)
{
   int i;
   unsigned int test_arr[5] = {0};

   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;
   register unsigned int r6 asm("r6") = 0;

   r0 = test_arr;  // r1에는 arr의 시작주소가 들어간다.

   asm volatile("mov r1,#0x3\n"
            "mov r2,r1,lsl #2\n"
            "mov r4,#0x2\n"
            "add r3,r1,r2,lsl r4\n"
            "stmia r0!,{r1,r2,r3}\n"
            "str r4,[r0]\n"
            "mov r5, #128\n"
            "mov r6,r5,lsl #3\n"
            "stmia r0, {r4,r5,r6}\n"
            "sub r0,r0,#12\n"
            "ldmia r0,{r4,r5,r6}");
   for(i=0;i<7;i++)
```

```
    printf("test_arr[%d] = %d\n",i,test_arr[i]);

    printf("r4 = %u, r5 = %u, r6 = %u\n",r4,r5,r6);
    return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/lec/5_1$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test_arr[0] = 3
test_arr[1] = 12
test_arr[2] = 51
test_arr[3] = 2
test_arr[4] = 128
test_arr[5] = 1024
test_arr[6] = 66472
r4 = 3, r5 = 12, r6 = 51
```

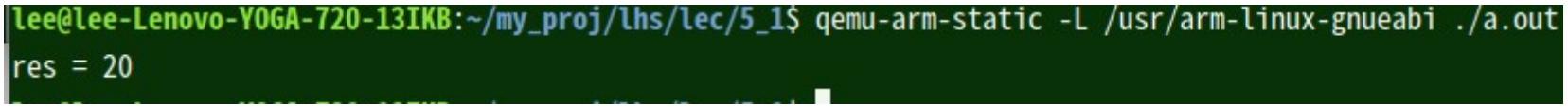
: 44분 몇초부터 시작했음…

***func**

```
#include <stdio.h>

int my_func(int num)
{
   return num*2;
}

int main(void)
{
   int res, num = 2;
   res = my_func(num);
   printf("res=%d\n",res);
   return 0;
}
```

***func2, 1시간 52 분**

```
#include <stdio.h>

int my_func(int n1,int n2,int n3, int n4,int n5)
{
   return n1 + n2 + n3 + n4 + n5;
}

int main(void)
{
   int res, n1 = 2, n2 = 3, n3 = 4, n4 = 5, n5 = 6;
   res = my_func(n1,n2,n3,n4,n5);
   printf("res = %d\n",res);
   return 0;
}
```



R11는 bp 시작주소임

lr 은 복귀주소 2시간 5분

arm , 펌웨어
공업수학
신호처리
dsp
fpga

내일은 어셈블리로 시스템콜 써보기.

데이터시트 보는방법 → 이를 토대로 펌웨어 부팅되는 방법.