Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 전문가 과정

<ARM Architecture>
2018.05.01 - 45 일차

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 - 안상재
sangjae2015@naver.com

<소스코드 및 명령어 분석>

1. asr : arithematic shift right
→ asr #2 : 2 칸 만큼 오른쪽으로 시프트

2. mrs : CPSR 레지스터 값을 특정 레지스터로 가져옴.
→ “mrs r0, cpsr” : cpsr 레지스터 값을 r0 으로 가져옴.

3. ldr
- 메모리로부터 레지스터로 데이터를 가져옴.

```
#include <stdio.h>

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

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") = 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]");	// r1 주소에서 r2 값의 바이트만큼 offset 위치의 값을 ro 로 가져옴.

	printf("r0 = %u\n", r0);	// r0 = 3

	return 0;
}
```

4. ldr 새로운 방법

```
#include <stdio.h>

char test[9] = "HelloARM";

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 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]!");	// 이동한 곳 까지 값을 갱신시킴. r2 만큼 이동시켜서 r1 을 fix 시킴.

	printf("test = %s, r1 = %s\n", test, r1);		// r1 = “ARM”

	return 0;
}
```

4. ldr 새로운 방법 2

```
#include <stdio.h>

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

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") = 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→r0 , r2→r1  따로따로 대입

	printf("r0 = %u, r1 = %u\n", r0,*r1);	// r0 = 1, *r1 = 2 (r1 에서 4 바이트 만큼 offset 됨)

	return 0;
}

/*
따로 놈
r1->r0 , r2->r1
*/
```

5. lsl (logical shift left)
→ "add r0, r1, r2, lsl #7" : r0 = r1 + (r2<<7)

6. mla
→ "mla r1, r2, r3, r4" : r1 = (r2*r3) + r4
(원래는 20~30clock 에 끝나지만 DSP 는 곱셈과 덧셈을 동시에 수행가능해서 1clock 에 끝남)

7. mul
→ “mul r1, r2, r3” : r1 = r2*r3

8. 항이 3 개 일 때, mov 연산
→ "mov r0, #0xff, 8" : 8 바이트를 버리지 않고 오른쪽으로 시프트 시킴.

9.

```
#include <stdio.h>

char test[9] = "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 에 test 배열의 5 번째 주소를 저장

	asm volatile("mov r0, #61");	// 61 은 아스키코드로 문자 ‘=’ 에 해당
	asm volatile("strb r0, [r1]");	// 문자 ‘=’ 을 r1 주소에 저장

	printf("test = %s\n", test);	// test = “Hello=RM”

	return 0;
}

/*
strb : 레지스터->메모리 로 1byte 저장
*/
```

10. stmia

```
#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)
{
	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");		// r1 = 3
	asm volatile("mov r2, r1, lsl #2");	// r2 = 12
	asm volatile("mov r4, #0x2");		// r4 = 2
	asm volatile("add r3, r1, r2, lsl r4");	// r3 = 51
	asm volatile("stmia r0, {r1,r2,r3}");	// {r1,r2,r3} 값을 r0 메모리 주소에 한꺼번에 store 함

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

	return 0;
}

/*
stm : store multiple
ia : increment after(먼저 증가시키고 값을 메모리에 집어넣음)
store : 레지스터 -> 메모리(스택)
*/
```

11. '!' 가 있는 strmia 사용

```
#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");		// r1 = 0x3
	asm volatile("mov r2, r1, lsl #2");	// r2 = 12
	asm volatile("mov r4, #0x2");		// r4 = 0x2
	asm volatile("add r3, r1, r2, lsl r4");	// r3 = 51
	asm volatile("stmia r0!, {r1,r2,r3}");	// !가 있을 경우 최종 주소가 r0 에 갱신
	asm volatile("str r4, [r0]");		// r4 가 r0 의 새롭게 갱신된 주소에 store 됨

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

	return 0;
}
```

12. umull

```
#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 r8, #0x44, 8");
	asm volatile("mov r9, #0x200");
	asm volatile("umull r4, r5, r8, r9");  /* r8 과 r9 를 곱해서 하위 32bit 는 r5 에, 상위 32bit 는 r4 에 저장함 */

	printf("r1r0 = 0x%x %08x\n", r5, r4); // %08x 는 0 8 개를 앞에 출력함
	return 0;
}
```

13. umlal

```
#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, #0xf");		// r0 = 0xf
	asm volatile("mov r1, #0x1");		// r1 = 0x1
	asm volatile("mov r2, #0x44, 8");	// r2 = 0x44000000
	asm volatile("mov r3, #0x200");		// r3 = 0x200
	asm volatile("umlal r0,r1,r2,r3");

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

```
        return 0;
}
/*
mulal : 곱하고 더해서 확장(상위가 r1  )

r1 = r2*r3(상위 32bit) + r1
r0 = r2*r3(하위 32bit) + r0
*/
```

14. stmia 와 ldmia 의 혼합형태

```
#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)
{
        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"           // r1 = 0x3
                     "mov r2,r1,lsl #2\n"       // r2 = 12
                     "mov r4, #0x2\n"           // r4 = 0x2
                     "add r3, r1, r2, lsl r4\n" // r3 = 51
                     "stmia r0!, {r1,r2,r3}\n"  // r0 의 위치는 test_arr 배열의 3 번째 index
                     "str r4, [r0]\n"
                     "mov r5, #128\n"           // r5 = 128
                     "mov r6, r5, lsr #3\n"     // r6 = 16
                     "stmia r0, {r4,r5,r6}\n"   // 현재 r0 위치부터 3 칸에 {r4,r5,r6} 를 store 함
                     "sub r0,r0, #12\n"         // r0 의 위치를 12 비트 뺌 (3 칸 이동)
                     "ldmia r0, {r4,r5,r6}");   // 현재 r0 위치부터 3 개의 값을 {r4,r5,r6} 에 load 함

        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);   // r4=3, r5=12, r6=51

        return 0;
}
```

15. ldreqb

```
#include <stdio.h>

char test[9] = "HelloARM";

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

	return 0;
}

/*
eq 는 cmp 의 결과에 의해 CPSR 레지스터의 Zero bit 가 1 이 되면 수행함
-> CPSR 레지스터의 zero bit 는 default 값이 1 이므로 바로 ldreqb 가 수행됨
b 는 1byte 를 적재하라는 의미
*/
```