

## 설계와 구현을 통한 임베디드 OS의 이해와 응용 ④

## MMU와 캐시 설정



지난 호에서는 부트로더의 나머지 부분과 간단하게 cuteOS의 시작 부분을 작성해 보았다. 이번 기사에서는 MMU와 캐시 설정 부분을 작성해 보기로 하자.

글: 서민우/과학기술 정보연구소([www.stii.co.kr](http://www.stii.co.kr))  
minucy@hanmail.net

소프트웨어가 동작하는 환경에서 MMU(Memory Management Unit)와 캐시(cache)가 하는 기능은 중요하다. 캐시는 시스템의 성능을 높이기 위해 사용하는 하드웨어 장치이며, 가상주소를 관리하는 MMU는 여러 프로세스 간에 주소영역을 분리해 메모리 상호 간섭을 막는 역할을 한다. 여기서는 MMU의 특성을 명확하게 이해하고 가상주소가 갖는 의미를 생각해 보기로 하자.

먼저 head.S 파일의 내용을 살펴보기로 하자. head.S 파일의 내용은 다음과 같다.

```
head.S
.equ IRQ32MOD, 0x12
.equ SVC32MOD, 0x13
.equ SYS32MOD, 0x1f
.equ NOINT, 0xc0

.globl _start
```

```
_start:
    bl mmusetup

stack_setup:
    mov r0,#0x100000
    sub sp,r0,#4

    msr cpsr_c,#NOINT|IRQ32MOD
    sub sp,r0,#0x1000
    msr cpsr_c,#NOINT|SYS32MOD
    ldr sp,=0x408000
    msr cpsr_c,#NOINT|SVC32MOD

    ldr pc,=kmain
```

head.S 파일에서는 'mmusetup' 이라는 함수를 호출하며, mmusetup 함수에서는 앞으로 cuteOS가 가상주소 공간상

에서 동작하도록 translation 테이블 작성 및 MMU를 설정한다. 따라서 mmusetup 함수를 호출하기 전에는 물리주소 공간상에서 동작하지만, mmusetup 함수에서 리턴한 이후에는 가상주소 공간상에서 동작한다.

뒤에서 구체적으로 살펴보겠지만, \_start의 주소로 mmusetup 함수를 호출하기 전에는 물리주소 0x3010\_0000 번지를 사용한다. 하지만, mmusetup 함수에서 리턴한 이후에는 \_start의 주소로 가상주소 0x0000\_0000 번지를 사용하게 된다. 가상주소로 0x0000\_0000 번지를 사용하는 이유는 다음 기사에서 구체적으로 다룰 벡터테이블의 위치가 \_start 번지에 놓이기 때문이다. mmusetup 함수는 mmusetup.S 파일에 정의되어 있다.

mmusetup 함수에서 리턴한 후에는 가상 주소 공간상에서 SVC 모드, IRQ 모드, SYSTEM 모드로 사용할 스택 포인터 값을 설정한다. 먼저 SVC 모드에서 사용할 스택 포인터 값을 (0x10\_0000-4)로 설정한다. 다음은 IRQ 모드로 전환한 후, 이 모드에서 사용할 스택 포인터 값을 ((0x10\_0000-4)-0x1000)으로 설정한다. 참고로 SVC 모드에서 사용할 스택의 주소 범위는 (0x0f\_f000~0x10\_0000)으로, IRQ 모드에서 사용할 스택의 주소 범위는 (0x0f\_e000~0x0f\_f000)으로 설정한다.

이후 SYSTEM 모드로 전환해서 이 모드에서 사용할 스택 포인터의 값을 0x40\_8000으로 설정하고, 다시 SVC 모드로 돌아온다. 주의할 점은 모드 변경시 아직은 exception을 처리할 준비가 되어 있지 않기 때문에 cpsr 레지스터의 I bit와 F bit를 1로 설정함으로써 외부에서 들어오는 인터럽트를 막아야 한다. 이렇게 각 모드에서 사용할 스택 포인터 값을 적당히 설정했다면 kmain 루틴으로 쏜다.

여기까지가 head.S 파일의 내용이다. 다음은 mmusetup.S 파일의 내용을 살펴보기로 하자. 먼저 mmusetup.S 파일의 내용은 다음과 같다.

```
mmusetup.S
#define L1PT 0x30000000
#define L2PT1 0x30004000
#define L2PT2 0x30004400
```

```
#define L2PT3 0x30004800
#define L2PT4 0x30004c00

#define KERNELBASE (0x30100000|0x55<<4|0x2<<2|0x2)
#define KERNELL2PT (L2PT1|0x03<<5|1<<4|0x1)
#define SFRSECBASE (0x48000000|0x1<<10|0x03<<5|1<<4|0x0<<2|0x2)
#define PTSECBASE (L1PT|0x1<<10|0x03<<5|1<<4|0x2<<2|0x2)
#define DOMAINVAL 0x00000040

.globl mmusetup
mmusetup:
mmu_init_pt:
    ldr r0,=L1PT
    add r1,r0,#0x5000
    mov r2,#0x0
1:
    str r2,[r0],#4
    cmp r0,r1
    blt 1b

mmu_map_kernel:
    ldr r0,=L2PT1
    add r1,r0,#0x400
    ldr r2,=KERNELBASE
1:
    str r2,[r0],#4
    cmp r0,r1
    addlt r2,r2,#0x1000
    blt 1b

    ldr r0,=L1PT
    ldr r2,=KERNELL2PT
    str r2,[r0]
    add r0,r0,#0xc00
```

```

add r0,r0,#4
str r2,[r0]

mmu_map_sfr:
    ldr r1,=L2PT1
    sub r0,r1,#0x600
    ldr r2,=SFRSECBASE
1:
    str r2,[r0],#4
    cmp r0,r1
    addlt r2,r2,#0x100000
    blt 1b

mmu_map_pt:
    ldr r0,=L1PT
    add r0,r0,#4
    ldr r2,=PTSECBASE
    str r2,[r0]

ttbmap:
    ldr r0,=L1PT
    mcr p15,0,r0,c2,c0,0

domainset:
    ldr r0,=DOMAINVAL
    mcr p15,0,r0,c3,c0,0

controlset:
    mov r0,#0
    mcr p15,0,r0,c7,c7,0 @ flush v3/v4 cache
    mcr p15,0,r0,c8,c7,0 @ flush v4 TLB

mrc p15,0,r0,c1,c0,0
bic r0,r0,#0x00002300 @ clear bits 13,9:8 (--V- --RS)
bic r0,r0,#0x00000087 @ clear bits 7,2:0 (B--- -CAM)
orr r0,r0,#0x00000002 @ set bit 2 (A) Align

```

```

orr r0,r0,#0x00001000 @ set bit 12 (I) I-Cache
orr r0,r0,#0x00000004 @ set bit 2 (C) D-Cache
orr r0,r0,#0x00000001 @ set bit 0 (M) MMU enable
mcr p15,0,r0,c1,c0,0

mov pc,lr

```

mmusetup.S 파일에서는 가상주소를 이용하기 위하여 페이지 테이블을 작성하고 MMU를 활성화 한다. ARM920T에서 가상주소를 활성화하기 위해서는 다음과 같은 동작을 순서대로 수행해야 한다.

1. level 1 translation table과 level 2 translation table을 작성한다. 각 테이블의 엔트리는 가상주소를 물리주소로 연결하는 부분, 메모리 보호역할을 하는 부분, 캐시와 write buffer에 대한 사용여부를 결정하는 부분으로 나뉜다.
2. CP15의 c2 레지스터(translation table base register, ttbr)가 level 1 translation table의 시작 주소를 가리키도록 한다.
3. CP15의 c3 레지스터(domain access control register, dacr)를 설정한다.
4. CP15의 c7 레지스터(cache operation register)를 이용하여 Instruction cache, Data cache를 초기화한다.
5. CP15의 c8 레지스터(TLB operation register)를 이용하여 TLB를 초기화한다.
6. CP15의 c1 레지스터(control register)를 이용하여 Instruction cache, Data cache, MMU를 활성화한다.

그럼 코드의 내용을 구체적으로 살펴보기로 하자. 먼저 필자가 사용하는 보드의 메모리 맵은 다음과 같다.

```

0x0000_0000~0x0010_0000:1MB NOR flash ROM(boot ROM)
0x3000_0000~0x3400_0000:64MB SDRAM
0x4800_0000~0x6000_0000:SFR

```

SDRAM 영역에는 다음과 같은 소프트웨어가 올라간다.

0x3000\_0000~0x3000\_5000:level 1, level 2 translation table 영역

0x3010\_0000~0x3020\_0000:cuteOS 영역

이 중 (0x3000\_0000~0x3000\_5000)의 영역은 다음과 같이 세분화되어 있다.

0x3000\_0000~0x3000\_4000:level 1 translation table(L1PT)

0x3000\_4000~0x3000\_4400:level 2 translation table(L2PT1)

0x3000\_4400~0x3000\_4800:level 2 translation table(L2PT2)

0x3000\_4800~0x3000\_4c00:level 2 translation table(L2PT3)

0x3000\_4c00~0x3000\_5000:level 2 translation

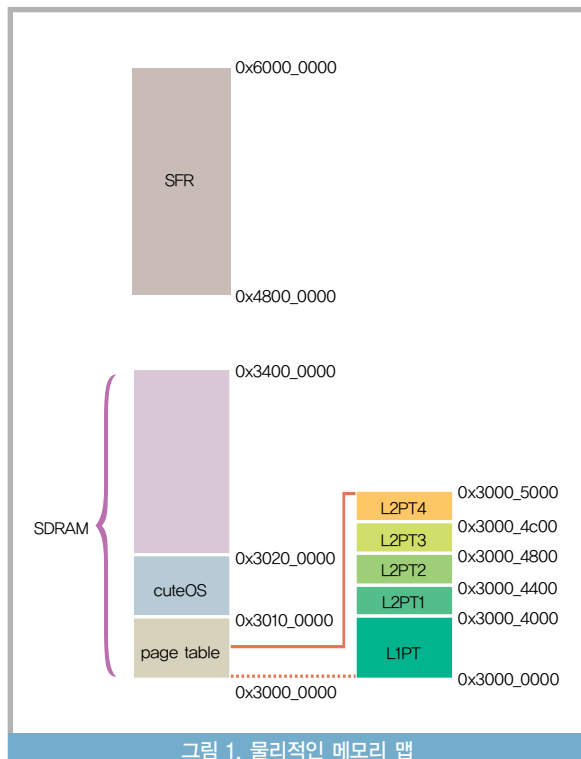


그림 1. 물리적인 메모리 맵

table(L2PT4)

물리적인 메모리 맵의 내용은 그림 1과 같이 나타낼 수 있다.

참고로 level 1 translation table은 translation table, level 2 translation table은 page table이라고도 한다. 또는 둘 다 page table이라고도 한다. 이후에는 이러한 용어들을 혼용해서 사용하기로 하겠다.

mmusetup 함수는 크게 일곱 부분으로 이루어져 있다.

1. mmu\_init\_pt - 페이지 테이블을 초기화 한다.
2. mmu\_map\_kernel - 커널 영역을 페이지 테이블에 맵핑한다.
3. mmu\_map\_sfr - SFR(Special Function Registers) 영역을 페이지 테이블에 맵핑한다.
4. mmu\_map\_pt - 페이지 테이블이 있는 영역을 페이지 테이블에 맵핑한다.
5. ttbmap - CP15의 c2 레지스터가 L1PT의 시작 주소를 가리키도록 한다.
6. domainset - CP15의 c3 레지스터를 초기화한다.
7. controlset - CP15의 c7 레지스터를 이용하여 Instruction cache, Data cache를 초기화하고, CP15의 c8 레지스터를 이용하여 TLB를 초기화한 후에 CP15의 c1 레지스터를 이용하여 Instruction cache, Data cache, MMU를 활성화한다.

여기서는 설명의 편의상 mmu\_init\_pt, mmu\_map\_sfr, domainset, mmu\_map\_kernel, mmu\_map\_pt, ttbmap, controlset의 순서로 살펴보기로 하겠다.

먼저 mmu\_init\_pt 부분에서는 L1PT, L2PT1, L2PT2, L2PT3, L2PT4 테이블을 초기화한다. L1PT는 level 1 translation table로 사용할 것이며, L2PT1은 커널 영역을 관리할 level 2 page table로 사용할 것이다. 또 L2PT2, L2PT3는 각각 일반 task의 영역을 관리할 level 2 page table로 사용할 것이다. task에 대한 추가와 L2PT2, L2PT3 page table에 대한 설정은 다음 기회에 다룰 것이다. mmu\_init\_pt 루틴의 수행과정은 그림 2와 같이 나타낼 수 있다.

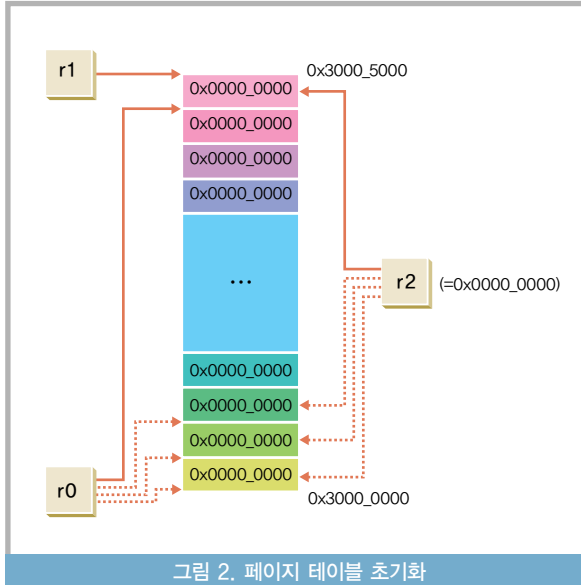


그림 2. 페이지 테이블 초기화

level 1 translation table의 크기는 16KB이며, 총 4096개의 엔트리를 가진다. 엔트리 하나의 크기는 4B이며, 1MB의 가상주소 공간에 대한 물리주소로의 맵핑 정보와 메모리 접근 권한 정보, 캐시와 write buffer의 사용여부에 대한 정보를 가진다. level 1 translation table 엔트리에는 그림 3과 같은 타입의 값이 들어갈 수 있다.



그림 3. level 1 descriptor

Level 2 page table의 크기는 1KB 또는 4KB이며, 각각 256 또는 1024개의 엔트리를 가진다. 엔트리 하나의 크기는 4B이며, 64KB, 4KB 또는 1KB의 가상주소 공간에 대한 물리주소로의 맵핑 정보와 메모리 접근 권한 정보, 캐시와 write buffer의 사용여부에 대한 정보를 가진다.

여기서는 1KB의 256개 엔트리를 갖는 Course page table을 사용할 것이며, 각각의 엔트리는 4KB의 가상주소 공간을 물리주소로 맵핑하게 된다. level 2 page table 엔트리에는 그림 4와 같은 타입의 값이 들어갈 수 있다.

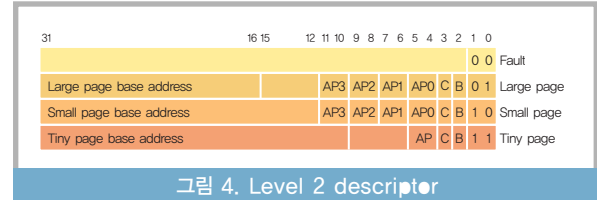


그림 4. Level 2 descriptor

mmu\_map\_sfr 부분에서는 SFR 영역(0x4800\_0000~0x5fff\_fff)을 가상주소 공간의 (0xe800\_0000~0xffff\_fff) 영역에 맵핑시킨다. mmu\_map\_sfr 루틴의 수행 과정은 그림 5와 같이 나타낼 수 있다.

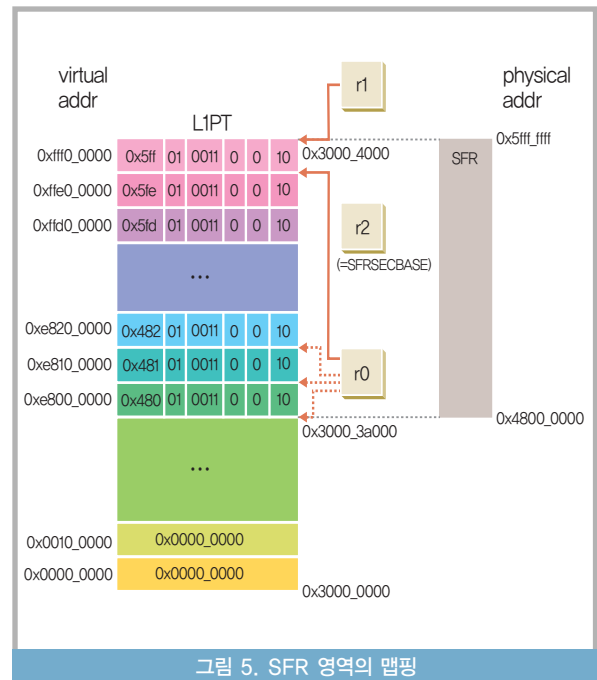
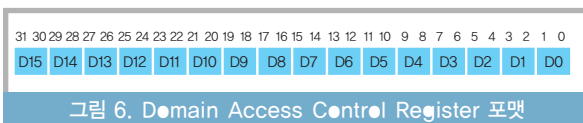


그림 5. SFR 영역의 맵핑

그림 5에서 L1PT의 0x3000\_3a00 번지에 있는 엔트리를 살펴보자. 이 엔트리는 Section descriptor이며 가상주소 (0xe800\_0000~0xe810\_0000) 영역을 물리주소 (0x4800\_0000~0x4810\_0000)에 맵핑시킨다. 또 AP(access permission) 필드의 값은 이진수 01이며, DS(Domain Selector) 필드의 값은 이진수 0011이다. SFR 영역이기 때문에 C(cacheable) bit와 B(bufferable) bit를 0으로 설정함으로써 non-cacheable, non-bufferable 영역으로 설정했다.

이 중 DS 필드와 AP 필드는 밀접하게 관련되어 있으며, MMU가 메모리를 보호하기 위해서 사용한다. DS 필드는 CP15의 c3(Domain Access Control Register)과 밀접하게 관련되어 있다. dacr(Domain Access Control Register) 레지스터는 32bit이며, 2bit씩 하나로 묶어 총 16개의 도메인을 나타낸다. 그림 6은 Domain Access Control Register의 포맷이다.



각각의 도메인은 2bit 크기를 가지며, 설정값에 따라 해당 주소 영역에 대한 접근 방법이 달라진다. 도메인의 값이 이진수 01일 경우에는 level 1 translation table 또는 level 2 page table 엔트리의 AP값에 따라 해당 주소 영역에 대한 접근 방법이 결정된다. 도메인의 값이 11일 경우에는 AP값을 무시하고, user mode나 supervisor mode에서 무조건 해당 주소 영역을 접근할 수 있다. 도메인의 값이 00일 경우에는 모든 접근에 대해서 도메인 fault가 발생한다. AP 필드의 경우 설정값에 따라 user mode나 supervisor mode에서 모두 읽기/쓰기가 가능하거나, supervisor mode에서만 읽기/쓰기가 가능하게 하는 등의 역할을 한다.

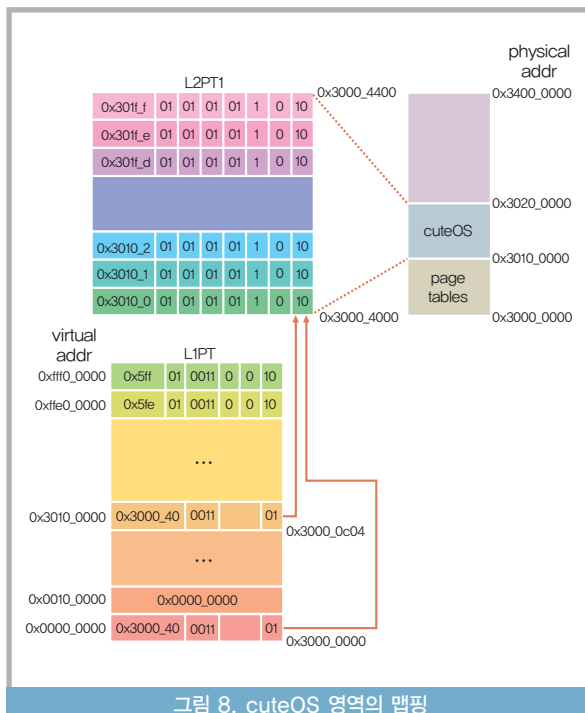
mmusetup 함수의 domainset 루틴에서는 CP15의 c3 레지스터에 그림 7과 같이 도메인 접근 정보를 넣는다.



L1PT의 0x3000\_3a00 번지에 있는 엔트리의 DS 필드는 그 값이 이진수 0011이며 dacr 레지스터의 D3 필드를 가리킨다. D3 필드의 값은 이진수 01이므로 AP 필드의 값에 따라 해당 주소 영역(0x4800\_0000~0x4810\_0000)에 대한 접근 권한이 결정된다. L1PT의 0x3000\_3a00 번지에 있는 엔트리의 AP 필드의 값은 01이며, 이는 supervisor mode에서만 해당 주소 영역에 대해 읽기/쓰기가 가능하다. user mode에서는 접근이 허용되지 않는다.

mmu\_map\_kernel 부분에서는 cuteOS 영역(0x3010\_0000~0x3020\_0000)을 가상주소 공간(0x0000\_0000~0x0010\_0000) 영역에 맵핑시킨다. mmu\_map\_kernel 루틴의 수행과정은 그림 8과 같이 나타낼 수 있다. CuteOS의 시작 주소를 가상주소 0x0000\_0000 번지에 맵핑시키는 이유는 exception vector table을 cuteOS의 시작 위치에 놓을 예정이기 때문이다.

L2PT1의 각 엔트리는 small page에 대한 descriptor이다. small page의 크기는 4KB이다. 각 엔트리의 C bit는 1로 설정하여, cuteOS 영역을 cacheable한 영역으로 설정한다. small page에 대한 level 2 descriptor에는 4개의 AP 필드가 존재하는데, 이는 4KB 페이지를 4개로 나누어 1KB 단위로 해당 영역에 대한 접근 권한을 설정할 수 있도록 하기 위함이다. 또한, mmu\_map\_kernel 부분에서는 cuteOS 영역(0x3010\_0000~0x3020\_0000)을 가상주소 공간(0x3010\_0000~0x3020\_0000) 영역에 맵핑시킨다.



먼저 Makefile의 내용부터 살펴보기로 하자. Makefile의 내용은 다음과 같다.

```

Makefile
.c.o:
    arm-linux-gcc $< -c

.S.o:
    arm-linux-gcc $< -c

# BOOTLOADER
OBJ = start.o led.o clksetup.o memsetup.o cuteOS.bin.o

cute-boot: $(OBJ)
    arm-linux-ld $(OBJ) -o cute-boot -Ttext 0x00000000
    -N -T cute-boot.lds
    arm-linux-objcopy cute-boot cute-boot.bin -O binary

start.o: start.S
    arm-linux-gcc start.S -c -DOS_RAM_BASE=0x30100000

# KERNEL
KERNEL_OBJ = head.o main.o mmusetup.o

cuteOS.bin.o: cuteOS
    arm-linux-ld -r -o cuteOS.bin.o -b binary cuteOS.bin

cuteOS: $(KERNEL_OBJ)
    arm-linux-ld $(KERNEL_OBJ) -o cuteOS -Ttext 0x000
    00000 -N
    arm-linux-objcopy cuteOS cuteOS.bin -O binary

clean:
    rm -f *.o
    rm -f cute-boot
    rm -f cute-boot.bin
    rm -f cuteOS
    rm -f cuteOS.bin

```

Makefile에서 음영이 들어간 부분을 살펴보자. 여기서 cuteOS가 동작할 주소를 0x00000000 번지로 설정했다. 앞에서도 말했듯이 exception vector table을 cuteOS의 시작위치에 놓을 예정이기 때문에 이와 같이 처리했다. 이 경우 생성한 커널의 메모리 맵을 보면 다음과 같다.

```

$ arm-linux-nm cuteOS -n
00000000 T _start
00000004 t stack_setup
00000012 a IRQ32MOD
00000013 a SVC32MOD
0000001f a SYS32MOD
0000002c t gcc2_compiled.
0000002c T kmain
000000c0 a NOINT
000000e8 t mmu_init_pt
000000e8 T mmusetup
00000100 t mmu_map_kernel
00000134 t mmu_map_sfr
00000150 t mmu_map_pt
00000160 t ttbmap
00000168 t domainset
00000170 t controlset
000001b4 A __bss_end__
000001b4 A __bss_start
000001b4 A __bss_start__
000001b4 D __data_start
000001b4 A __end__
000001b4 A __bss_end__
000001b4 A _edata
000001b4 A _end

```

여기서 kmain 함수의 위치는 0x0000002c 번지임을 알 수 있다. 따라서 head.S 파일의 마지막에 놓인 다음 명령어에서 pc 레지스터에 kmain 함수의 주소인 0x0000002c 값이 들어간다.



```
ldr pc, =kmain
```

head.S 파일의 \_start 번지에서는 다음 명령어에 의해 페이지 테이블을 작성하고, MMU를 활성화한다.

```
bl mmusetup
```

따라서 kmain 함수의 주소인 0x0000002c 값은 가상주소가 된다. 이 경우 MMU가 이미 활성화되어 있기 때문에 LIPT의 첫 번째 엔트리에 의해서 물리주소 0x3010002c 번지로 맵핑된다.

MMU가 활성화되는 시점은 mmusetup 함수내에 있는 controlset 루틴의 마지막 부분에 있는 다음 명령어가 수행되고 나서이다.

```
mcr p15,0,r0,c1,c0,0
```

즉, \_start 번지에 있는 명령어부터 시작해서 이 명령어까지는 물리주소를 이용해 수행되며, 다음 명령어부터는 가상주소를 이용해 수행된다. 즉, 이 명령어 이후에는 내부적으로 MMU가 동작하게 된다. cuteOS는 수행 초기에 물리주소를 사용하며, pc 레지스터의 초기값은 0x30100000이 된다. <arm-linux-objdump cuteOS -D>를 이용하여 <mcr p15,0,r0,c1,c0,0> 명령어의 물리주소값을 확인해 보면, 0x30100198 번지가 됨을 알 수 있다. 이 명령어를 수행하기 위해 pc 레지스터의 값은 0x30100198이 된다. 또한 이 명령어 다음에 오는 명령어의 주소 값은 0x3010019c가 된다.

이후에 오는 명령어를 수행하기 위해 pc 레지스터의 값은 0x3010019c가 된다. 그런데 0x30100198의 경우는 물리주소로 MMU를 거치지 않지만, 0x3010019c의 경우는 가상주소로 MMU를 거치게 된다. 즉, \_start 루틴과 mmusetup 함수는 0x30100000 번지 근처에서 동작하고, kmain 함수를 비롯해 나머지 루틴은 0x00000000 번지 근처에서 동작한다.

이 중 \_start 번지부터 controlset 루틴의 <mcr

p15,0,r0,c1,c0,0> 명령어까지는 물리주소 0x30100000 번지 근처에서 동작하고, 이 명령어 다음부터 kmain 루틴으로 뛰는 <ldr pc, =kmain> 명령어까지는 가상주소 0x30100000 번지 근처에서 동작한다. 일단 kmain 루틴으로 뛰면 가상주소 0x00000000 번지 근처에서 동작한다. 따라서 <mcr p15,0,r0,c1,c0,0> 명령어 다음부터 kmain 루틴으로 뛰는 <ldr pc, =kmain> 명령어까지 사용하는 가상주소 공간에 대해 물리주소로의 맵핑이 필요하다.

Makefile의 앞부분에서 다음과 같은 문장을 볼 수 있다.

```
.c.o:
arm-linux-gcc $< -c
```

이를 suffix rule이라고 한다. 여기서는 .c로 끝나는 파일을 .o 파일로 바꾸는 역할을 한다. .c로 끝나는 파일을 .o파일로 바꾸기 위해 여기서는 <arm-linux-gcc \$< -c> 명령어를 쓴다. \$<는 .c로 끝나는 임의의 파일을 뜻한다. 예를 들어 main.c 파일이 있을 경우 main.o 파일로 바꾸기 위해 <arm-linux-gcc main.c -c> 명령어를 수행하게 된다.

mmu\_map\_pt 부분에서는 페이지 테이블 영역(0x3000\_0000~0x3010\_0000)을 가상주소 공간(0x0010\_0000~0x0020\_0000) 영역에 맵핑시킨다. mmu\_map\_pt 루틴의 수행과정은 그림 9와 같이 나타낼 수 있다.

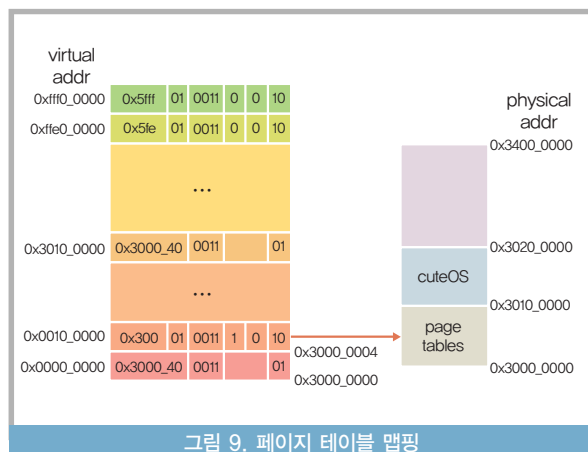


그림 9. 페이지 테이블 맵핑



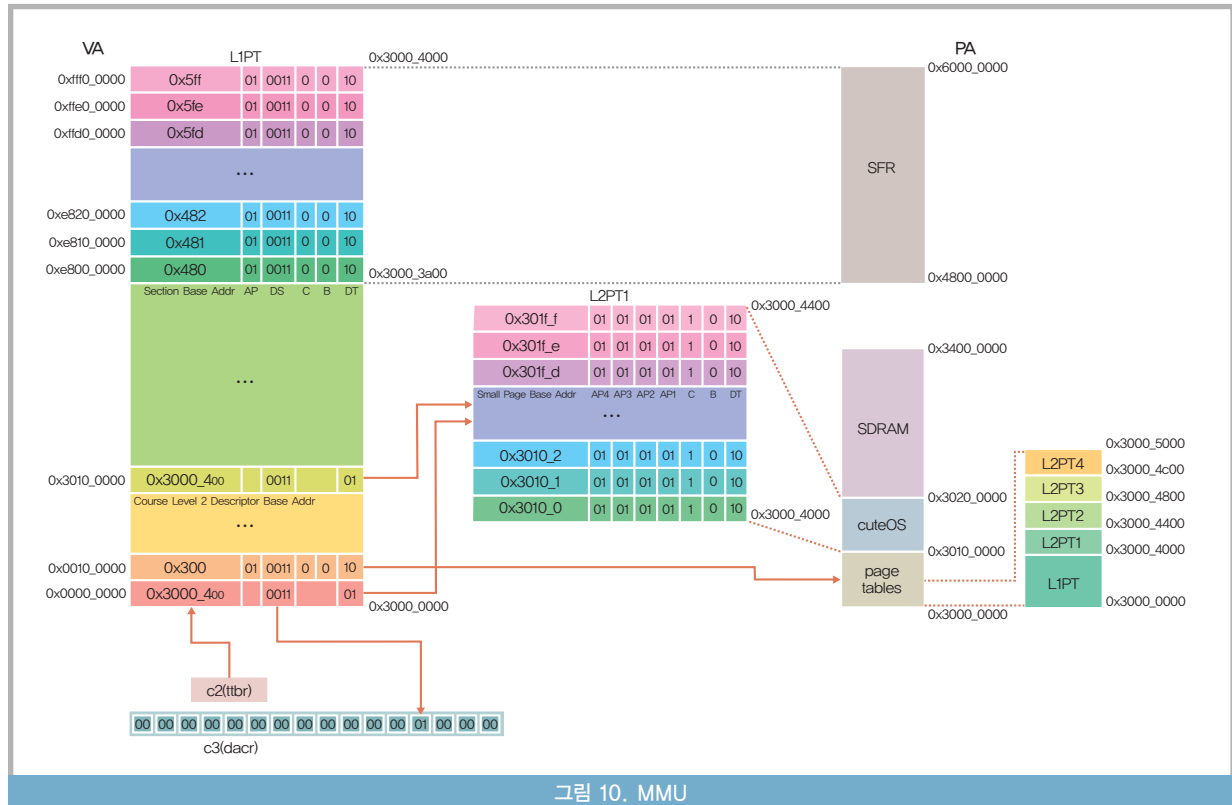


그림 10. MMU

```
main.c
#define rGPFDAT (*(volatile unsigned long *)0xf6000054)

void kmain(void)
{
    int i;
    while(1) {
        rGPFDAT = rGPFDAT | 0x000000f0;
        for(i=0; i<0x100000; i++);
        rGPFDAT = rGPFDAT & ~0x000000f0;
        for(i=0; i<0x100000; i++);
    }
}
```

페이지 테이블 영역을 가상주소 공간상에 맵핑시키는 이유는 뒤에서 task의 주소공간을 맵핑시킬 때 페이지 테이블을 접근해야 하기 때문이다. 또 task간 문맥전환을 수행할 경우

에도 페이지 테이블에 대한 접근이 필요하다.

ttbmap 루틴에서는 L1PT의 주소값을 CP15의 c2 레지스터에 넣음으로써 MMU가 level 1 translation table에 접근할 수 있게 한다. 마지막으로 controlset 루틴에서는 Instruction Cache, Data Cache, TLB를 초기화하고, Instruction Cache, Data Cache, MMU를 활성화시킨다. 이상의 과정을 전체적으로 나타내면 그림 10과 같다.

마지막으로 main.c 파일의 내용이다. 앞에서 MMU를 활성화했기 때문에 rGPFDAT 레지스터를 다음 주소 (0xf6000054)로 접근할 수 있다.

이상에서 MMU와 캐시 설정 부분을 작성해 보았다. 다음 기사에서는 디버깅 메시지를 보기 위해 uart 관련 루틴을 작성해 볼 것이다. 또, exception vector table을 작성해 보고, system call 루틴도 추가해 보기로 하자. <sup>Real Time</sup>