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

#45

강사 : Innova Lee(이 상훈)

학생 : 김 시윤

1.배운내용 복습.

ARM 입문

Calling Convention

Register	APCS	Comment
r0	a1	Function Argument 1, Returnn Value, GP Scratch Register
r1	a2	Function Argument 2, GP Scratch Register
r2	a3	Function Argument3, GP Scratch Register
r3	a4	Function Argument 4, GP Scratch Register
r4	v1	Variable Register 1
r5	v2	Variable Register 2
r6	v3	Variable Register 3
r7	v4	Variable Register 4, Systemcall
r8	v5	Variable Register 5
r9	v6/sb	Variable Register 6
r10	v7/sl	Variable Register 7, Stack의 Limit Address 저장
r11	fp	이전 버전의 ARM Compiler에서 Frame Pointer
r12	ip	GP Scratch Register
r13	sp	Stack의 현재 위치를 저장하는 Register
r14	lr	Link Registerr 복귀할 주소를 저장하는 Register. Scratch Register
r15	pc	Program Counter

function.c

```
#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;
}
```

```
-> 0x00010460 <+0>:  push   {r11, lr}
   0x00010464 <+4>:  add    r11, sp, #4
   0x00010468 <+8>:  sub    sp, sp, #8
   0x0001046c <+12>: mov    r3, #2
   0x00010470 <+16>: str    r3, [r11, #-12]
   0x00010474 <+20>: ldr    r0, [r11, #-12]
   0x00010478 <+24>: bl     0x10438 <my_func>
   0x0001047c <+28>: str    r0, [r11, #-8]
   0x00010480 <+32>: ldr    r1, [r11, #-8]
   0x00010484 <+36>: ldr    r0, [pc, #16]   ; 0x1049c <main+60>
   0x00010488 <+40>: bl     0x102e0 <printf@plt>
```

```
   0x0001048c <+44>: mov    r3, #0
   0x00010490 <+48>: mov    r0, r3
   0x00010494 <+52>: sub    sp, r11, #4
   0x00010498 <+56>: pop    {r11, pc}
   0x0001049c <+60>: andeq  r0, r1, r0, lsl r5
info reg
sp = 0xf6ffee78

   0x00010460 <+0>:  push   {r11, lr}
-> 0x00010464 <+4>:  add    r11, sp, #4
   0x00010468 <+8>:  sub    sp, sp, #8
   0x0001046c <+12>: mov    r3, #2
   0x00010470 <+16>: str    r3, [r11, #-12]
   0x00010474 <+20>: ldr    r0, [r11, #-12]
   0x00010478 <+24>: bl     0x10438 <my_func>
   0x0001047c <+28>: str    r0, [r11, #-8]
   0x00010480 <+32>: ldr    r1, [r11, #-8]
   0x00010484 <+36>: ldr    r0, [pc, #16]   ; 0x1049c <main+60>
   0x00010488 <+40>: bl     0x102e0 <printf@plt>
   0x0001048c <+44>: mov    r3, #0
   0x00010490 <+48>: mov    r0, r3
   0x00010494 <+52>: sub    sp, r11, #4
   0x00010498 <+56>: pop    {r11, pc}
   0x0001049c <+60>: andeq  r0, r1, r0, lsl r5

sp = 0xf6ffee70

push r11,lr 2개 8byte
sp + 4byte = r11
```

```
r11위치에 sp + 4byte 넣어줌 0xf6ffee74

r11             0xf6ffee74

sub    sp, sp, #8

스택 공간 4바이트 할당
r11 이 bp라 했을때
r11은 sp +4 이므로 sp -8 은 4바이트 할당한것과 같다.

mov    r3, #2
str    r3, [r11, #-12]

r3에 2를 넣고  r3을 r11 - 12 위치 즉 sp에 집어넣은

sp = 0xf6ffee68
r11 =  0xf6ffee74
r11 - sp = 12 이므로 위 식이 맞음을 증명

(gdb) p/x *0xf6ffee68
$1 = 0x2

확인해 보면 맞다

ldr    r0, [r11, #-12]

sp에 있는 2를 r0에 넣음

(gdb) p/x $r0
$2 = 0x2
```

```
bl      0x10438 <my_func>

my_func 함수로 콜!!
복귀주소는 lr에 저장

lr              0x1047c

push    {r11}           ; (str r11, [sp, #-4]!)
add     r11, sp, #0

sp 4byte 공간에 r11 push
sp 를 r11에 저장

r11             0xf6ffee64
sp              0xf6ffee64

0x00010444 <+12>:    str     r0, [r11, #-8]

(gdb) p/x *0xf6ffee5c
$11 = 0x2

ldr     r3, [r11, #-8]

2를 r3에 넣음

lsl     r3, r3, #1

왼쪽으로 1바이트 쉬프트 즉 r3 *2
그래서 r3 =4
```

```
mov     r0, r3

r3 을 r0에 저장 (함수의 리턴 은 r0담당)

복귀

str     r0, [r11, #-8]

r0을 r11 -8byte 위치에 저장

ldr     r1, [r11, #-8]

r1에 다시 옮김

그리고 출력
```

function2.c

```c
#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;
}
```

```
   0x00010488 <+0>:  push   {r11, lr}
   0x0001048c <+4>:  add    r11, sp, #4
   0x00010490 <+8>:  sub    sp, sp, #32
=> 0x00010494 <+12>: mov    r3, #2
   0x00010498 <+16>: str    r3, [r11, #-28]; 0xffffffe4
   0x0001049c <+20>: mov    r3, #3
   0x000104a0 <+24>: str    r3, [r11, #-24]; 0xffffffe8
   0x000104a4 <+28>: mov    r3, #4
   0x000104a8 <+32>: str    r3, [r11, #-20]; 0xffffffec
   0x000104ac <+36>: mov    r3, #5
   0x000104b0 <+40>: str    r3, [r11, #-16]
   0x000104b4 <+44>: mov    r3, #6
   0x000104b8 <+48>: str    r3, [r11, #-12]
   0x000104bc <+52>: ldr    r3, [r11, #-12]
   0x000104c0 <+56>: str    r3, [sp]
   0x000104c4 <+60>: ldr    r3, [r11, #-16]
   0x000104c8 <+64>: ldr    r2, [r11, #-20]; 0xffffffec
   0x000104cc <+68>: ldr    r1, [r11, #-24]; 0xffffffe8
   0x000104d0 <+72>: ldr    r0, [r11, #-28]; 0xffffffe4
   0x000104d4 <+76>: bl     0x10438 <my_func>
   0x000104d8 <+80>: str    r0, [r11, #-8]
   0x000104dc <+84>: ldr    r1, [r11, #-8]
   0x000104e0 <+88>: ldr    r0, [pc, #16]   ; 0x104f8 <main+112>
   0x000104e4 <+92>: bl     0x102e0 <printf@plt>
   0x000104e8 <+96>: mov    r3, #0
   0x000104ec <+100>:        mov    r0, r3
   0x000104f0 <+104>:sub    sp, r11, #4
   0x000104f4 <+108>:pop    {r11, pc}
   0x000104f8 <+112>:andeq  r0, r1, r12, ror #10
```

```
r3, #2
r3, [r11, #-28]; 0xffffffe4
sp -32 위치에 2를 저장
3, 4, 5, 6 까지 4바이트에 한번씩 저장한다.
근데 4번째 변수를 초과하여 5번째 변수를 저장할때는
레지스터가 아닌 메모리를 활용한다.
sp 에 r3을 저장하고 함수 안에서
        ldr     r3, [r11, #4]
그 저장했던 메모리로 접근하여 6이라는 값을 빼낸 후에 더하는 작업을 실행한다.

이렇게 레지스터가 아닌 메모리를 이용하는 경우에 속도가 느려진다는 단점이 있다.

그래서 변수가 4개가 넘어가게 되버리면 구조체 4개를 전달하는 방법으로 대체한다.
```

```
=> 0x00010438 <+0>:  push   {r11}            ; (str r11, [sp, #-4]!)
   0x0001043c <+4>:  add    r11, sp, #0
   0x00010440 <+8>:  sub    sp, sp, #20
   0x00010444 <+12>: str    r0, [r11, #-8]
   0x00010448 <+16>: str    r1, [r11, #-12]
   0x0001044c <+20>: str    r2, [r11, #-16]
   0x00010450 <+24>: str    r3, [r11, #-20]; 0xffffffec
   0x00010454 <+28>: ldr    r2, [r11, #-8]
   0x00010458 <+32>: ldr    r3, [r11, #-12]
   0x0001045c <+36>: add    r2, r2, r3
   0x00010460 <+40>: ldr    r3, [r11, #-16]
   0x00010464 <+44>: add    r2, r2, r3
   0x00010468 <+48>: ldr    r3, [r11, #-20]; 0xffffffec
```

```
   0x0001046c <+52>: add     r2, r2, r3
   0x00010470 <+56>: ldr     r3, [r11, #4]
   0x00010474 <+60>: add     r3, r2, r3
   0x00010478 <+64>: mov     r0, r3
   0x0001047c <+68>: sub     sp, r11, #0
   0x00010480 <+72>: pop     {r11}              ; (ldr r11, [sp], #4)
   0x00010484 <+76>: bx      lr
```

위의 두 작업을 디버깅 하기 위해 간단한 소스코드 예제로 확인해보았었다.

---

add_mov.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("mov r0,#0xff,8");
        asm volatile("mov r1,#0xf");
        asm volatile("add r2,r1,r0");
        printf("r2 = 0x%x\n",r2);
        return 0;
}
```

asr.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("mov r1,#32");
        asm volatile("add r0,r1,asr #2");
        //right  shift
        printf("r0 = 0x%x\nr0= %d\n",r0,r0);
        return 0;
}
```

cpsr.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("mov r1,#32");
        asm volatile("add r0,r1,asr #2");
        asm volatile("mrs r0, cpsr");
        //pushf
        //중요
        show_reg(r0);
        return 0;
}
```

ldmia.c

```
#include <stdio.h>
int main(void)
{       int i;
        unsigned int test_arr[7] = {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;
                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, lsr #3\n"
                        "stmia r0, {r4,r5,r6}\n"
                        "sub r0, r0, #12\n"
                        "ldmia r0, {r4, r5, r6}");
                /* multiful load -> memory 에 있는걸 register 에 옮김
                sub r0 는 r0를 원래위치로*/
        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;
}
```

ldr.c

```
#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;
        asm volatile("mov r2,#0x8");
        asm volatile("ldr r0,[r1,r2]");
        printf("r0 = %u\n",r0);
        return 0;
```

```
}/*r1 의 주소에서 8 바이트 이동해라 */
```

ldr2.c

```
#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;
        asm volatile("ldr r0,[r1,#0x4]");
        printf("r0 = %u\n",r0);
        return 0;
}/*r1 의 주소에서 8 바이트 이동해라 */
```

ldr3.c

```
#include <stdio.h>
/* char *test */
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;
                /*b = 1byte 씩 처리하겠다 */
                asm volatile("ldreqb r0,[r1,#0x5]");
        printf("r0 = %c\n",r0);
        return 0;
}/*r1 의 주소에서 8 바이트 이동해라 */
```

ldr4.c

```
#include <stdio.h>
/* char *test */
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 5바이트 이동해라 셋커서 느낌
        */
        printf("test = %s,r1 = %s\n",test,r1);
        return 0;
}/* str = register to memory */
```

ldr5.c

```
#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;
                asm volatile("mov r2,#0x4");
        asm volatile("ldr r0,[r1],r2");
                /* r1 은 배열의 시작주소 그래서 1 이 r0에 들어감
        r2는 4 즉 4바이트 뒤에 있는 녀석의 주소가 r1에 들어감 */
                printf("r0 = %u,r1= %u\n",r0,*r1);
        return 0;
}
```

lsl.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("mov r1,#7");
        asm volatile("mov r2,#3");
        asm volatile("add r0,r1,r2,lsl #7");
        //logical left shift
        printf("r0 = 0x%x\n",r0);
        return 0;
}
```

lsl2.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("mov r1,#7");
        asm volatile("mov r2,#3");
        asm volatile("mov r3,#2");
        asm volatile("add r0,r1,r2,lsl r3");
        //logical left shift
        printf("r0 = 0x%x\n",r0);
        return 0;
}
```

lsl3.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("mov r1,#2");
	asm volatile("add r0,r1,r1,lsl #2");
	//logical left shift
	printf("r0 = 0x%x\nr0= %d\n",r0,r0);
	return 0;
}
```

mla.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("mov r2,#3");
	asm volatile("mov r3,#7");
	asm volatile("mov r4,#33");
	asm volatile("mla r1,r2,r3,r4");
		/* mla = multi add
		r2*r3+r4
	1 clock 에 끝남
	*/
	printf("r1 = %d\n",r1);
	return 0;
}
```

mov.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("mov r0,#0xff,8");
	/* 8 = 버림없이 쉬프트 하라.
```

```
        0xff = 0x00 00 00 ff
        = 0xff 00 00 00*/
        printf("r0 = 0x%x\n",r0);
        return 0;
}
```

mul.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("mov r2,#3");
        asm volatile("add r3,#7");
        asm volatile("mul r1,r2,r3");
        printf("r1 = %d\n",r1);
        return 0;
}
```

stmia.c

```
#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;
                asm volatile("mov r1,#0x3");
        asm volatile("mov r2,r1,lsl #2");
        asm volatile("mov r4,#0x2");
        asm volatile("add r3,r1,r2,lsl r4");
        asm volatile("stmia r0,{r1,r2,r3}");
        /* store multiful increement? after?
        먼저 증가시키고 값을 넣어라 3,  3을 쉬프트 해서 12 가들어가고
        51 */
                for(i = 0; i<5; i++)
        printf("test_arr[%d] = %d\n",i,test_arr[i]);
        return 0;
}
```

stmia2.c

```
#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;
                asm volatile("mov r1,#0x3");
        asm volatile("mov r2,r1,lsl #2");
        asm volatile("mov r4,#0x2");
        asm volatile("add r3,r1,r2,lsl r4");
        asm volatile("stmia r0!,{r1,r2,r3}");
        asm volatile("str r4, [r0]");
        /* ! 는 셋커서 느낌이다. */
        for(i = 0; i<5; i++)
        printf("test_arr[%d] = %d\n",i,test_arr[i]);
        return 0;
}
```

stmia3.c

```
#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;
                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;
}
```

strb.c

```
#include <stdio.h>
/* char *test */
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];          //r1 = test;
                asm volatile("mov r0,#61");
        asm volatile("strb r0,[r1]");       //strb r0, [r1,#5]
        printf("test = %s\n",test);
        return 0;
}/* str = register to memory */
```

umlal.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("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");
        printf("r1r0 = 0x%x%x\n",r1,r0);
        return 0;
}/* 16 진수 곱셈 */
/*곱한다음 상위끼리 더하고 하위끼리 더한다 */
```

umull..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("mov r2,#0x44,8");
        asm volatile("mov r3,#0x200");
        asm volatile("umull r0,r1,r2,r3");
        printf("r1r0 = 0x%x%08x\n",r1,r0);
        return 0;
}/* 16 진수 곱셈 */
```