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

강사 - Innov (이상훈)
gcccompil3r@gmail.com
학생 - 이유성
dbtjd1102@naver.com

BOOTLOADER_CODE_CORETEX-M4(32bit)

부팅을 하면 초기화
-전원을 누르면 reset_handler가 작동해서 proc reset(초기화/재설정)을 함

BOOT CODE

```
; Reset handler
Reset_Handler   PROC
                EXPORT  Reset_Handler             [WEAK]
        IMPORT  SystemInit
        IMPORT  __main

                ;FPU settings
                LDR     R0, =0xE000ED88         ; Enable CP10,CP11
                LDR     R1,[R0]
                ORR     R1,R1,#(0xF << 20)
                STR     R1,[R0]

                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =__main
                BX      R0
                ENDP

; Dummy Exception Handlers (infinite loops which can be modified)

NMI_Handler     PROC
                EXPORT  NMI_Handler               [WEAK]
                B       .
                ENDP
HardFault_Handler₩
                PROC
                EXPORT  HardFault_Handler         [WEAK]
                B       .
```

● LDR R0=0xE000ED88 r0 에 0xE000ED88를 넣는다
● LDR R1,[R0] r0의 데이터 값을 r1에 넣는다
● ORR R1,R1,#(0xF <<20) R1 | 0x00f0 0000 (0xF는 32비트에서==0x0000000F)
이 말은 R1 20~23비트를 1로 set하겠다. 다른 값은 모름

● R1에 있는 값으로 R0에 있는 데이터를 변경 시킨다. (cp 11, cp 10을 1로 set)

여기서 0xE000ED88을 찾아보면,
Floating Point Unit (FPU) (*FPU :부동소수점)

Address	Name	Type	Reset	Description
0xE000ED88	CPACR	RW	0x00000000	*Coprocessor Access Control Register*
0xE000EF34	FPCCR	RW	0xC0000000	*Floating-point Context Control Register* on page 4-49
0xE000EF38	FPCAR	RW	-	*Floating-point Context Address Register* on page 4-50
-	FPSCR	RW	-	*Floating-point Status Control Register* on page 4-50
0xE000EF3C	FPDSCR	RW	0x00000000	*Floating-point Default Status Control Register* on page 4-52

이 표는 Cortex-M4F FPU안에 있는 floating-point system registers를 보여준다.
여기서 CPACR을 보면,

CPACR (FPU쓸건지 안 쓸건지 정한다.) Coprocess의 정보 (0~ 31)

4.6.1 Coprocessor Access Control Register

The CPACR register specifies the access privileges for coprocessors. See the register summary in *Cortex-M4F floating-point system registers* for its attributes. The bit assignments are:

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
Reserved		CP11	CP10	Reserved			

Table 4-50 CPACR register bit assignments

Bits	Name	Function
[31:24]	-	Reserved. Read as Zero, Write Ignore.
[2*n*+1:2*n*] for *n* values10 and 11	CP*n*	Access privileges for coprocessor *n*. The possible values of each field are: `0b00` = Access denied. Any attempted access generates a NOCP UsageFault. `0b01` = Privileged access only. An unprivileged access generates a NOCP fault. `0b10` = Reserved. The result of any access is `Unpredictable`. `0b11` = Full access.
[19:0]	-	Reserved. Read as Zero, Write Ignore.

31 ~ 24 기능 x
23 ~ 20 0b00 아무도 못 쓴다. 0b01시스템만 쓰고 유저는 못 쓴다.
0b10 아무기능 x 0b11 모두 접근 가능.
19 ~ 0 기능 x

다시 소스코드로 돌아가자

●R0에 SystemInit의 값이 저장된다. 함수 포인터 (함수의 이름은 주소)

●BLX R0 R0에 있는 주소로 점프하겠다. (systeminit함수로 가겠다.)

systeminit함수를 보면수를 보면

```
/**
  * @brief  Micro Controller System을 설정한다.
  *         Embedded Flash Interface, PLL을 초기화하고 SystemFrequency 변수를 갱신한다.
  * @param  None
  * @retval None
  */
void SystemInit(void)
{
  /* RCC Clock 구성을 Default Reset State로 reset(재설정)한다. */
  /* Set HSION bit */
  RCC->CR |= (uint32_t)0x00000001;

  /* Reset CFGR register */
  RCC->CFGR = 0x00000000;

  /* Reset HSEON, CSSON and PLLON bits */
  RCC->CR &= (uint32_t)0xFEF6FFFF;

  /* Reset PLLCFGR register */
  RCC->PLLCFGR = 0x24003010;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;

  /* 모든 Interrupt를 비활성화한다. */
  RCC->CIR = 0x00000000;

#ifdef DATA_IN_ExtSRAM
  SystemInit_ExtMemCtl();
#endif /* DATA_IN_ExtSRAM */

  /* System Clock Source, PLL 곱셈기, 나눗셈기, AHB/APBx Prescalers와 Flash 설정을 구성한다. */
  SetSysClock();

  /* Offset Address를 더한 Vector Table 위치를 구성한다. */
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* 내부 SRAM에 Vector Table 재배치 */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* 내부 FLASH(NAND)에 Vector Table 재배치 */
#endif
}
```

(데이터 시트 검색)

● RCC->CR |=0x00000001 0번 비트 set

6.3.1 RCC clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		PLLSAI RDY	PLLSAI ON	PLLI2S RDY	PLLI2S ON	PLLRD Y	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
		r	rw	r	rw	r	rw					rw	rw	r	rw

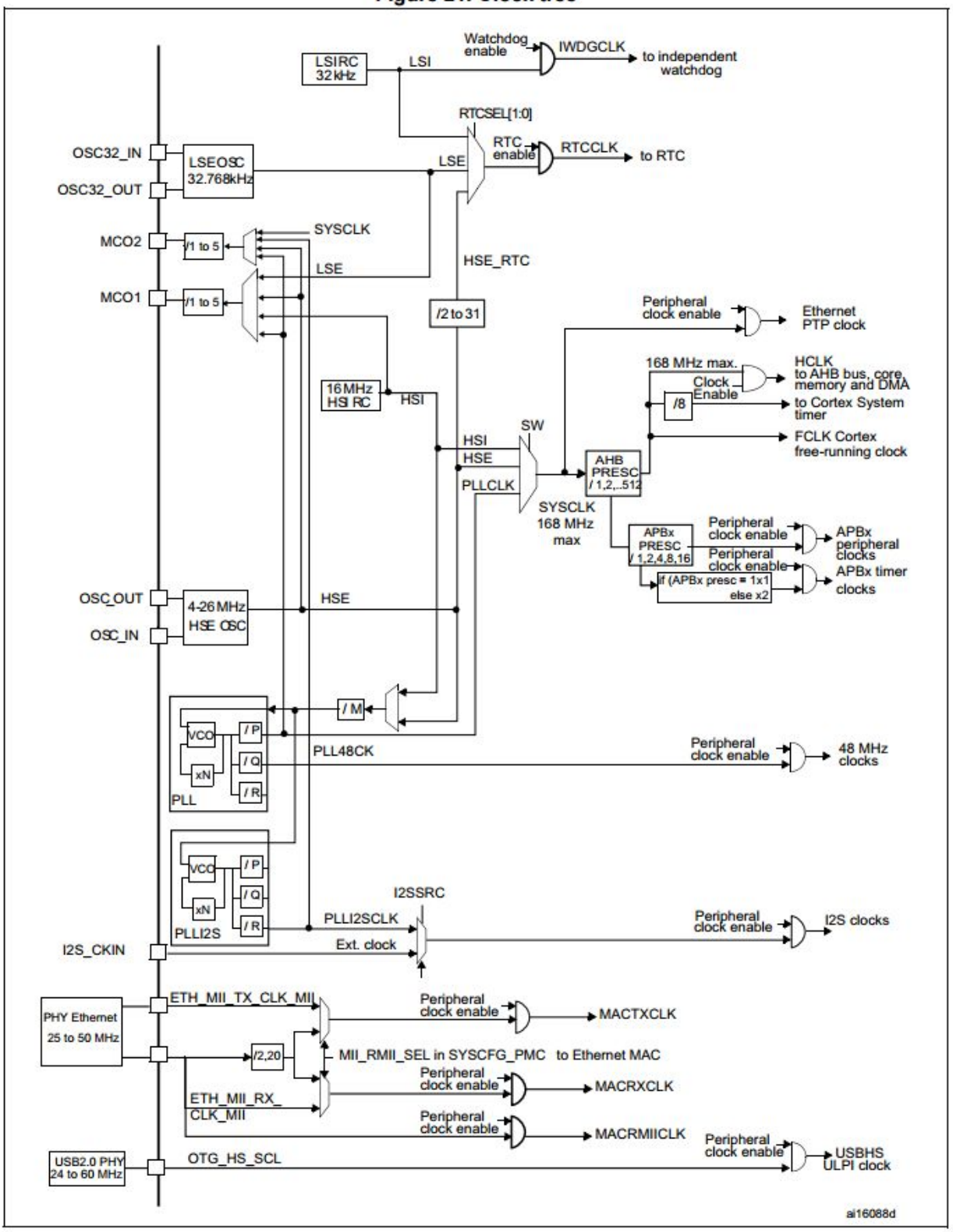
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Bit 0 HSION ：Internal high-speed clock enable

0: HSI oscillator OFF
1: HSI oscillator ON

oscillator란?(clock tree를 켜보자)->16 MHz HSI RC를 ON하겠다
HSI 내부 오실레이터

Figure 21. Clock tree



● RCC->CFGR　CFGR레지스터를 0x00000000으로 초기화 해준다

● RCC->CR &= 0xFEF6 FFFF
0x 1111 1110 1111 0110 FFFF　16 19 24번 째 비트를 클리어 하겠다.

(앤드연산자는 clear 해줄 때 사용 // OR연산자는 set 해줄 때.)

7.3.1 RCC clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

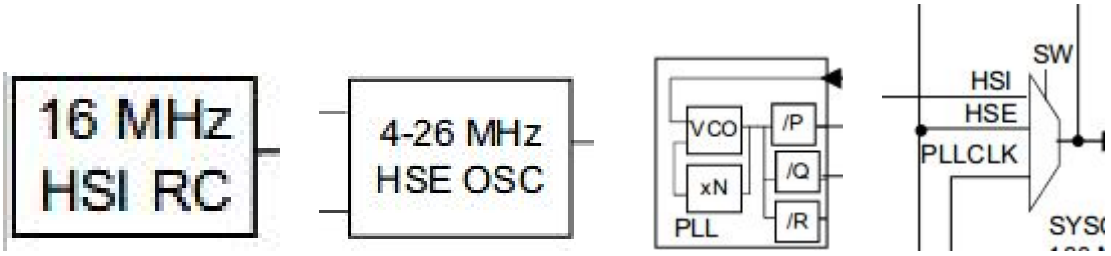
Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLI2SRDY	PLLI2SON	PLLRDY	PLLON	Reserved				CSSON	HSEBYP	HSERDY	HSEON
				r	rw	r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSIRDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

PLLON(24,Phase Locked Loop) ,CSSON(19,보안) , HSEON(16,외부 클록 오실레이터) 다 끔

*HSE : 32.7~~Mhz
*PLL : 동적고주파 깔끔히 쓰려고
SW(스위치)에 따라 HSI를 쓸지 HSE를 쓸지 PLL를 쓸지 고를 수있다.



● RCC->PLLCFGR = 0x24003010 (PLLCFGR 레지스터를 이 값으로 reset)

● RCC->CR &= 0xFFFB FFFF　1111 1111 1111 1011 FFFF
31비트　0비트

18번째 비트 꺼짐

Bit 18 HSEBYP: HSE clock bypass

0: HSE oscillator not bypassed
1: HSE oscillator bypassed with an external clock

The HSEBYP bit can be written only if the HSE oscillator is disabled
HSE 오실레이터가 꺼질 때만 쓸 수 있다.
● RCC->CIR = 0x0000 0000

*Clock Interrupt Register 인터럽트를 다 끄겠다 부팅할 때 인터럽트 받지 않기 위해

● DATA IN ExtSRAM ->SRAM을 장착하면 안에 있는 함수를 실행해라.

(외부에서 메모리를 하나 더 끼면 초기화 해줘야 하기 때문에)

SetSysClock함수

```
/**
  * @brief  System Clock Source, PLL 곱셈기 & 나눗셈기, AHB/APBx Prescalers와 Flash 설정을 구성한다.
  * @Note  이 함수는 RCC Clock 구성을 Default Reset State로 Reset하기 위해 단 한 번만 호출된다.
  * @param  None
  * @retval None
  */
static void SetSysClock(void)
{
/******************************************************************************/
/*            PLL (clocked by HSE)을 System Clock Source로 사용한다.          */
/******************************************************************************/
  __IO uint32_t StartUpCounter = 0, HSEStatus = 0;

  /* HSE를 활성화 */
  RCC->CR |= ((uint32_t)RCC_CR_HSEON);

  /* Time Out되서 종료되거나 HSE가 종료될때까지 대기한다. */
  do
  {
    HSEStatus = RCC->CR & RCC_CR_HSERDY;
    StartUpCounter++;
  } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

  if ((RCC->CR & RCC_CR_HSERDY) != RESET)
  {
    HSEStatus = (uint32_t)0x01;
  }
  else
  {
    HSEStatus = (uint32_t)0x00;
  }
```

HSE_STARTUP_TIMEOUT의 값은 0x05000

RESET = 0

● RCC->CR |= RCC_CR_HSEON (HSEON ON..아마도 RCC_CR_HSEON == 0x FFF xxx1 FFFF) (펌웨어는 디파인값으로 주지 않고 이름으로 짐작할 수 있게 하는게 많다.by SB)

● HSEStatus = RCC->CR &RCC_CR_HSERDY (Set by hardware to indicate that the HSE oscillator is stable

*HSE가 ON되어있다고 해서 Ready상태가 되는 것이 아니라 HSE가 안정화 되면
하드웨어에서 값을 셋팅해줌 (셋팅되면 while문 안 들어감)
 Ready 안 되었다면 안정될 때까지 While문

..//Q. RCC_CR_HSERDY , 바로 위에 부분 다시..///

● if((RCC->CR & RCC_CR_HSERDY) !=reset) if문 만족은 레디가 되어있다.

```
if (HSEStatus == (uint32_t)0x01)
{
  /* High Performance Mode를 활성화하고, System Frequency를 168 MHz로 올린다. */
  RCC->APB1ENR |= RCC_APB1ENR_PWREN;
  PWR->CR |= PWR_CR_PMODE;          RCC->APB1ENR |= RCC_APB1ENR_PWREN;

  /* HCLK = SYSCLK / 1*/
  RCC->CFGR |= RCC_CFGR_HPRE_DIV1;

  /* PCLK2 = HCLK / 2*/
  RCC->CFGR |= RCC_CFGR_PPRE2_DIV2;

  /* PCLK1 = HCLK / 4*/
  RCC->CFGR |= RCC_CFGR_PPRE1_DIV4;

  /* main PLL을 구성한다. */
  RCC->PLLCFGR = PLL_M | (PLL_N << 6) | (((PLL_P >> 1) -1) << 16) |
                 (RCC_PLLCFGR_PLLSRC_HSE) | (PLL_Q << 24);

  /* main PLL을 활성화 */
  RCC->CR |= RCC_CR_PLLON;

  /* main PLL이 준비될때까지 대기한다. */
  while((RCC->CR & RCC_CR_PLLRDY) == 0)
  {
  }

  /* Flash Prefetch, Instruction Cache, Data Cache를 구성하고 대기 상태 */
  FLASH->ACR = FLASH_ACR_ICEN |FLASH_ACR_DCEN |FLASH_ACR_LATENCY_5WS;

  /* System Clock Source로 main PLL을 선택한다. */
  RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
  RCC->CFGR |= RCC_CFGR_SW_PLL;

  /* System Clock Source로 main PLL이 사용될때까지 대기한다. */
  while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS ) != RCC_CFGR_SWS_PLL);
  {
  }
}
else
{ /* HSE가 Start-Up에 실패하면 Application은 잘못된 Clock을 구성할 것이다.
     사용자(학생분들)가 이러한 오류를 다루기 위한 Code를 이곳에 추가하면 된다. */
}
}
```

●RCC->APB1ENR |=RCC_APB1ENR_PWREN Peripheral 1번 bus가 활성화 됨.

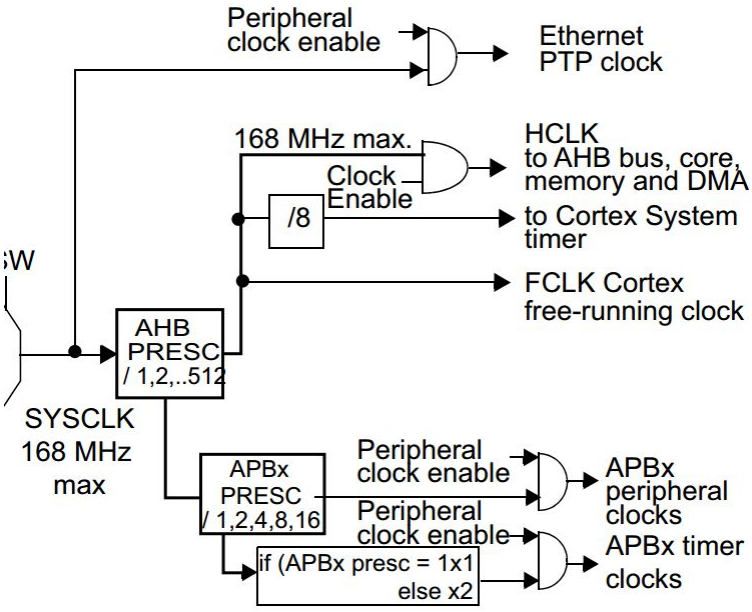
Bit 28 PWREN: Power interface clock enable
Set and cleared by software.

0: Power interface clock disabled
1: Power interface clock enable

PWR->CR |=PWR_CR_PMODE; PWR_CR은 처음 전원 켜질 때 플로팅을 방지하기 위한 녀석.

SW(스위치)있는 쪽 설정

AHB , APB로 분주기 하는이유 각각 필요한 주파수가 다르므로.



7.3.3 RCC clock configuration register (RCC_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]			MCO1 PRE[2:0]			I2SSCR	MCO1		RTCPRE[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPRE2[2:0]			PPRE1[2:0]			Reserved		HPRE[3:0]				SWS1	SWS0	SW1	SW0
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	r	r	rw	rw

(RCC_CFGR reset 0x0000 0000)

● RCC->CFGR |= RCC_CFGR_HPRE_DIV1

Bits 7:4HPRE:AHB prescaler 분주기
0xxx: system clock not divided

RCC_CFGR_HPRE_DIV1 (=>0xxx: system clock not divided 0xxx로 셋팅된 것을 확인 가능)
7~4비트 0xxx로 됨
0x 0000 0000 | 0000 000x

● RCC_CFGR_PPRE2_DIV2 (100: AHB clock divided by 2, 100으로 셋팅된 것을 확인 가능.)
(분주비를 2로 나누어줌)
15 ~13비트 100

0x 0000 0000 | 0x 0000 8000

● RCC_CFGR_PPRE1_DIV4(101: AHB clock divided by 4, 101으로 셋팅된 것을 확인가능.)
12 ~ 10비트 101

0x 0000 0000 | 0x 0000 1400

*핀이랑 연결된 모든 것이 Peripheral

2.Arm ASM Systemcall

stc.c

```
#include<stdio.h>
#include<unistd.h>
#include<errno.h>
#include<stdlib.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;
        register int r7 asm("r7") =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}\n"
                "swp r6,r3,[r0]");

        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);

        r7 = 2;  //시스템 콜

        asm volatile("swi #0" : "=r"(r0) : "r"(r7) : "memory"); //software interrupt

        if(r0>0)
                printf("r0 = %p,Parent\n",r0);
        else if(r0 ==0)
                printf("r0 =%p,Child\n",r0);
        else{
                perror("fork() ");
                exit(-1);
        }
        return 0;
}
```

명령어 : 출력 : 입력 : 특수지시어
 “=r”(0) 레지스터(r0)에 넣어라 , “r” r7 값을 전달 (r7 =2 ; 에서 2는 fork?)
“memory” memory barrier. (이 명령어 끝날 때까지 instruction scheduling을 하지마라
메모리에 다른 값이 접근하면 레지스터 값이 바뀔 수도 있다.)

r0,r1,r2,r3: 함수의 인자로 사용
r0 : 함수의 리턴값
r7 :시스템 콜
r13,r14,r15 : sp lr pc
r11 :스택의 베이스 포인터
명령어 : 출력 : 입력 : 특수지시어