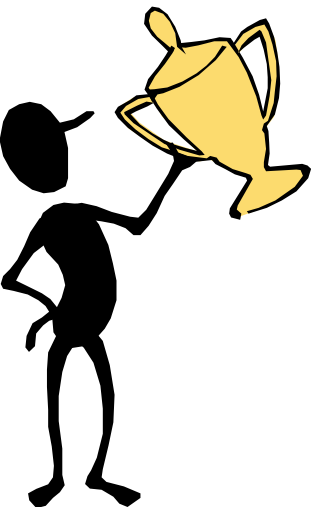
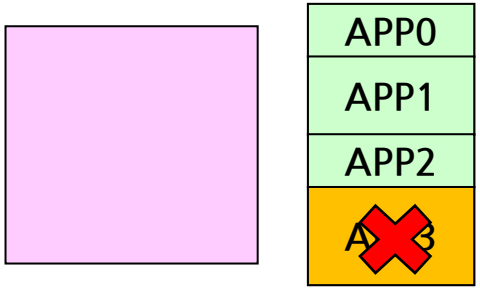
Demand Paging의 검토



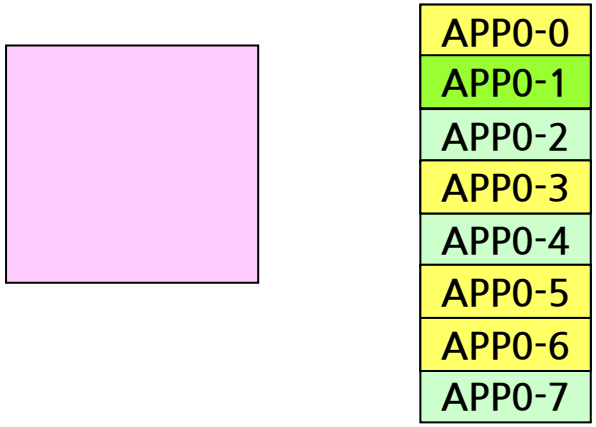
Demand Paging

- OS가 APP 구동시 메모리를 고정 배정할 경우 메모리 효율이 저하된다
  - 따라서 APP과 메모리를 Page로 나누고 Page단위로 필요 시만 적재한다
    - 즉, APP은 NAND나 HDD에 들어 있고 필요할 때마다 Page 단위로 RAM으로 적재한다
    - 이러한 방식을 Demand Paging이라 한다
    - 일반적으로 OS는 APP을 CODE, RO, RW, STACK을 분리하여 Page 단위로 적재한다

APP들을 고정 배치하는 경우



하나의 APP을 필요 시마다 분할 적재하는 경우



  - Page 단위 적재를 하면 메모리가 꽉 차 있을 경우 다른 Page를 제거하고 적재된다
    - 따라서 여러 APP들을 적은 메모리로 동시에 구동할 수 있다
    - 단, 스택과 힙은 Replacement가 발생하지 않으므로 부족 시 해당 APP 실행이 중지된다

Demand Paging의 고려 사항

APP이 NAND나 HDD에 있는 경우

- 변수 영역이 Page 적재되었다가 수정되면 교체될 때 반드시 백업이 필요하다
  - 그러나 일반적으로 HDD, NAND 메모리 등은 Write 속도가 느리고 Block 쓰기가 필요하다
  - 만약 백업을 하게 된다면? 원본 프로그램의 손상이 발생하여 재 실행 시 문제된다
    - 해결책 1. RW 영역의 복사본을 저장장치에 저장해둔 후 프로그램 종료 시 원본을 복구한다
    - 해결책 2. RW 속성(BSS 포함)이 필요한 영역은 고정 적재하여 교체되지 않도록 한다
  - 스택은 원래 실행파일에 들어 있지 않고 필요 시 배정하므로 백업 문제는 발생하지 않는다
  - 일반 멀티프로세싱 OS는 APP을 속성에 따라 분할 적재되도록 Relocation을 이용한다
    - 이 경우 APP도 Relocatable또는 PI(Position Independent)하게 컴파일 되어야 한다

Demand Paging 고려 사항

- PABT, DABT 발생 : 해당 Page 영역을 읽어서 비어있는 Page에 배정
  - 비어있는 page가 없으면 기존 page를 replace해야 한다 ➔ 정책은?
  - 단, 변수 등의 영역이 replace된다면 무조건 삭제가 아니라 원본에 백업되어야 한다
  - OS는 page 배정 상황을 표로 관리하여 paging을 구현해야 한다

실험용 App을 위한 Link Script 파일

CODEXPERT
임베디드 소프트웨어 연구소

- 실험을 위한 APP은 다음과 같이 실행파일을 생성한다
  - CODE에서 Page간 분기가 발생하도록 4KB 이상으로 만들되 서로 상호 호출 되도록 설계
    - Crt0.s에 LED_ON 등과 같은 ASM 함수 만들어서 Main에서 호출하도록 설계한다

```
SECTIONS
{
    .text :
    {
      crt0.o(.text)
      . = ALIGN(4096);
       *(.text)
       . = ALIGN(4);
    } > REGION_TEXT

    .rodata : {...} > REGION_RODATA
    .data : {...} > REGION_DATA
    .bss :{...} > REGION_BSS
}
```

APP의 LDS 파일

Crt0.s 코드와 다른 코드들을 4KB 분리 배치하여
Main.c에서 Crt0.s 함수들 호출시 Page 경계를 넘도록 설계

생성된 App의 BIN 파일 분석

CODE XPERT
임베디드 소프트웨어 연구소

실험을 위해 생성된 APP의 구조

```
00000000  0F 50 2D E9 30 00 9F E5 30 10 9F E5 00 20 A0 E3  .P-é0.Ÿå0.Ÿå.  ã
00000010  01 00 50 E1 04 20 80 34 FC FF FF 3A 2B 04 00 EB  ..Pá. €4üÿÿ:+..ë
00000020  0F 90 BD E8 00 00 00 EF 1E FF 2F E1 01 00 00 EF  ..½è...ï.ÿ/á...ï
00000030  1E FF 2F E1 02 00 00 EF 1E FF 2F E1 30 00 20 00  .ÿ/á...ï.ÿ/á0. .
00000040  34 00 20 00 00 00 00 00 00 00 00 00 00 00 00 00  4. .............
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

RO-CODE
Crt0.s 함수들

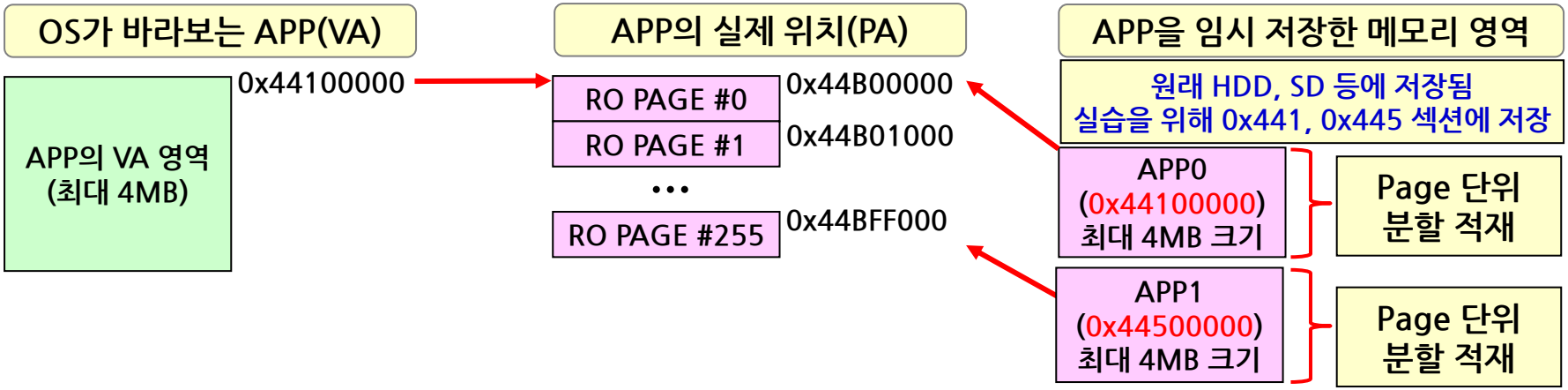
```
00000FE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001000  0D C0 A0 E1 00 D8 2D E9 04 B0 4C E2 10 D0 4D E2  .À á.Ø-é.°Lâ.ÐMâ
00001010  18 00 0B E5 00 30 A0 E3 10 30 0B E5 02 00 00 EA  ...å.0 ã.0.å...ê
00001020  10 30 1B E5 01 30 83 E2 10 30 0B E5 18 30 1B E5  .0.å.0ƒâ.0.å.0.å
00001030  10 27 02 E3 92 03 02 E0 10 30 1B E5 03 00 52 E1  .'.ã’..à.0.å..Rá
00001040  F6 FF FF CA 0C D0 4B E2 00 A8 9D E8 0D C0 A0 E1  öÿÿÊ.ÐKâ.¨.è.À á
00001050  00 D8 2D E9 04 B0 4C E2 04 30 00 E3 10 3A 4E E3  .Ø-é.°Lâ.0.ã.:Nã
```

RO-CODE
Main.c의 함수들
(4KB Align되어 있음)

Demand Paging 전략

CPU 입장에서 APP의 메모리 구조

- PA 0x44B00000 부터 2~32개의 지정한 Page수 만큼만 APP이 적재되어 사용됨
  - 따라서 2nd T/T의 0x44100000 ~ 0x441FFFFF 까지 256개 Entry가 동적 관리 되어야 함

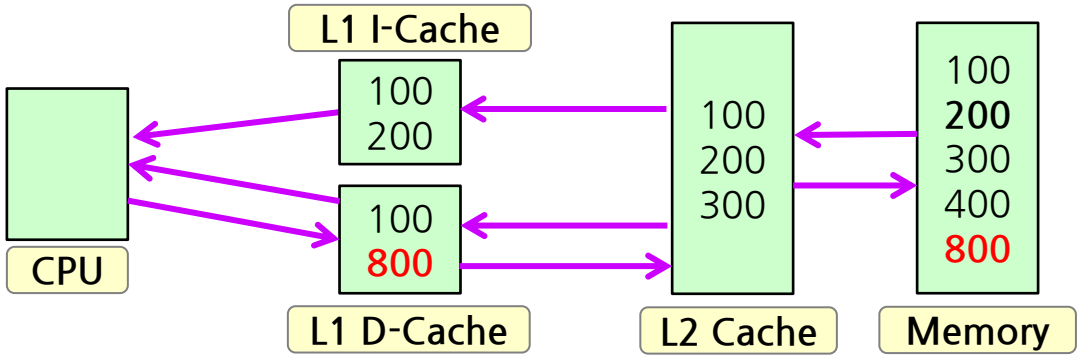


- APP의 RO 영역이 Page Fault 발생 때마다 동적으로 한정된 수의 Page에 적재
  - 2nd T/T에는 App이 적재된 Page는 접근 가능, 비어있는 Page는 접근 불가능으로 설정
  - 그리고 동일한 Page에는 적재된 App의 PA에 해당하는 주소로 변환 되도록 설정해야 함
  - Page 교체 함수는 적재, 백업 페이지의 정보(주소, 페이지 번호 등)을 인쇄하여 디버깅한다
- Page 영역(0x44B 섹션)은 어떤 Cache 속성으로 설정되어야 할까?

Demand Paging 영역의 Cache 속성

Demand Paging을 위한 영역의 Cache 속성 설정

- L1, L2 Cache가 모두 WBWA나 WB로 설정되어 있으면 어떻게 될까?
- WBWA_WBWA 모드 설정 시 문제가 발생하는 이유는?
  - Page에 적재하면(메모리에 쓰기) L2, L1-D Cache로 적재된다 ➔ L1 IC는 L2에서 로딩
  - 이후 같은 Page에 다른 영역 코드가 적재(쓰기)되면 L1 DC에만 변경된다
    - ✖ Page의 PA는 동일한데 서로 다른 RO 영역(다른 PA)의 데이터가 저장될 수 있게 된다
  - 이로 인하여 L1, L2 Cache가 불일치 된다 ➔ L1 IC는 L1 DC가 아닌 L2 Cache에서 로딩
  - 이 문제는 I-Cache가 VIPT 타입과 무관하게 발생한다 ➔ 따라서 L1은 WT, L2는 WBWA

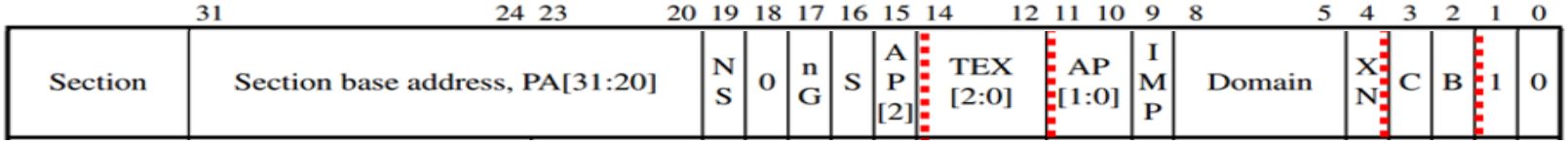


- Demand Paging 동작의 정상 여부 실험을 위하여 아래와 같이 먼저 실습해 본다
  - L1-D, L2 Cache Off로 설정하고 L1-I는 page가 변경될 때마다 Invalidate all을 수행한다

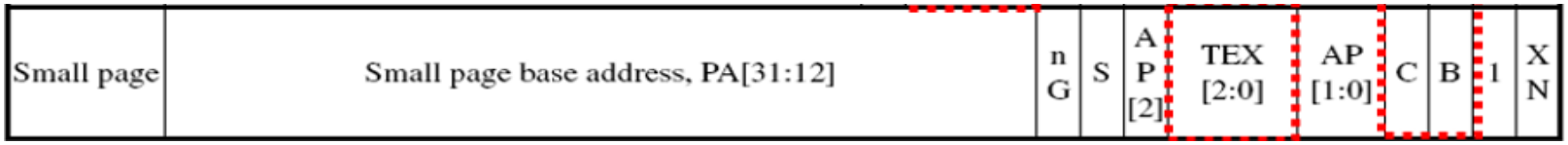
Inner, Outer Cache 속성 설정

CODEXPERT
임베디드 소프트웨어 연구소

- Demand Paging 영역은 Cache를 L1, L2 각각 설정하도록 하였다
  - 1st T/T에서 TEX[2]=1이면 Inner, Outer의 Cache 속성을 각각 설정할 수 있다



  - 2nd T/T에서 TEX[2]=1이면 Inner, Outer의 Cache 속성을 각각 설정할 수 있다



  - 이 설정을 위한 Option.h의 정의는 다음과 같다

1BB	A	A	Cacheable memory:	AA = Inner attribute[b]	Normal	S bit[a]
TEX[2:0]	C	B		BB = Outer attribute		

Encoding		Cache attribute
0	0	Non-cacheable
0	1	Write-Back, Write-Allocate
1	0	Write-Through, no Write-Allocate
1	1	Write-Back, no Write-Allocate

1차 Entry를 위한 설정

```
#define NC_NC          ((0x4<<12)|(0x0<<3)|(0x0<<2))
#define WBWA_WBWA      ((0x5<<12)|(0x0<<3)|(0x1<<2))
#define WT_WBWA        ((0x5<<12)|(0x1<<3)|(0x0<<2))
```

2차 Entry를 위한 설정

```
#define NC_NC_PAGE     ((0x4<<6)|(0x0<<3)|(0x0<<2))
#define WT_WBWA_PAGE   ((0x5<<6)|(0x1<<3)|(0x0<<2))
```