TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-04-30 (44 회차)

강사: lnnova Lee(이상훈)
gcccompil3r@gmail.com
학생: 정유경
ucong@naver.com

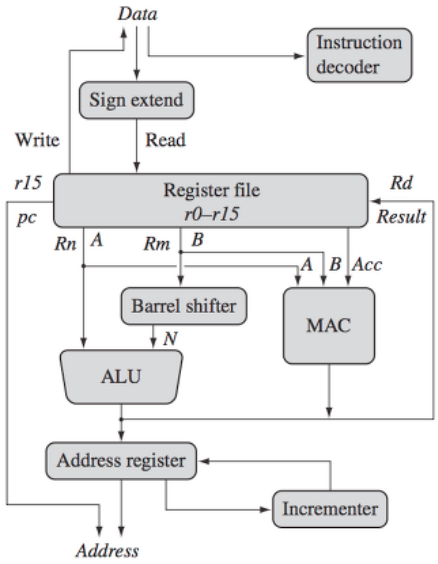
■ ARM 프로세서가 많이 쓰이는 이유
1. 저전력 소모
2. RISC - 간단한 명령어로 회로가 단순하고 그로 인해 속도 빠름
3. 소형의 다이 사이즈

■ CPU 발전 변천사
CISC → RISC → VLIW → Multi-Core → Heterogeneous

■ DSP 와 FPGA 의 이점과 어셈블리어의 관계성
- 샘플링타임, 샘플 갯수, 아날로그식을 디지털식으로 바꾸는 방법
how? sinx 를 테일러 급수로 처리함
e^ix 는 오일러 공식으로 처리함 = cosx + isinx (복소표기)
주파수 무한대하면 T 는 0 , 근데 무한대가 불가능하므로 T 는 0 이 안된다 즉, 연속이 안된다
아날로그적으로 표현이 안된다.

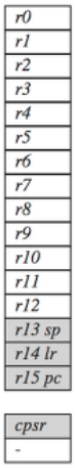
- T 를 샘플링타임이라고 한다.
샘플링 값을 일정시간마다 구해서 연결하면 근사한 값 얻을 수 있지만, 샘플링타임이 2 초로 길다면 의미없음
원래 데이터와 차이 많이난다. 샘플링 타임을 줄이면 일부 손실이 있더라도 유의미한 값이 됨 그러나 엄청 비싸다. (초고성능 ADC 600 800 만원 레이더용으로 사용함)
아날로그적으로 e^ixt 표현 불가
k[n]…. 샘플링 타임 줄이면 n 이 엄청나게 늘어남
이때 곱셈을 dsp 없이 한다고 하면 무척 곤란하다

■ ARM 아키텍쳐 내부구조



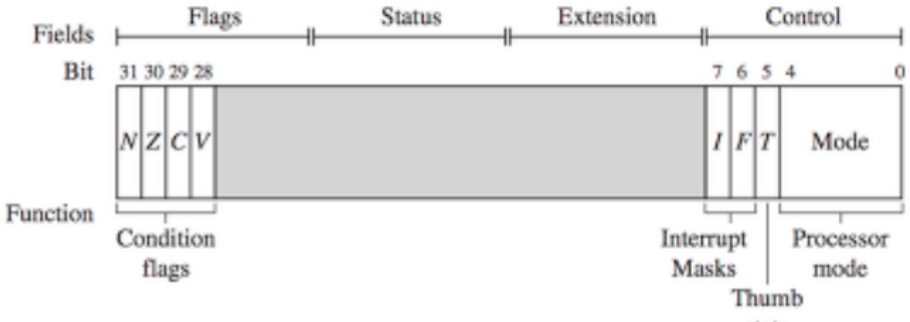
■ MAC 은 곱셈기이다. 옵션으로 ARM 아키텍쳐에 없을 수도있다.
들어있으면 DSP 가 된다. 없으면 그냥 ARM 일뿐
곱셈을 빠르게 하자 mac
보편화 시킨게 dsp
1 클록에 다 처리함, 4 개를 동시에 처리함 심드(SIMD) 구조이기 때문
맥이 필요한 이유?
연산 클럭수를 줄이기 위해!
맥이 있으면 곱셈연산을 1 클록에 처리할 수 있다
곱셈과 덧셈을 동시에 수행해도 1 클록에 끝냄
이걸 병렬로 수행할 수도 있음
DSP 가 있고 없고에서 엄청난 성능의 차이가 발생함

■ ARM 범용 레지스터 (r0 ~ r15, cpsr)



레지스터는 x86 과 상당한 차이점이 있다.
일단 범용 레지스터의 수가 더 많으며, x86 에는 존재하지 않는 Link Register 가 존재한다.
R0 ~ R12 : 범용 레지스터, 인자값 및 임시 계산 저장소 등
R13(SP) : Stack Pointer, x86 의 ESP 와 비슷한 역할 수행
R14(LR) : Link Register, 함수 호출 전 LR 에 리턴 주소를 저장하고 점프함(함수 호출 시 스택에 복귀주소 저장하지 않고 r14 에 저장 - 리턴주소 스택 활용 X)
PC : x86 에서의 EIP 레지스터와 동일한 역할 수행. 다음에 실행할 코드의 주소 저장

■ 현재 프로그램 상태 레지스터 (CPSR)



CPSR 은 eflags 레지스터의 역할을 함

- 조건 코드 비트
N = 계산 결과가 음수
Z = 계산 결과가 '0'
C = 계산 후 carry 발생
V = 계산 후 overflow 발생
- 인터럽트 disable 비트
7,6 번 interrupt mask : 인터럽트 어떻게 사용할지 결정
eflags 레지스터의 9 번비트와 같음
- mode 비트: 프로세서의 7 개 모드 중 하나를 표시
프로세서 모드 = CPU 동작모드
0~4 의 5 개 비트를 가지고 지정함

■ ARM 프로세서 동작모드 7 가지
하위 5bit(0-4)는 현재 동작 모드를 나타낸다.

mode	M[4:0]
usr	10000
fiq	10001
irq	10010
svc	10011
abt	10111
und	11011
sys	11111

■ ARM 어셈블리 프로그래밍 환경 구축

sudo apt-get update
sudo apt-get install qemu-user-static qemu-system
sudo apt-get install gcc-arm-linux-gnueabi
컴파일할 C 소스파일을 작성한다
arm-linux-gnueabi-gcc -g 소스파일
sudo apt-get install gdb-multiarch
터미널을 2 개 띄운다.

A 터미널에서 아래 명령어를 수행한다.	B 터미널에서 아래 명령어를 수행한다.
qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out : -g 1234 빼면 값을 확인할 수 있다 qemu-arm-static -g 1234 -L /usr/arm-linux-gnueabi ./a.out 커서 깜빡인다 이 상태로	gdb-multiarch file a.out target remote localhost:1234

*. 디버깅 진행하기
- info registers 입력하면 레지스터들 목록 나온다
- b main (b *주소)
- c

■ 리눅스 상에서 ARM 어셈블리 프로그래밍
(add, sub, rsb, and, orr, eor, bic, cmp, teq, tst, mov, mvn 명령어 실습)

```
/*add.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;

        r1 =77;
        r2 =37;

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

```
/*sub.c*/
#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;
```

```
        printf("r0=%d\n", r0);
        return 0;
}
```

```
        r3 = 34;
        if(r1 > r2)
                asm volatile("subgt r3, r3,#1"); //
sub + gt(greater than): 이면 r3 에 대입한다(33)
        printf("r3=%d\n",r3);
        return 0;
}
```

```
/*rsb.c*/
#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")=0;
        register unsigned int r5 asm("r5");

        r1 = 77;
        r2 = 37;
        r5 = 3;

        if(r2 <= r1)
                asm volatile("rsble r4, r5, #5");// 5-
r5 = 5-3 =r4 =2

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

```
/*and.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");
        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; // 32+2
        r2 = 37; // 32+5 32 만 겹치는구나

        asm volatile("and r0,r1,r2"); // r0 = r1 & r2
        show_reg(r0);

        return 0;
}
```

```
/*orr.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)
```

```
/*eor.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;

	r0 = 7;
	r1 = 7;
	r5 = 3;

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

	if(r0 == r1)
	{
		r0 = 10;
		r3 = 5;
		asm volatile("eors r1,r3,r0"); // r1 =
r3^r0
	}
	show_reg(r1); // r1 출력
	return 0;
}

//r1  0xf (1111)
// cpsr 0x20000010        536870928 로 바뀜 왜???
```

```
/*bic.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");
	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"); //r2 =
r3 &! #7
	}

	show_reg(r2);
```

```
/*cmp.c*/
// cmp 할때마다 cpsr 값 갱신된다

#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("cmp r0,r1"); //이미 6....상태유
지
/*r0-r1 을 conddition field 에 업데이트
r0 - r1 < 0 : CF(캐리 플래그)가 설정(1) - 캐리 1, 제
로 0  ????
r0 - r1 > 0 : 캐리 0, 제로 0
r0 - r1 = 0 : ZF(제로 플래그)가 설정(1) - 캐리 0, 제로
1*/
	asm volatile("mov r2, #5"); /*r2 에 5 를 대입
*/
	asm volatile("cmp r0, r2"); // 8 = 1000 r2 바
뀌었으니까 제로플래그 0 로 바뀜
```

```
        return 0;
}
/*42 &~ (2^3 -1)
42 &~ (7)
42 를 2^3 의 배수로 정렬하면 32 + 8*/
```

```
        return 0;
}
```

```
/*cmn.c*/

*. 비교연산 주의할 내용
음수비교 cmn //CMN: operand1 + operand2, but result not written
```

```
/*teq.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("cmp r0,r1");
        asm volatile("mov r2, #3");
        asm volatile("tsteq r2, #5");
/*여기 eq 는 위의 cmp 가 동작시킴 tst 에 의해 3 이랑 5 랑 and 하면 1 이됨
(cpsr 값이 6->2 된다) 제로플래그는 0 이 아니니까 꺼진것 */
        show_reg(r2);

// tst: and

        return 0;
}
```

```
/*tst.c*/
/*cpsr 0x60000010 1610612752 맨앞비트가 6 이었는데 2 로 바뀐다
16 비트를 2 진수로 바꾸면 4 비트이고 6 은 0110 2 는 0010 이니까
제로플래그가 1->0 이된것*\/
```

```
/*mvn.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("cmp r0,r1");
        asm volatile("mvneq r1, #0");
/* 제로플래그가 1 이어야지만 실행된다. eq 지우면
언제나 실행이고
eq 있으니까 위의 조건 만족하는 지 봄 0xFFFF 랑 r1
이랑 xor 이건 mvneq 의 특징임*/
        printf("r1=0x%x\n", r1);
        return 0;
}
```

[ARM 32 비트 주소표현]
mov 0x40000(20bit) << 12 0x4000 0000
되어서 32 비트를 표현할 수 있다.