TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/5/1
수업일수	45 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

- mov로테이션(1)
- mov로테이션(2)
- lsl(1)
- lsl(2)
- lsl(3)
- asr
- mrs
- mul
- mla
- umull
- umlal
- ldr(1)
- ldr(2)
- ldreqb
- strb
- ldr - !옵션
- ldr(3)
- stmia
- stmia - !옵션
- asm명령어 쓰는 법
- ldmia
- gdb(1)
- gdb(2)

mov 로테이션(1)

```
#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"); //mov 의 3 번째 인자를 통해 몇 비트의 숫자를 로테이션 시킬 것인지 설정.

	printf("r0 = 0x%x₩n",r0);

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.outr0 = 0xff000000

	0x00 00 00 ff
→ 0xff 00 00 00
```

mov 로테이션(2)

```
#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");
	asm volatile("mov r1, #0xf");
	asm volatile("add r2, r1, r0");

	printf("r2 = 0x%x₩n",r2);

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r2 = 0xff00000f

r0 = 0xff 00 00 00
r1 = 0xf
r2 = r0*r1 = 0xff 00 00 0f
```

lsl(1)

```
#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");
	asm volatile("mov r2, #3");
	asm volatile("add r0, r1, r2, lsl #7"); //r2 를 왼쪽으로 7 만큼 쉬프트하고 r1 과 더한 뒤 r0 에 저장

	printf("r0 = 0x%x₩n",r0); //r0 출력

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r0 = 0x187
==391
```

lsl (2) - 숫자 값 대신 레지스터를 lsl인자로 주면 레지스터 내의 값으로 수행.

```
#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");
        asm volatile("mov r2, #3");
        asm volatile("mov r3, #2");
        asm volatile("add r0, r1, r2, lsl r3"); //r3 에 저장된 값 만큼 r2 를 left 로 쉬프트 시킨 뒤 r1 과 더하고 r0 에 저장

        printf("r0 = 0x%x₩n",r0); //r0 출력

        return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r0 = 0x13

r1 = 7
r2 = 3
r3 = 2
3<<2 == 1100(2)
111+1100 == 10011(2) == 19(10) == 0x13
```

lsl (3)

```
#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"); // r1 을 2^2 만큼 왼쪽으로 쉬프트 시키고 r1 이랑 더한 뒤 r0 에 복사.

	printf("r0 = 0x%x₩n",r0); //r0 출력

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r0 = 0xa

2>>2 == 8
8 + 2 == 10 == a
```

asr - lsl와 반대로 오른쪽으로 쉬프트함

```
#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"); //r1 을  2^2 만큼 오른쪽으로 쉬프트하고 r0 과 더하기.

	printf("r0 = 0x%x₩n",r0);

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r0 = 0x8

32/4 = 8
```

mrs - Move from PSR (Program Status Register)to register, PSR명령어 중cpsr이나 spsr의 내용을 레지스터로 읽어오는 명령

```
#include<stdio.h>

void show_reg(unsigned int reg)
{
	int l;
	for(l=31;l>=0;)
		printf("%d", (reg>>l--)&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"); //cpsr 값이  r0 로 복사됨

	show_reg(r0); // r0 값 출력

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
01100000000000000000000000010000
```

mul

```
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"); //r2 와 r3 를 곱한 결과 값을 r1 에 저장

        printf("r1 = %d₩n",r1);

        return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r1 = 21
```

mla

: mla 명령어로 곱셈과 덧셈을 1클록만에 처리할 수 있음

```
#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"); //r2 와 r3 을 곱하고 r4 를 더함. mal 명령어를 1 클록만에 처리함

	printf("r1 = %d₩n",r1);

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r1 = 54

3*7+33=54
```

umull

```
#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"); // r2 와 r3 곱한 뒤 상위 32 비트는 r1 에, 하위 32 비트는 r0 에 넣음.

	printf("r1r0 = 0x%x%x₩n",r1,r0);

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r1r0 = 0x880

r2 = 0x44 00 00 00
r3 = 0x200
r2*r3 = 0x88    00 00 00 00 ->32 비트로 하면 짤리게 됨. 그래서 r1 은 상위비트 r0 는 하위 비트에 저장.

출력하면 r1 은 88 이 출력되지만 r0 는 0 이 출력되게 된다.

printf("r1r0 = 0x%x%08x₩n",r1,r0);로 바꾸면
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r1r0 = 0x8800000000
이런식으로 결과 값이 출력됨.
```

umlal

```
#include<stdio.h>

void show_reg(unsigned int reg)
{
	int l;
	for(l=31;l>=0;)
		printf("%d", (reg>>l--)&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");
	asm volatile("mov r1, #0x1");
	asm volatile("mov r2, #0x44, 8");
```

```
        asm volatile("mov r3, #0x200");
        asm volatile("umlal r0, r1, r2, r3");
        //r2 와 r3 를 곱한 뒤 상위비트는 r1 과 합한 뒤 r1 에 저장하고 하위비트는 r0 에 저장된 값과 합한 뒤 r0 에 저장.

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

        return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r1r0 = 0x89f
```

r2*r3 를 분할해서 r0 과 r1 에 넣은 뒤 상위비트와 r1 더하고 하위비트와 r0 더함.

Data transfer

ARM명령어 중 메모리에서 레지스터를 읽거나 메모리에 레지스터를 저장하는 LDR명령어와 STR명령어를 지원함.

LDR명령어는 메모리에서 워드를 레지스터로 읽어 들이는 명령어이다. 메모리의 주소를 레지스터로 읽어온다. (메모리에서 레지스터)

STR명령어는 레지스터에서 워드를 메모리에 저장하는 명령어이다.(레지스터에서 메모리)

ldr(1)

```
#include<stdio.h>

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

void show_reg(unsigned int reg)
{
	int l;
	for(l=31;l>=0;)
		printf("%d", (reg>>l--)&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]"); //8 바이트를 r1 의 위치에서 왼쪽으로 재배치해서 r0 에 저장

        printf("r0 = %u₩n",r0); //r0 값 출력

        return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r0 = 3
```

r1 이 배열의 시작주소를 저장. r2 가 0x8 을 저장하고 r1 에서 8byte 를 이동하면 int 형 배열이므로 2 칸이동해서 3 이 출력.

ldr(2)

```
#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;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r0 = 2
r1 에서 4 바이트 이동해서 다음 인자인 2 가 출력됨
```

ldreqb

```
#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]");//r1 에서 5 바이트 왼쪽으로 이동한 뒤 r0 에 저장.
```

```
        printf("r0 = %c₩n",r0);

        return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r0 = A
```

제로플래그가 셋팅되어 있어서 if 문 없어도 동작.
r1 인 test 배열 시작주소에서 5 바이트 이동해서 다음 요소인 A 출력

strb

```
#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");
        asm volatile("strb r0, [r1]");

        printf("test = %s₩n",test);

        return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
test = Hello=RM
```

str → ldr 의 반대.

ldr - !옵션

```
#include<stdio.h>

char test[] = "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]!"); //5 바이트만큼 왼쪽으로 이동한 뒤 r1 을갱신함.

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

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
test = HelloARM, r1 = ARM
```

ldr(3)

```
#include<stdio.h>

unsigned int arr[] = {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 이 r0 에 들어가고 r2 가 r1 에 들어간다.

	printf("r0 = %u, r1 = %u₩n",r0,*r1);

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
r0 = 1, r1 = 2
```

r1 이 r0 에 들어가고 따로 r2 가 r1 에 들어감.
r1 에 배열의 시작주소 들어가서 1 을 출력하고, r2 는 4 이므로 r1 의 위치에서 4 바이트만큼 이동한 값이 갱신되어 2 가 출력.

stmia - **store연산은 레지스터 값을 메모리에 넣는 연산이다. increment after이므로 레지스터의 값을 메모리(스택)에 넣을 때 주소를 증가시키고 값을 집어넣음.**

```
#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}"); // r0 가 가리키는거에 순서대로 r1,r2,r3 를 집어넣음.

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

	return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ 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

stmia 연산으로 r1, r2, r3 값들을 배열에 순서대로 넣어서 4 번째와 5 번째 배열은 초기값 0 을 그대로 가지고 있음.
```

stmia - !옵션

```
#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}"); // r0 가 가리키는 곳에 차례대로 r1,r2,r3 을 넣음. 그리고 이동한 위치로 갱신시킨다.
        asm volatile("str r4, [r0]");



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

        return 0;
}
```

```
xeno@xeno-NH:~/proj/0501$ 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

! 이동한 위치로 갱신시킴 그래서 2 가 나옴.
!없으면 3 이아니라 2 로 시작됐을것
```

asm명령어 쓰는 법

```
#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;
}
```

```
xeno@xeno-NH:~/proj/0501$ 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
```

ldmia

```
#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" //r1, r2, r3 레지스터 값을 r0 에 보내고 위치 값을 갱신
                        "str r4, [r0]₩n" //r0 에 r4 레지스터의 값을 전송
                        "mov r5, #128₩n"
                        "stmia r0, {r4, r5, r6}₩n" // r4, r5, r6 레지스터 값을 r0 에 전송
                        "sub r0,r0,#12₩n" //r0 의 위치 값에서 4 바이트*3 인 주소 값을 빼서 r0 에 넣음. r0 위치를 원상복구 시킨다.
                        "ldmia r0, {r4, r5, r6}"); //r0 가 가리키는 메모리 값을 r4, r5, r6 에 차례대로 넣음.

        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;
}
```

```
xeno@xeno-NH:~/proj/0501$ 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] = 16
test_arr[6] = 0
r4 = 3, r5 = 12, r6 = 51
```

ldmia 명령어 수행전에는 r4=2, r5=128, r6=0 이었는데 위의 결과 값을 보면 3, 12, 51 이 나오게 됨.

gdb(1)

```
#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;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
res=4
```

gdb(2)

```
#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;
}
```

```
xeno@xeno-NH:~/proj/0501$ qemu-arm-static -L /usr/arm-linux-gnueabi a.out
res=4
```

기계어 분석은 낼 하도록 하겠습니다…ㅠㅠ