:restart

linux/arch/arm/boot/compressed/head.S

iamroot12b
https://github.com/iamroot12b/kernel

Background

- restart label
  - 커널 압축 해제를 위한 준비 작업
  - 현재 실행 중인 Image(head.o + misc.o + piggy.gz)와 압축 해제할 공간이 겹치는지 확인
  - 겹친다면 relocate 작업 후 restart label 다시 실행

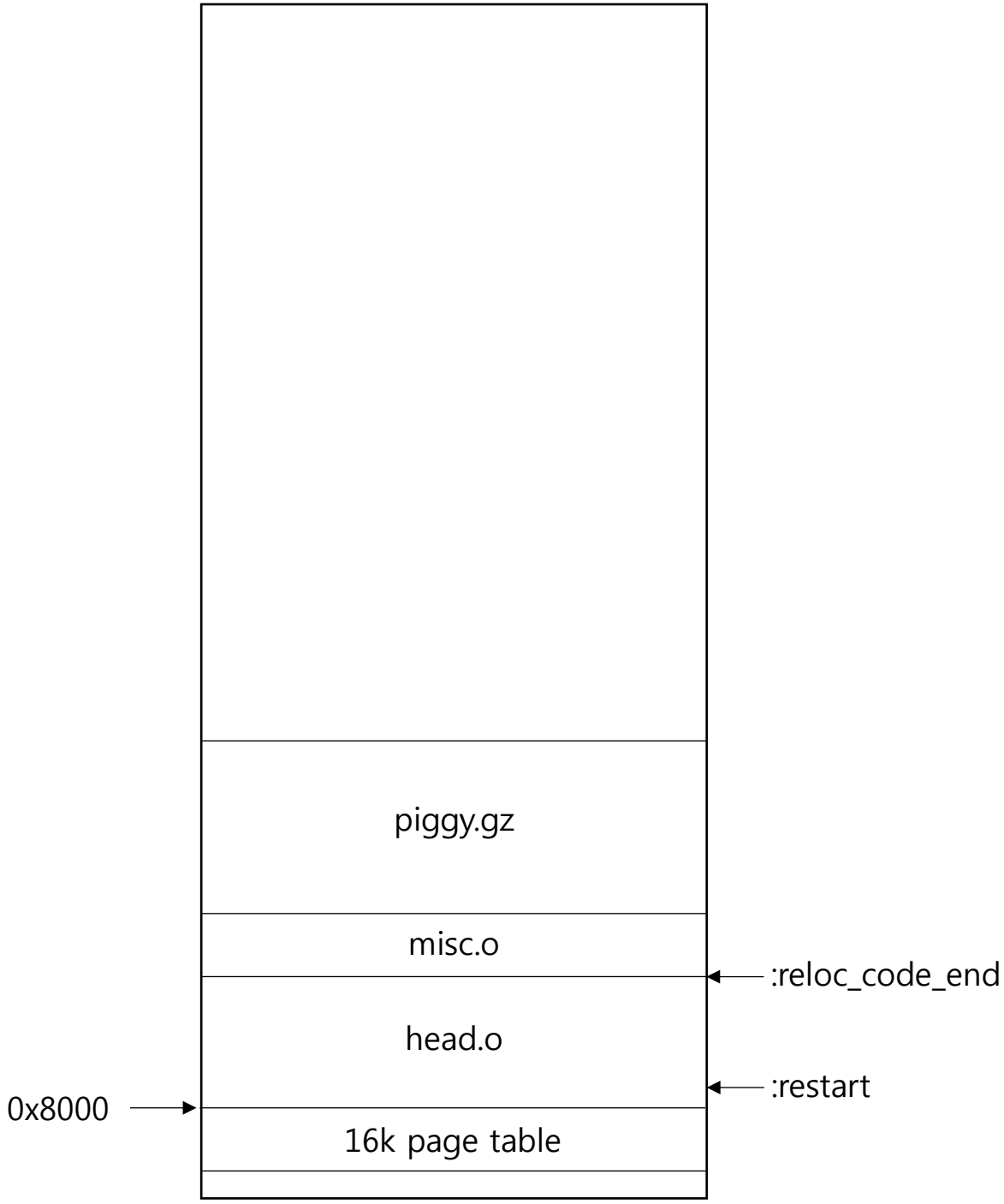
Assumption

- 커널 시작 위치는 0x8000 번지
- 압축 풀 위치도 0x8000 번지
  - bcm_2709의 경우 arch/arm/mach-bcm2709에 아래와 같이 정의 함
    - zreladdr-y := 0x00008000
- 아래 코드는 분석에서 제외
  - #define에 포함되지 않는 코드
    - #ifndef CONFIG_ZBOOT_ROM (not defined)
  - DTB
    - #ifdef CONFIG_ARM_APPENDED_DTB
  - Virtualization Extension
    - #define CONFIG_ARM_VIRT_EXT

Notation

- Memory address는 위쪽 방향으로 증가
- Memory layout 왼쪽은 address
- Memory layout 오른쪽은 label or register
- 현재 실행 하는 line은 붉은색으로 표기

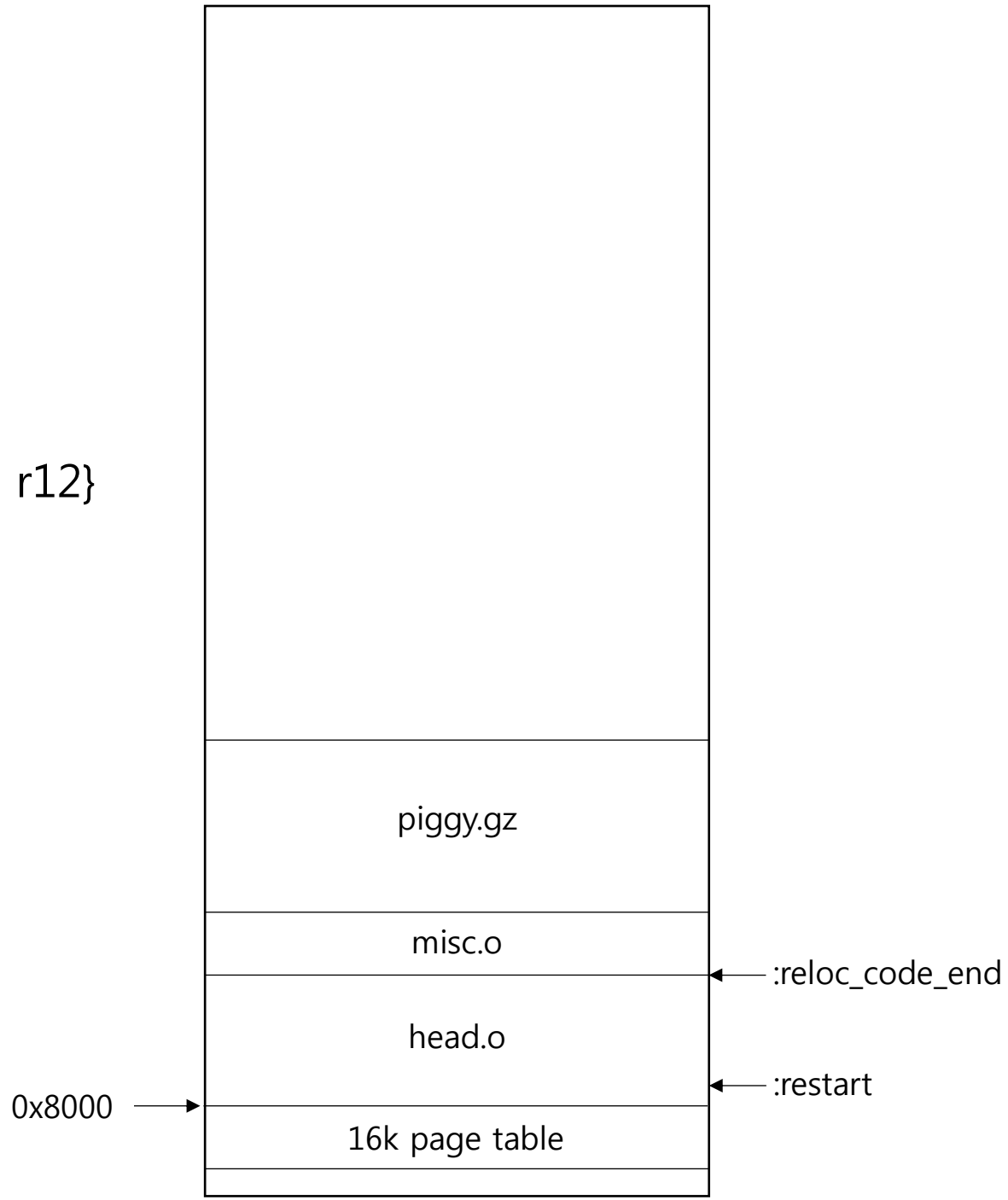
Initial state



1

```
restart:  adr     r0, LC0
          ldmia   r0, {r1, r2, r3, r6, r10, r11, r12}
          ldr     sp, [r0, #28]
```

```
570         .align  2
571         .type   LC0, #object
572 LC0:         .word   LC0         @ r1
573         .word   __bss_start     @ r2
574         .word   _end            @ r3
575         .word   _edata          @ r6
576         .word   input_data_end - 4  @ r10 (inflated size location)
577         .word   _got_start      @ r11
578         .word   _got_end        @ ip
579         .word   .L_user_stack_end   @ sp
580         .word   _end - restart + 16384 + 1024*1024
581         .size   LC0, . - LC0
```

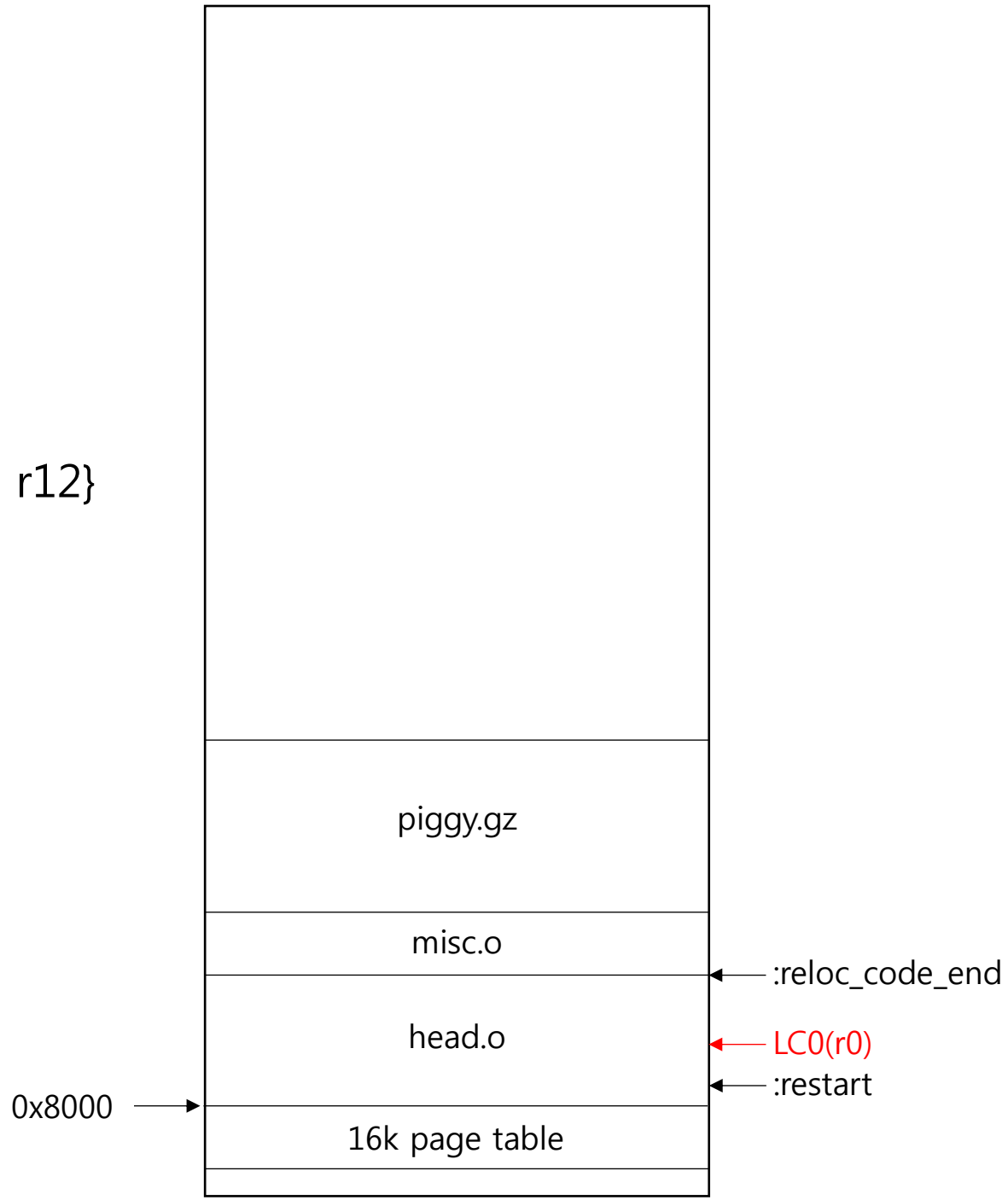


1

```
restart:  adr     r0, LC0
          ldmia   r0, {r1, r2, r3, r6, r10, r11, r12}
          ldr     sp, [r0, #28]
```

LC0의 pc 상대 주소를 r0에 저장
즉, 0x8000+LC0 label의 위치

```
570         .align  2
571         .type   LC0, #object
572 LC0:        .word   LC0         @ r1
573         .word   __bss_start     @ r2
574         .word   _end            @ r3
575         .word   _edata          @ r6
576         .word   input_data_end - 4  @ r10 (inflated size location)
577         .word   _got_start      @ r11
578         .word   _got_end        @ ip
579         .word   .L_user_stack_end   @ sp
580         .word   _end - restart + 16384 + 1024*1024
581         .size   LC0, . - LC0
```

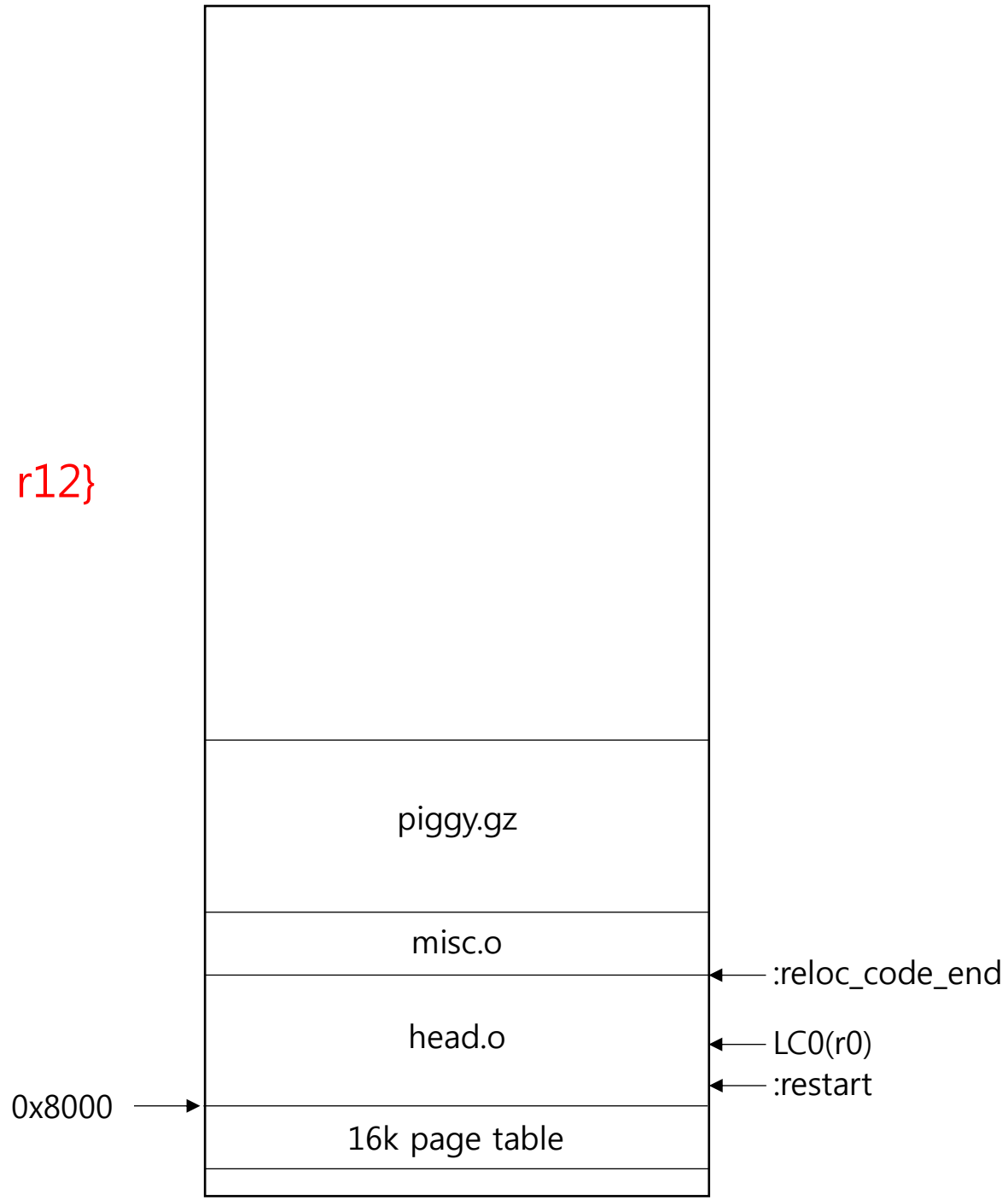


1

```
restart:  adr     r0, LC0
          ldmia   r0, {r1, r2, r3, r6, r10, r11, r12}
          ldr     sp, [r0, #28]
```

r0(LC0)의 주소를 시작으로 차례로
R1, r2, r3, r6, r10, r11, r12에 저장

```
570         .align  2
571         .type   LC0, #object
572 LC0:        .word   LC0         @ r1
573         .word   __bss_start     @ r2
574         .word   _end            @ r3
575         .word   _edata          @ r6
576         .word   input_data_end - 4  @ r10 (inflated size location)
577         .word   _got_start      @ r11
578         .word   _got_end        @ ip
579         .word   .L_user_stack_end   @ sp
580         .word   _end - restart + 16384 + 1024*1024
581         .size   LC0, . - LC0
```

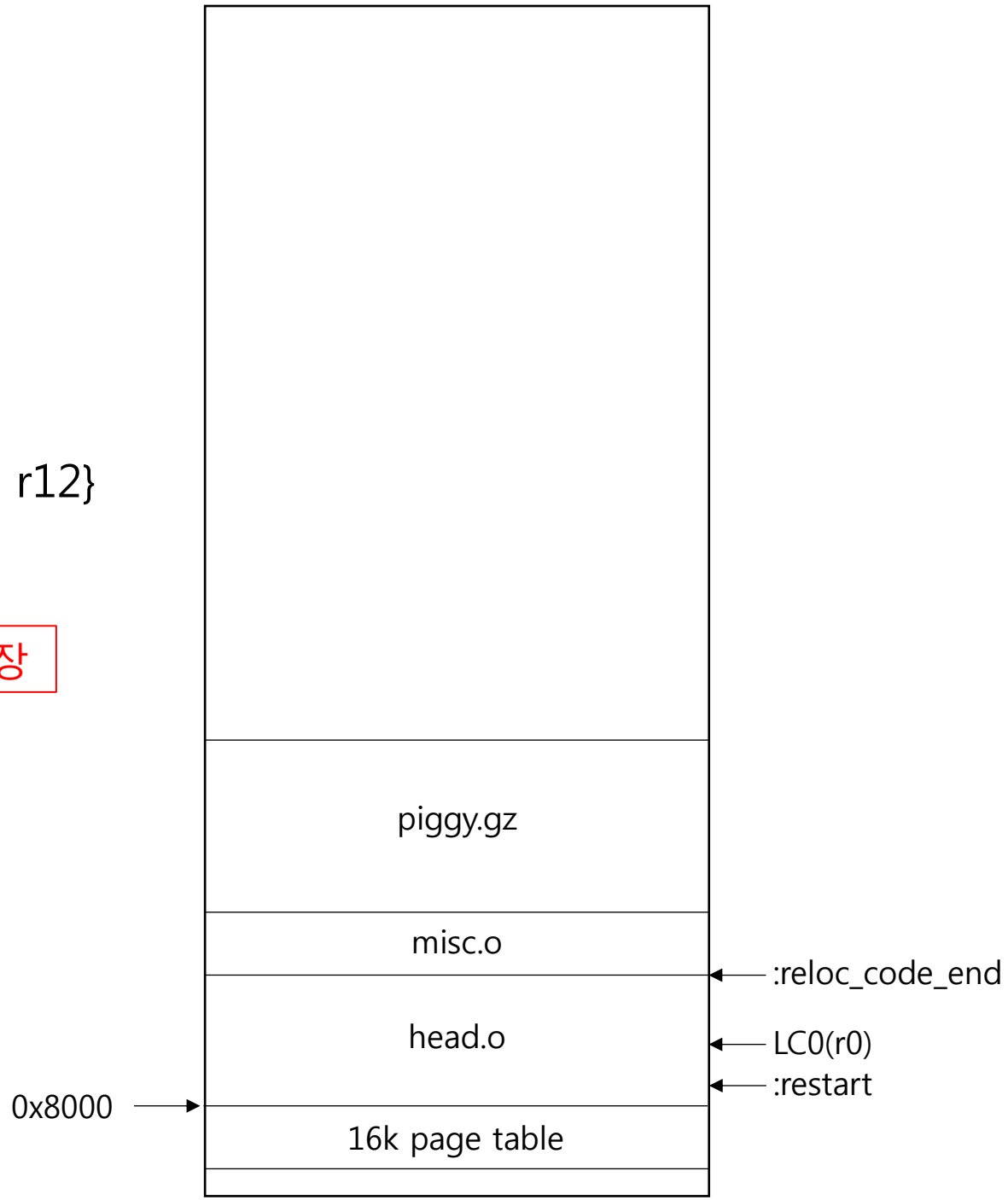


1

```
restart:  adr     r0, LC0
          ldmia   r0, {r1, r2, r3, r6, r10, r11, r12}
          ldr     sp, [r0, #28]
```

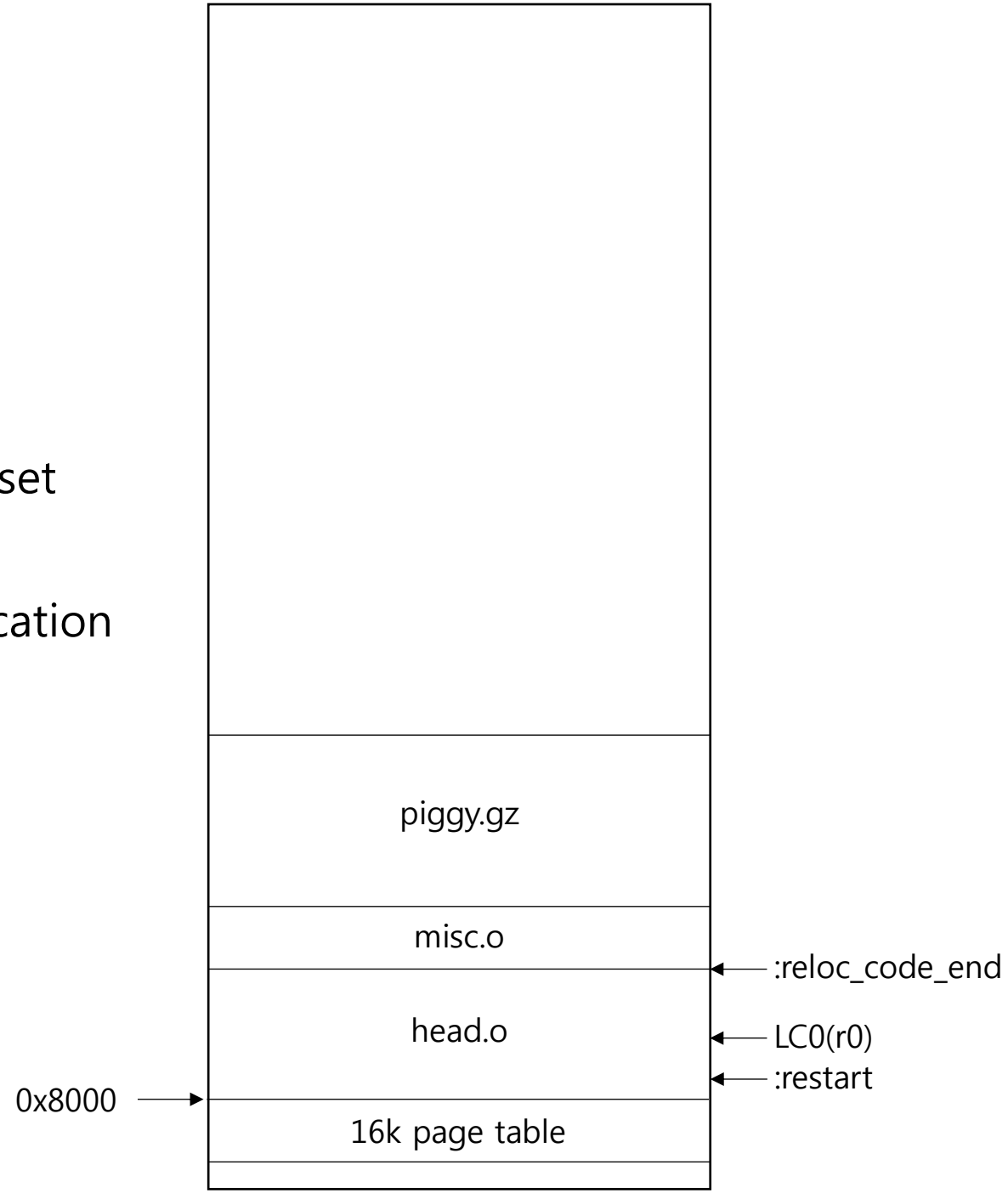
r0(LC0)+28번 위치의 값(user_stack_end)을 sp에 저장

```
570         .align  2
571         .type   LC0, #object
572 LC0:         .word   LC0         @ r1
573         .word   __bss_start     @ r2
574         .word   _end            @ r3
575         .word   _edata          @ r6
576         .word   input_data_end - 4  @ r10 (inflated size location)
577         .word   _got_start      @ r11
578         .word   _got_end        @ ip
579         .word   .L_user_stack_end   @ sp
580         .word   _end - restart + 16384 + 1024*1024
581         .size   LC0, . - LC0
```



2

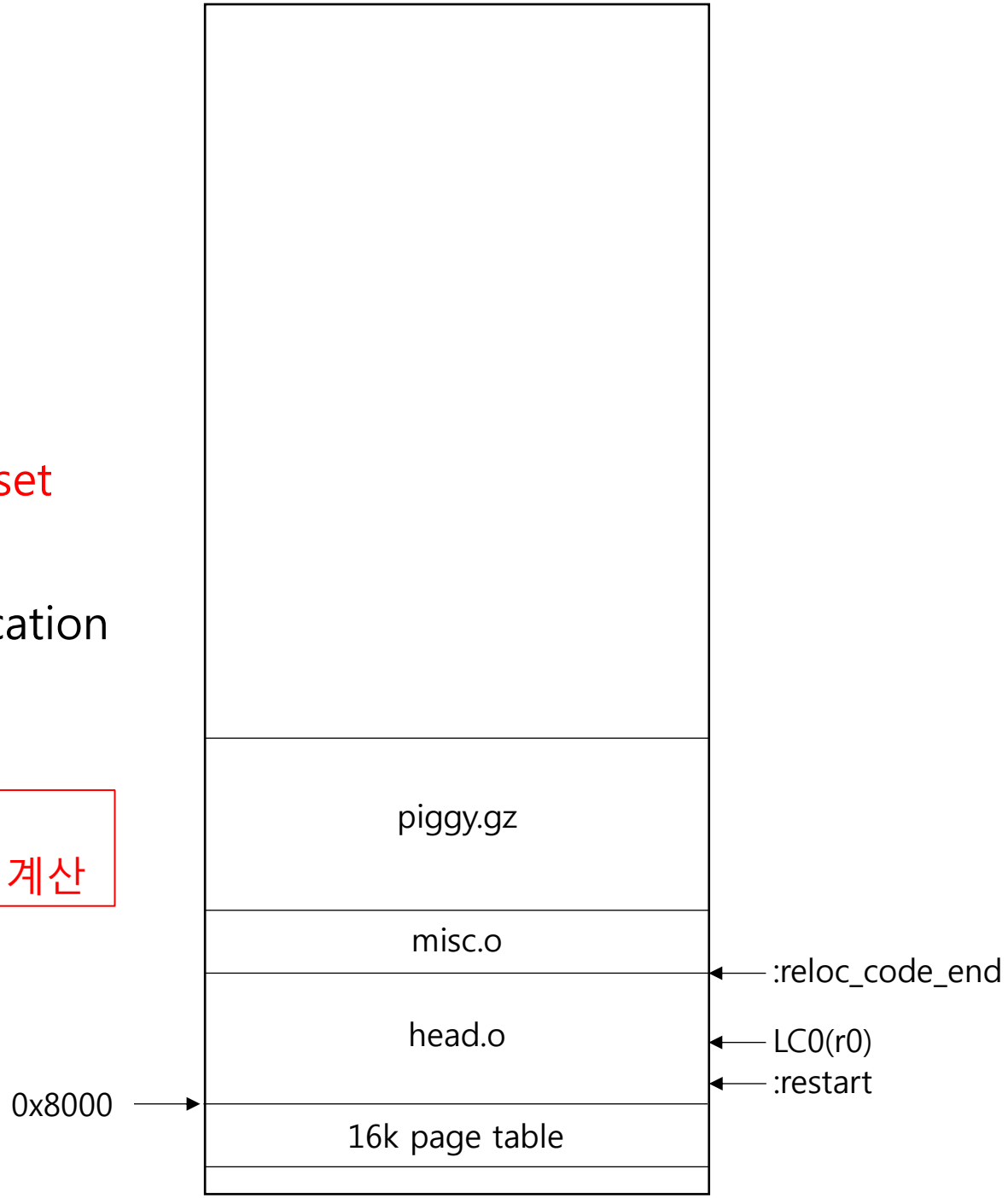
```
sub     r0, r0, r1      @ calculate the delta offset
add     r6, r6, r0      @ _edata
add     r10, r10, r0    @ inflated kernel size location
```



2

```
sub     r0, r0, r1      @ calculate the delta offset
add     r6, r6, r0      @ _edata
add     r10, r10, r0    @ inflated kernel size location
```

실제 메모리에 올라와 있는 LC0(r0)와
링커스크립트에 의해 배치된 LC0(r1) label 위치의 offset 계산

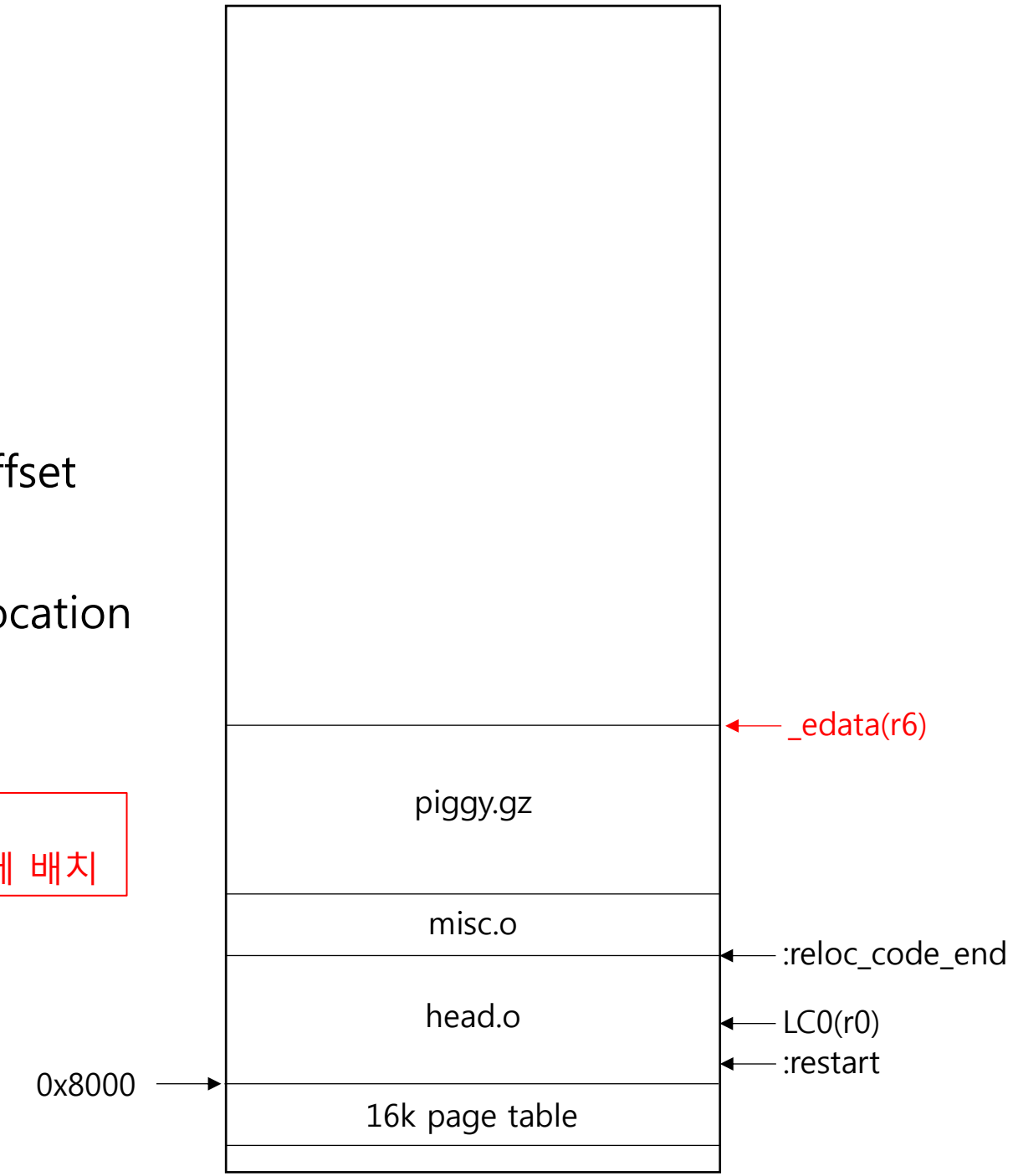


2

```
sub     r0, r0, r1          @ calculate the delta offset
add     r6, r6, r0          @ _edata
add     r10, r10, r0        @ inflated kernel size location
```

_edata에 offset 추가
vmlinux_lds.S를 참고하면 _edata는 piggydata+padding 이후에 배치

```
/* ensure the zImage file size is always a multiple of 64 bits */
/* (without a dummy byte, ld just ignores the empty section) */
.pad            : { BYTE(0); . = ALIGN(8); }
_edata = .;
```

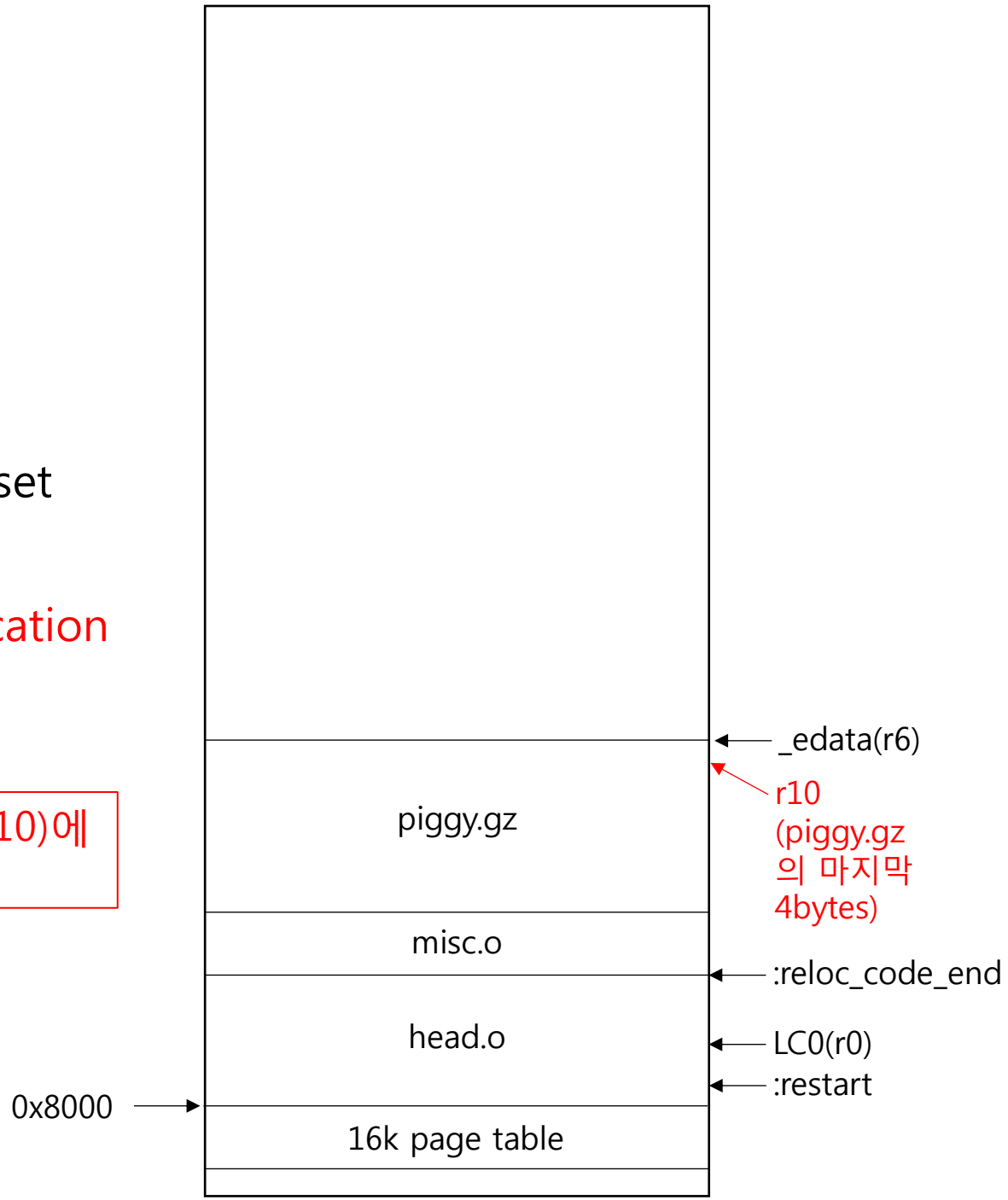


2

```
sub     r0, r0, r1      @ calculate the delta offset
add     r6, r6, r0      @ _edata
add     r10, r10, r0    @ inflated kernel size location
```

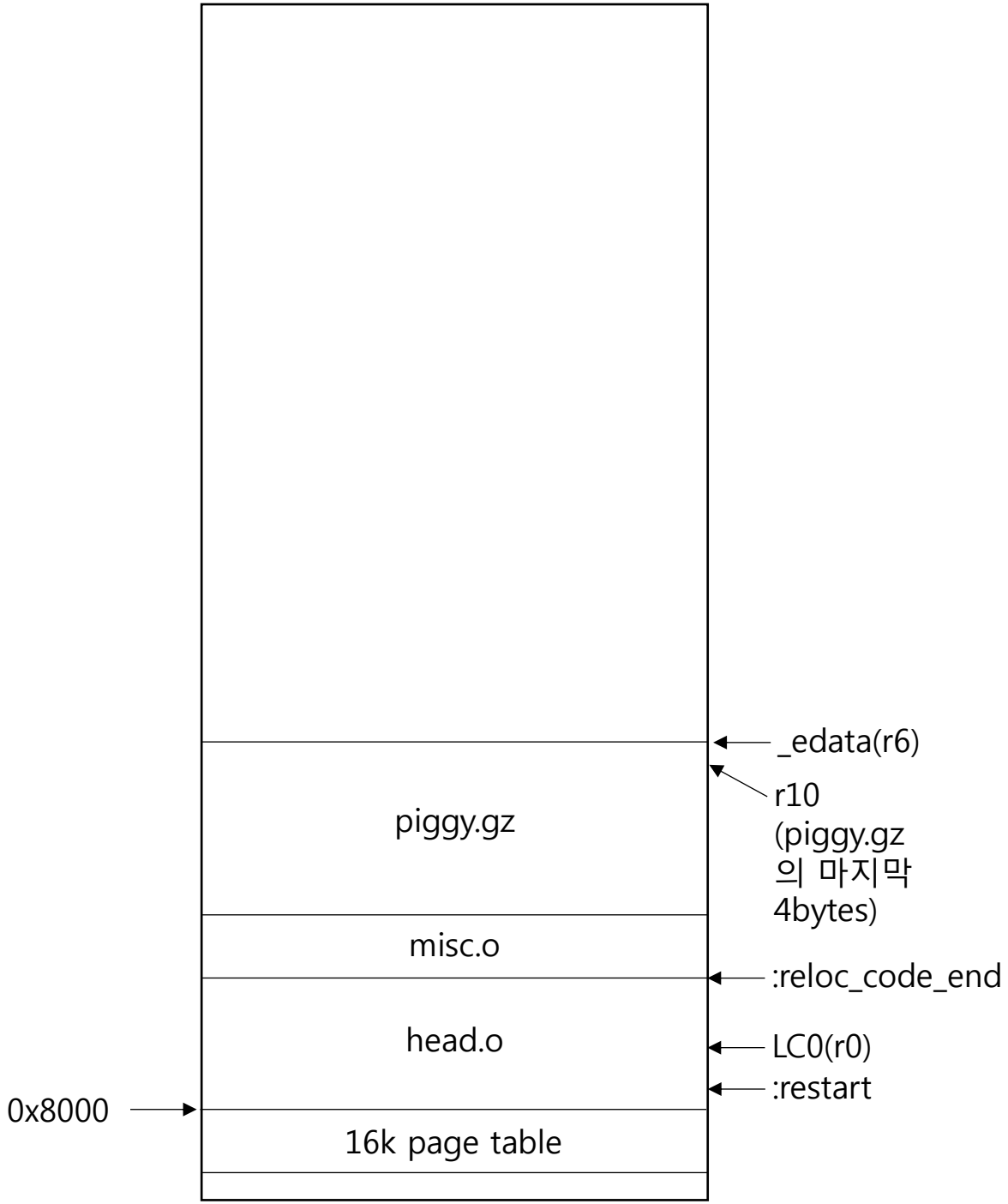
압축하기 전의 원본 커널 Image의 크기가 저장된 위치(r10)에 offset 추가

참고 : gzip으로 압축할 경우 압축 하기 전의 데이터 크기가 마지막 4bytes에 little-endian 방식으로 저장됨
https://tools.ietf.org/html/rfc1952



3

```
ldrb    r9, [r10, #0]
ldrb    lr, [r10, #1]
orr     r9, r9, lr, lsl #8
ldrb    lr, [r10, #2]
ldrb    r10, [r10, #3]
orr     r9, r9, lr, lsl #16
orr     r9, r9, r10, lsl #24
```

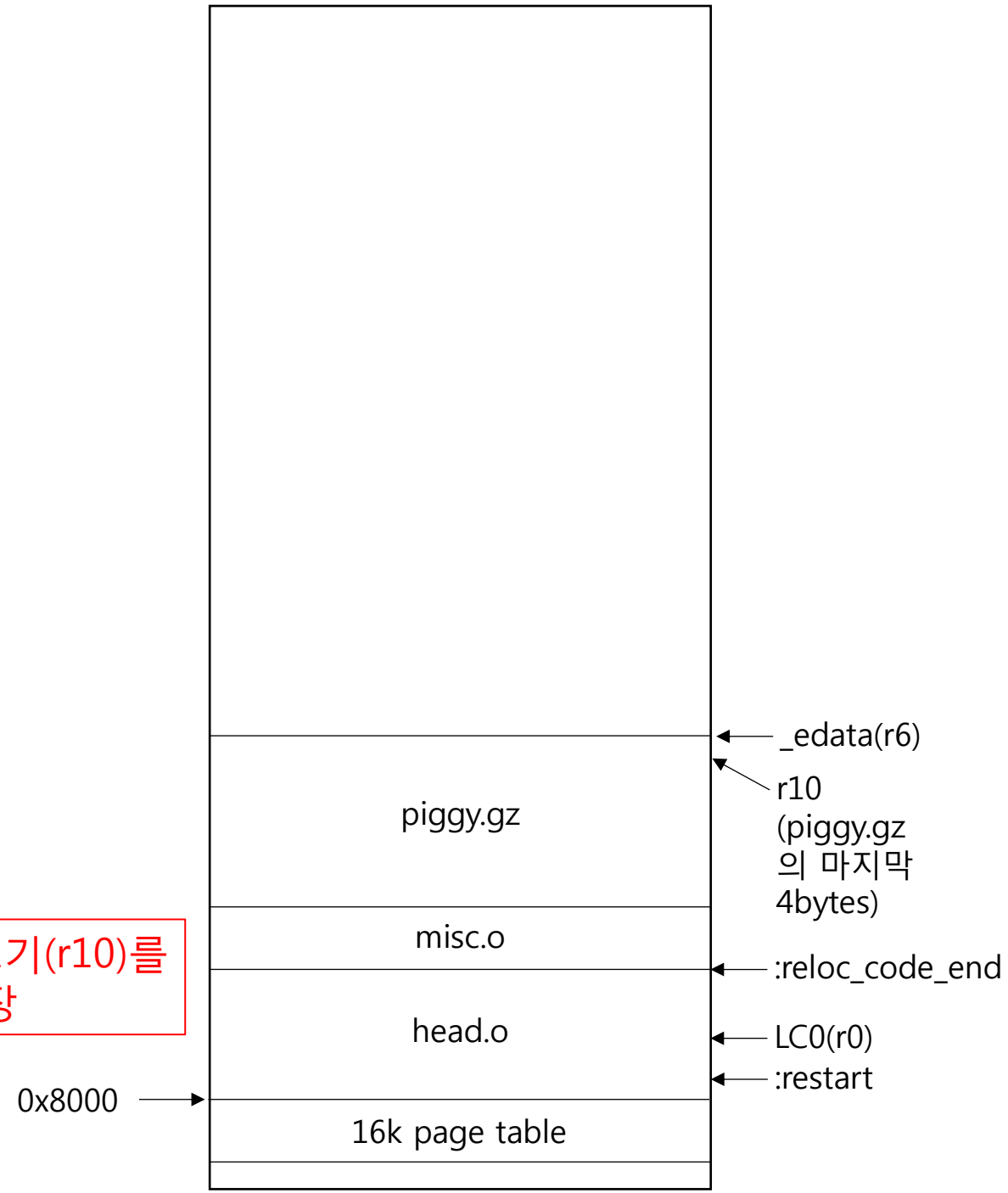


3

```
ldrb    r9, [r10, #0]
ldrb    lr, [r10, #1]
orr     r9, r9, lr, lsl #8
ldrb    lr, [r10, #2]
ldrb    r10, [r10, #3]
orr     r9, r9, lr, lsl #16
orr     r9, r9, r10, lsl #24
```

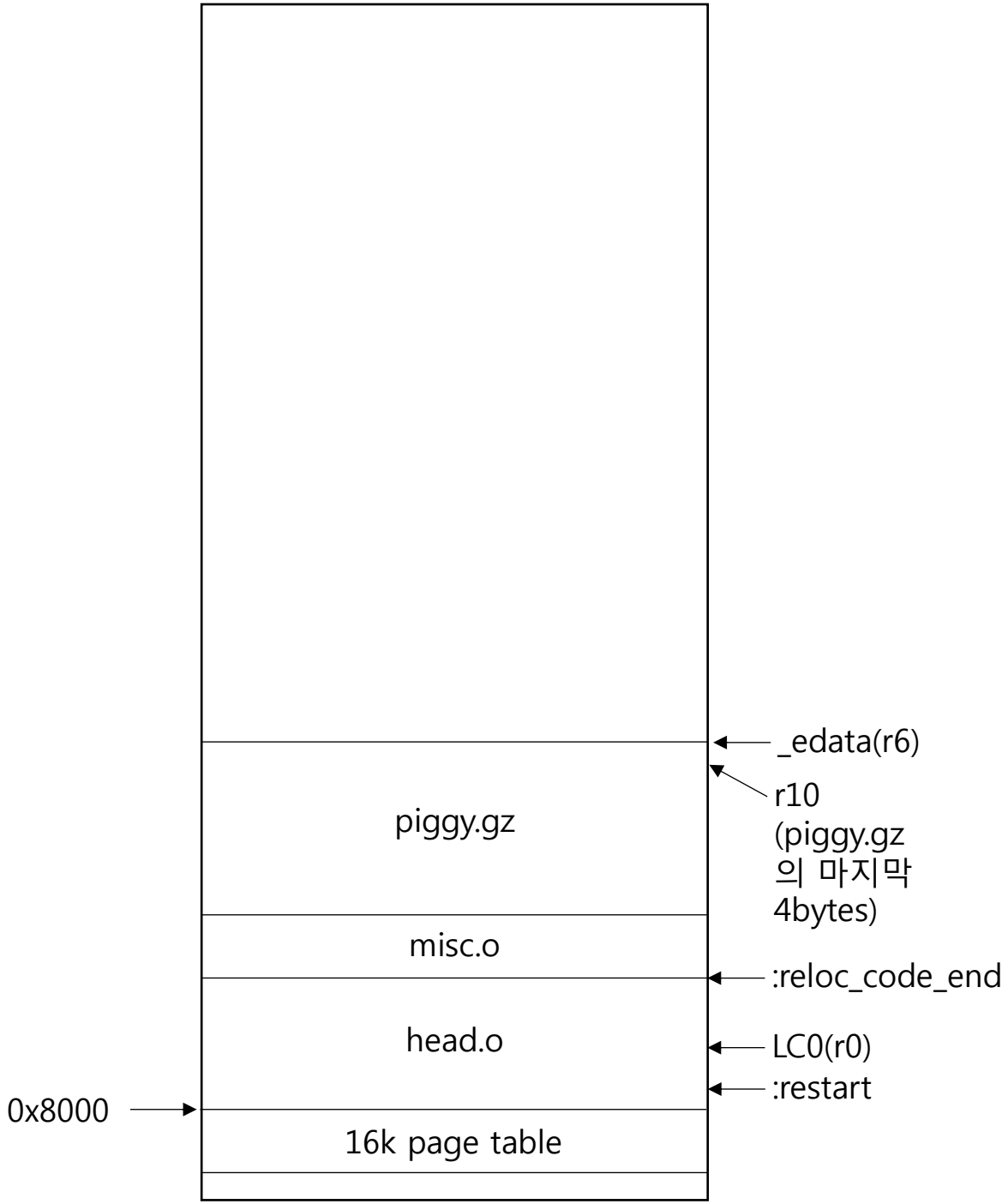
Little-endian으로 표현되어 있는 piggy.gz의 압축 하기 이전 크기(r10)를 이후 계산에 사용하기 위해 big-endian으로 변환해서 r9에 저장

r9 = decompressed kernel size



4

```
add     sp, sp, r0
add     r10, sp, #0x10000
```

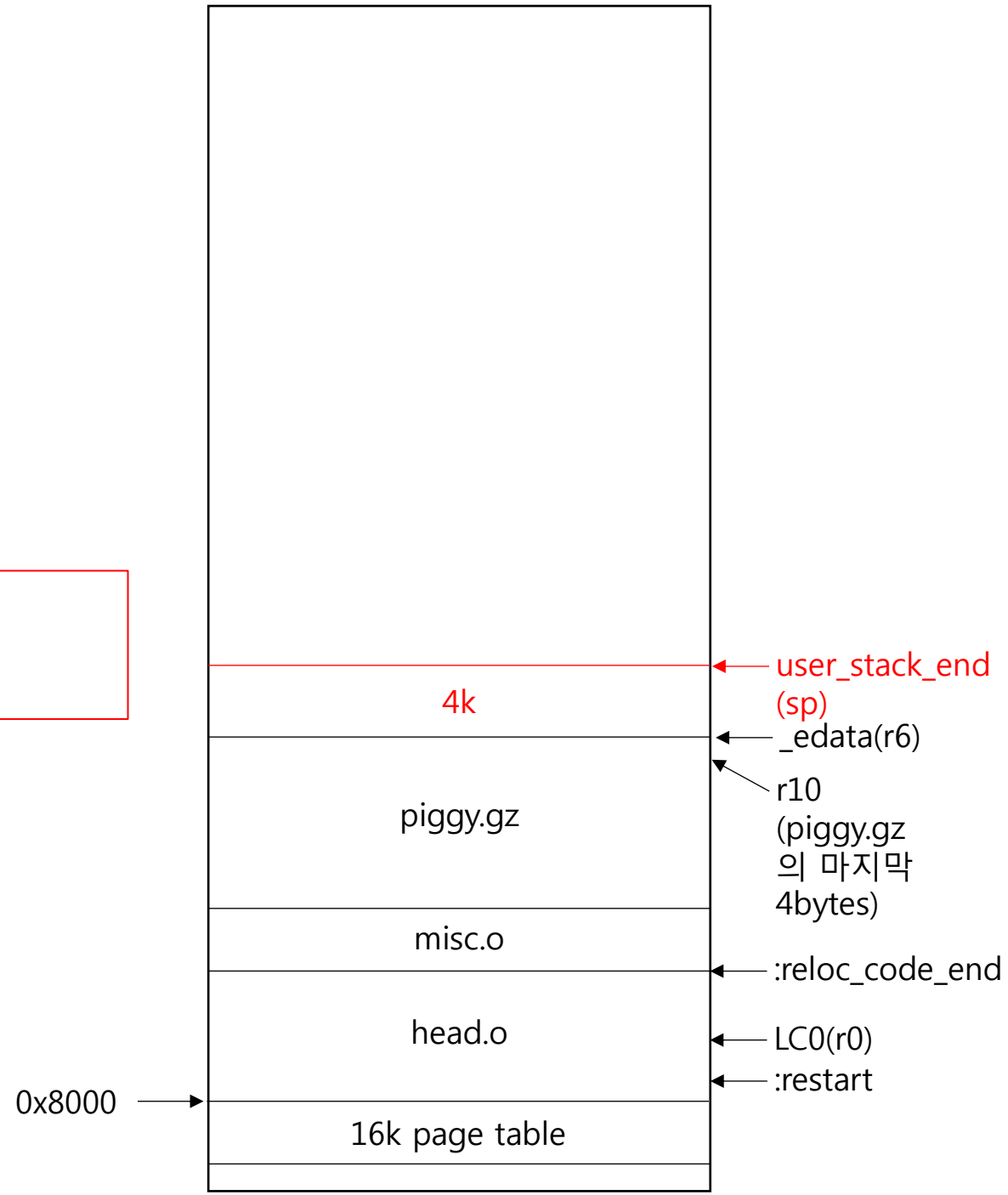


r9 = decompressed kernel size

4

```
add     sp, sp, r0
add     r10, sp, #0x10000
```

sp(user_stack_end)에 offset 추가
vmlinux_lds.S를 참고하면 stack 영역은 piggydata 이후에 배치 됨
user_stack(4k)은 head.S 맨 아래 stack section에 있음

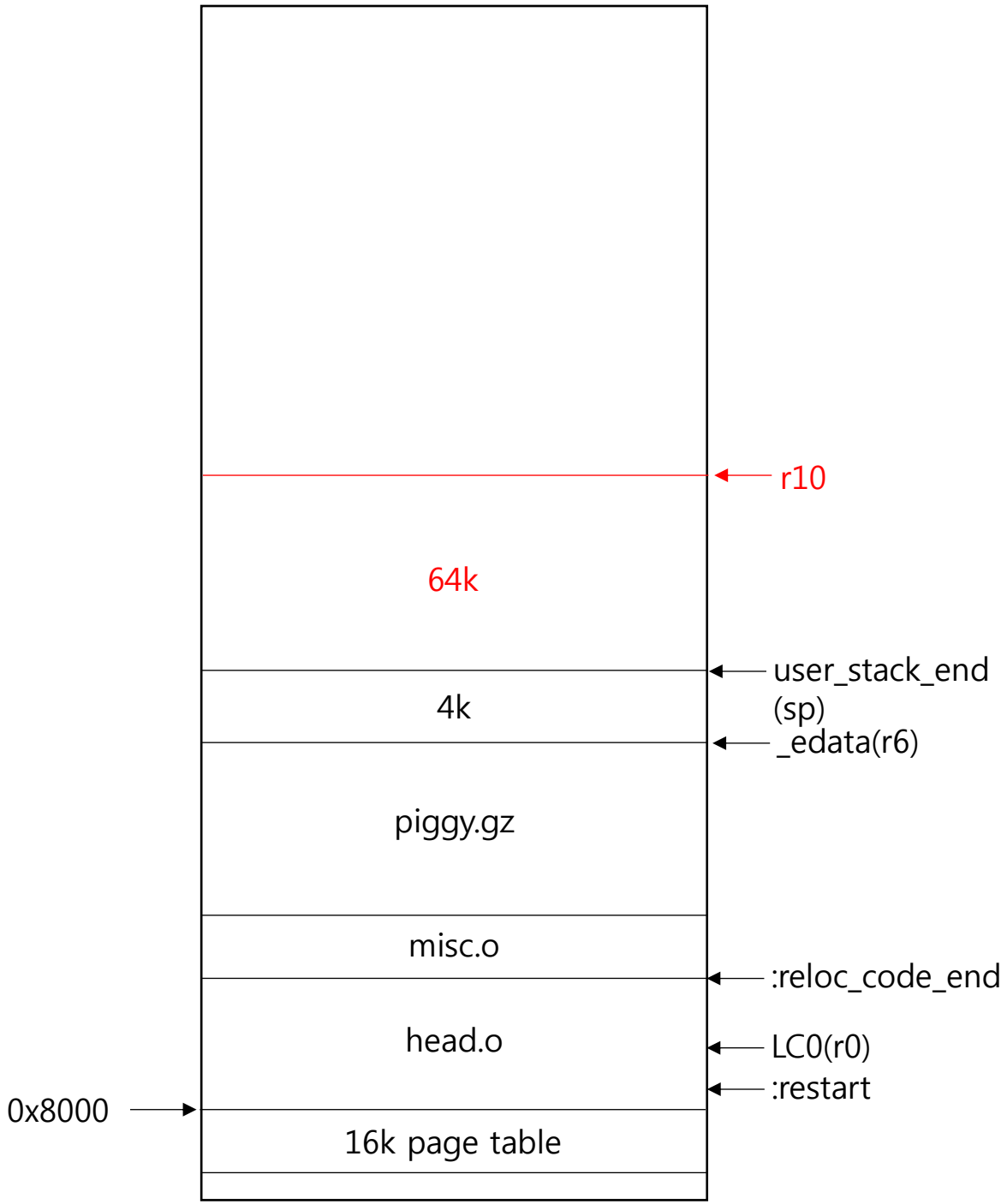


r9 = decompressed kernel size

4

```
add     sp, sp, r0
add     r10, sp, #0x10000
```

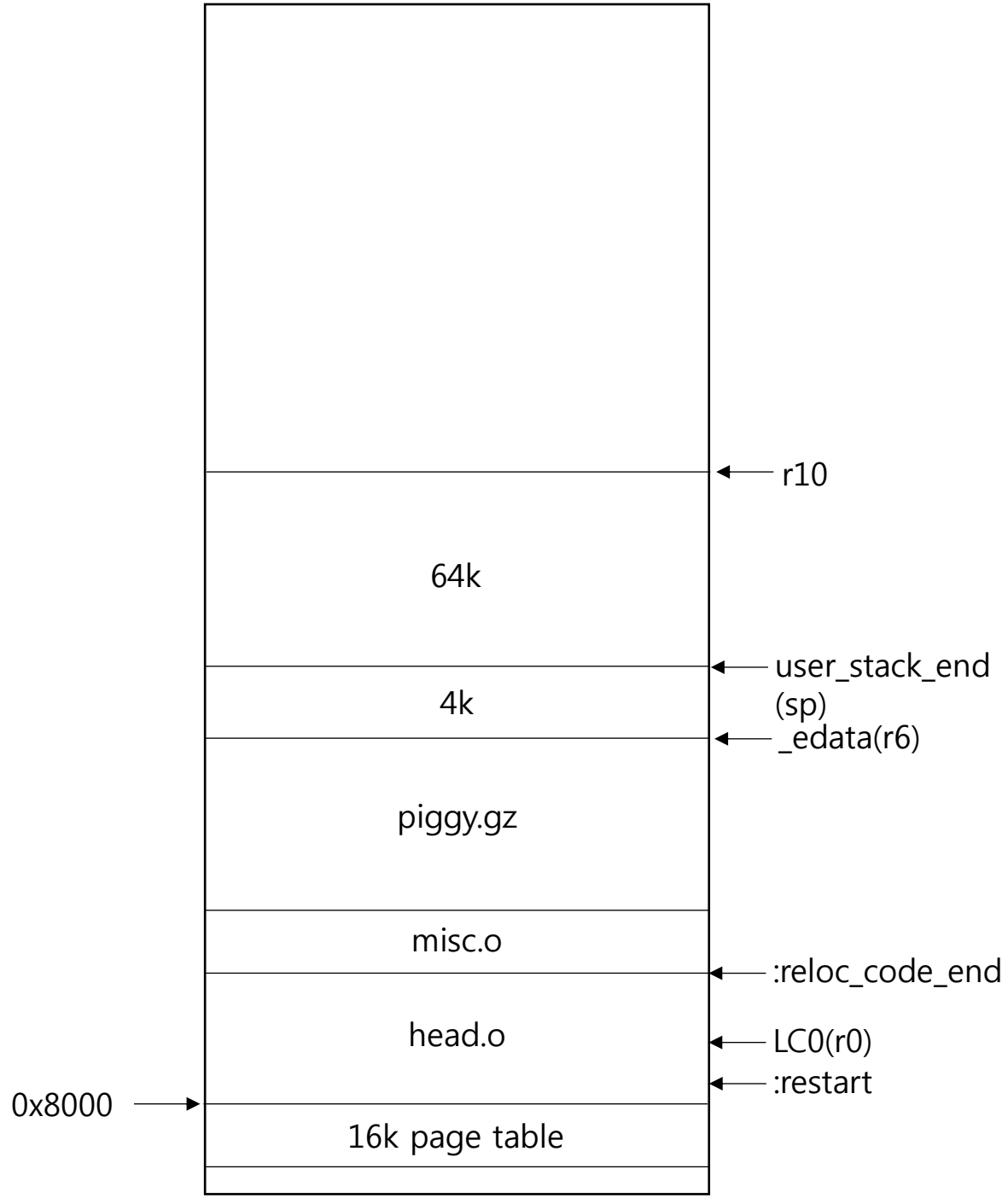
sp 이후 64k를 heap을 위한 공간으로 확보



r9 = decompressed kernel size

5

```
add     r10, r10, #16384
cmp     r4, r10
bhs     wont_overwrite
add     r10, r4, r9
adr     r9, wont_overwrite
cmp     r10, r9
bls     wont_overwrite
```

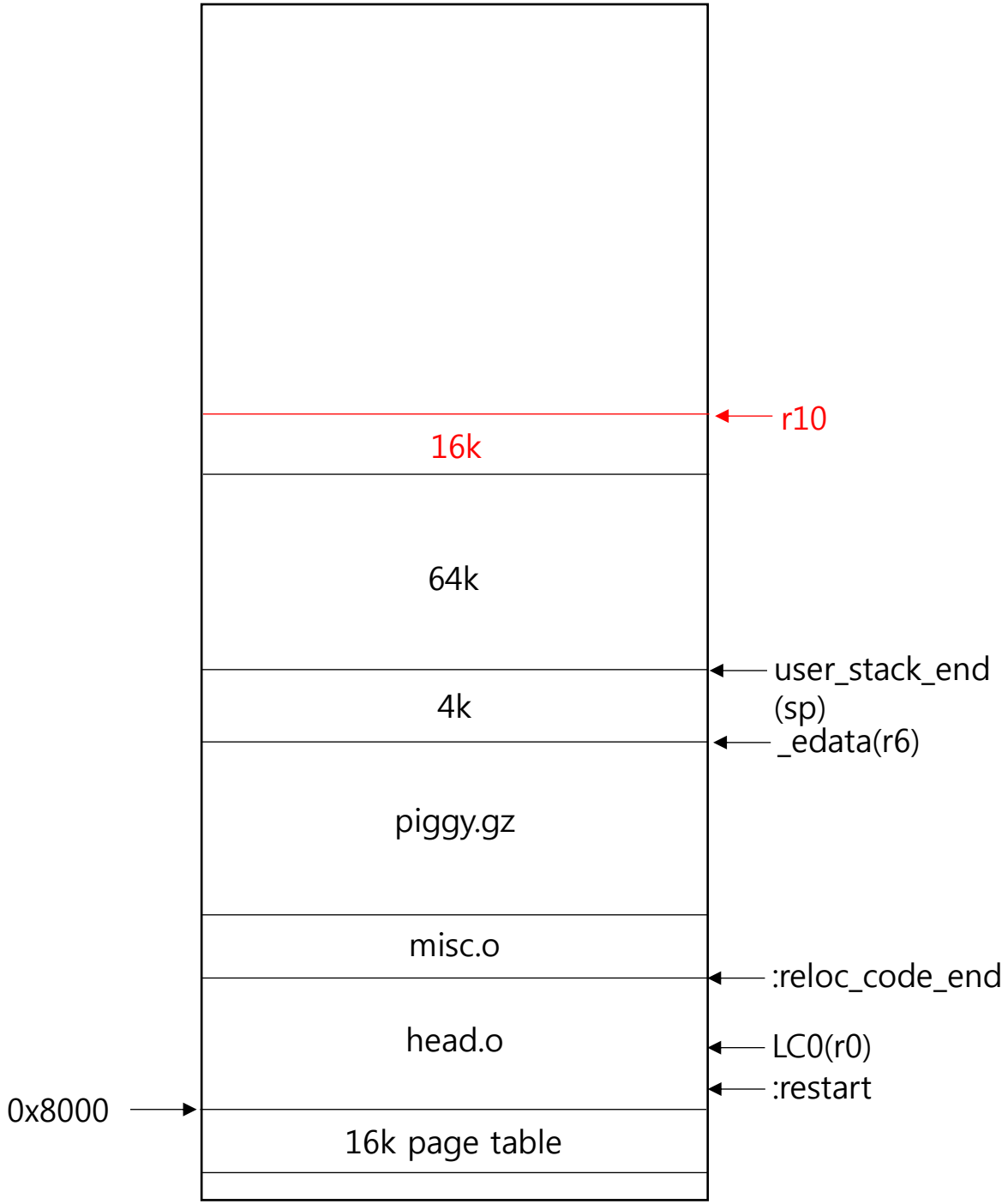


r9 = decompressed kernel size

5

```
add     r10, r10, #16384
cmp     r4, r10
bhs     wont_overwrite
add     r10, r4, r9
adr     r9, wont_overwrite
cmp     r10, r9
bls     wont_overwrite
```

r10 위치 위에 추가로 16k 공간 확보



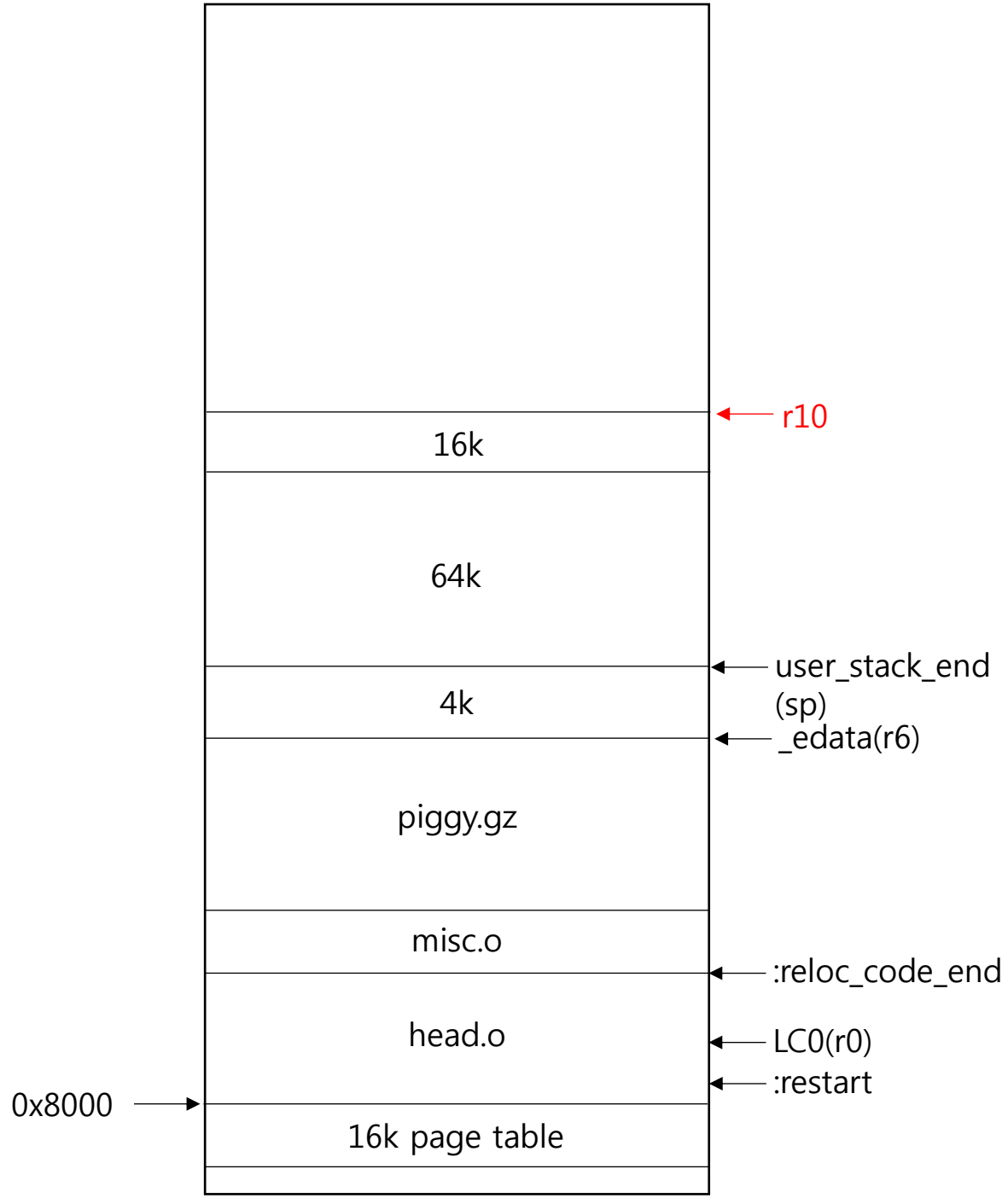
r9 = decompressed kernel size

5

```
add     r10, r10, #16384
cmp     r4, r10
bhs     wont_overwrite
add     r10, r4, r9
adr     r9, wont_overwrite
cmp     r10, r9
bls     wont_overwrite
```

압축을 풀 위치(r4(0x8000))와 r10 비교
r4가 r10보다 크거나 같으면(hs) wont_overwrite로 branch
즉, 현재 메모리에 있는 piggy+stack+heap 영역 끝보다
압축 풀 위치가 더 높은 곳에 있으면 압축 해제하면서 겹치지 않음
(wont_overwrite)
하지만, r4가 더 낮은 주소에 있으므로 압축 풀면서 현재 메모리를
overwrite 할 가능성이 있음

r9 = decompressed kernel size

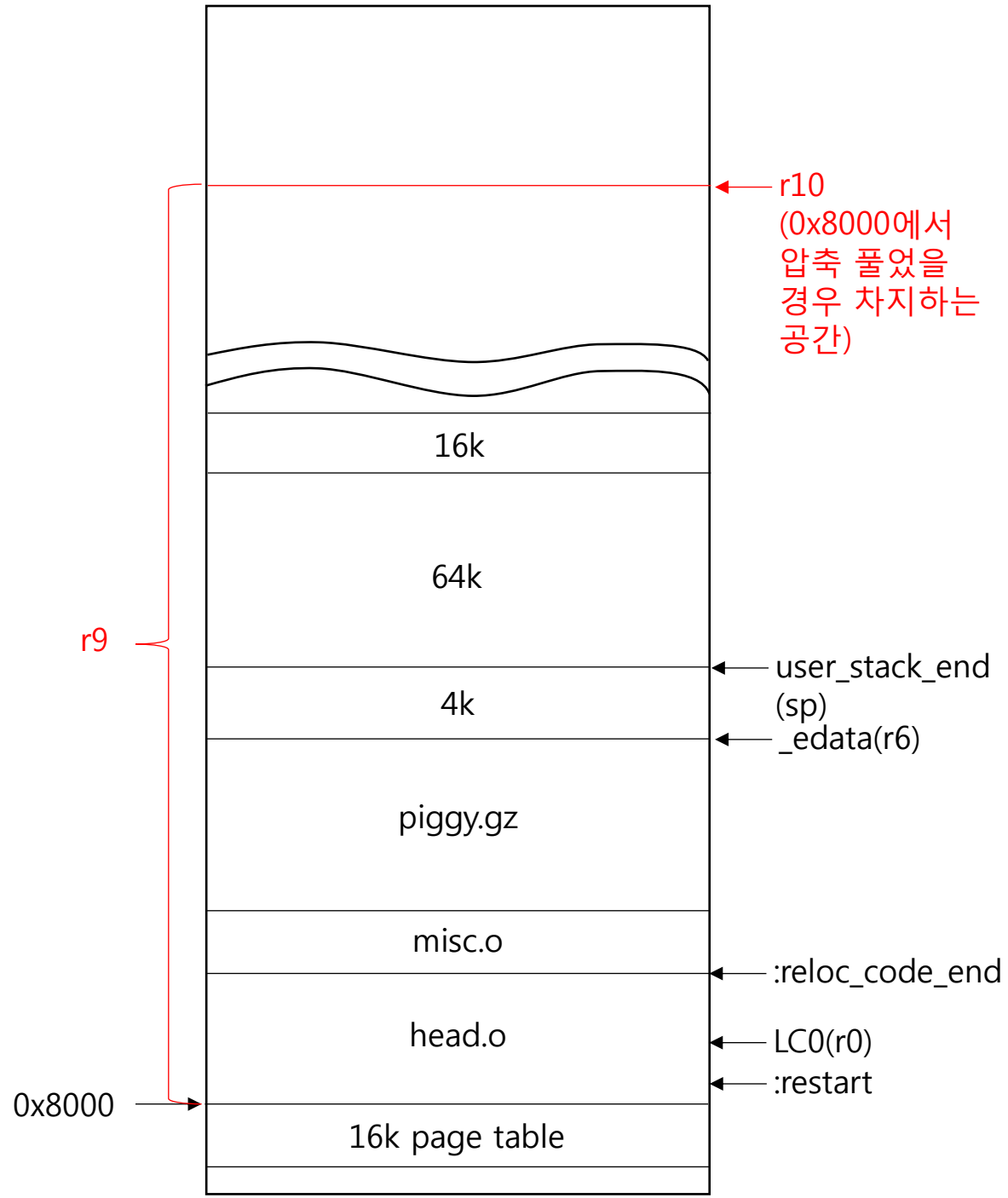


5

```
add     r10, r10, #16384
cmp     r4, r10
bhs     wont_overwrite
add     r10, r4, r9
adr     r9, wont_overwrite
cmp     r10, r9
bls     wont_overwrite
```

압축을 풀 위치(r4(0x8000))에 압축 해제한 커널 크기를 추가해서 r10에 저장
즉, 압축 해제 했을 때의 메모리 주소 upper boundary 계산

r9 = decompressed kernel size

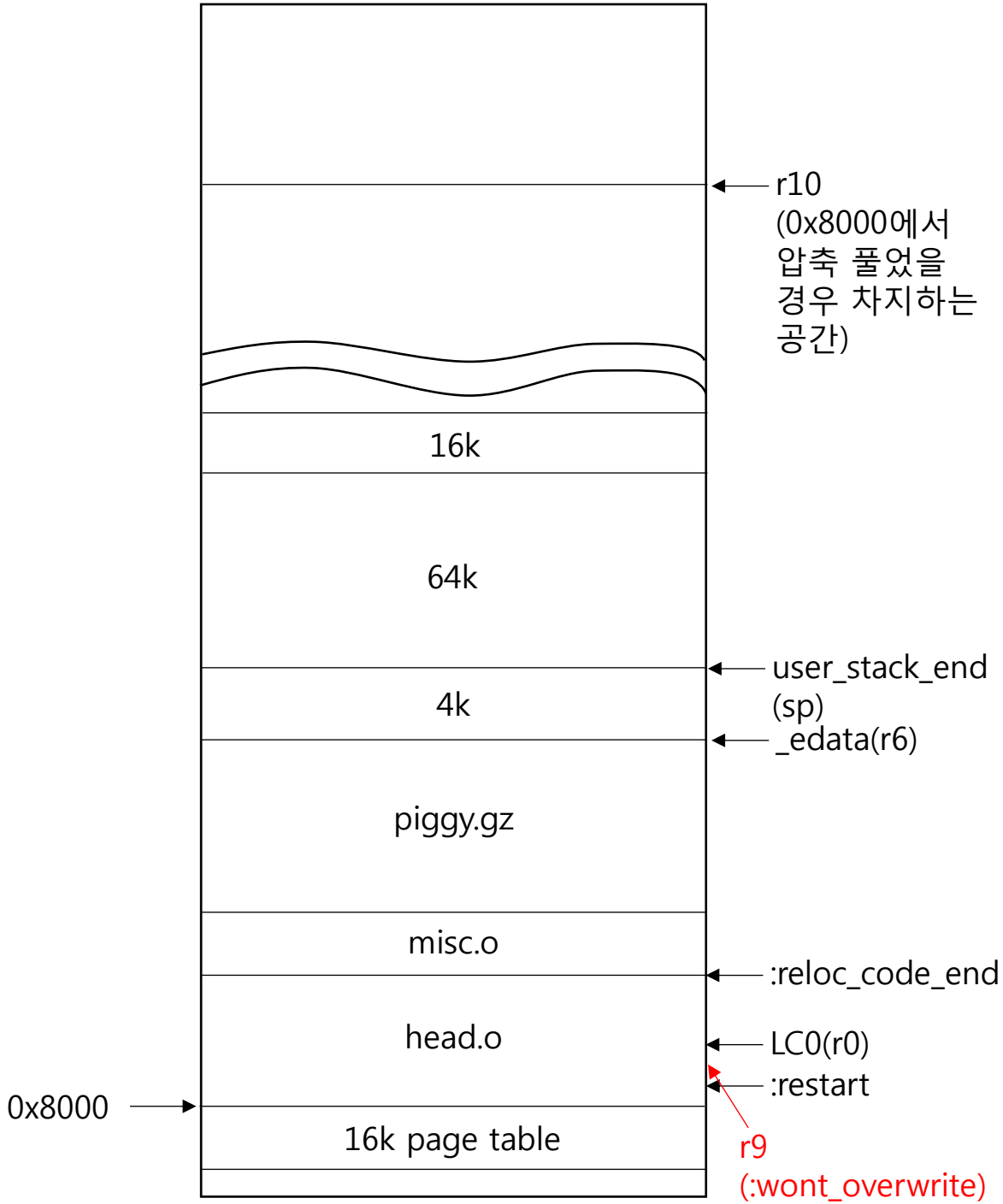


5

```
add     r10, r10, #16384
cmp     r4, r10
bhs     wont_overwrite
add     r10, r4, r9
adr     r9, wont_overwrite
cmp     r10, r9
bls     wont_overwrite
```

wont_overwrite label의 pc 상대 주소를 r9에 저장
head.o의 시작부터가 아닌 wont_overwrite의 위치로 이후에 비교하는 이유는 현재까지 사용한 코드는 겹쳐도 상관없기 때문

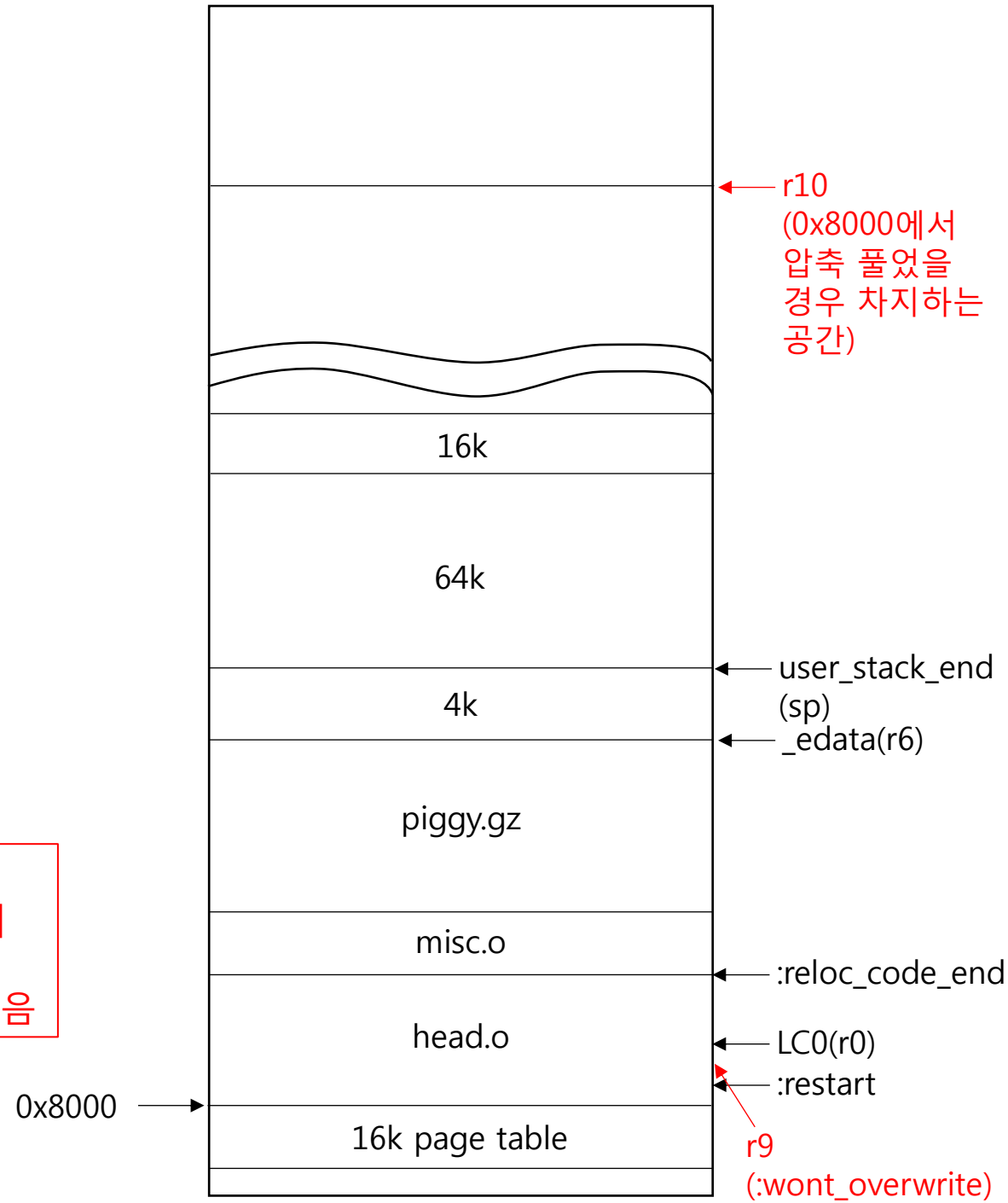
r9 = decompressed kernel size



5

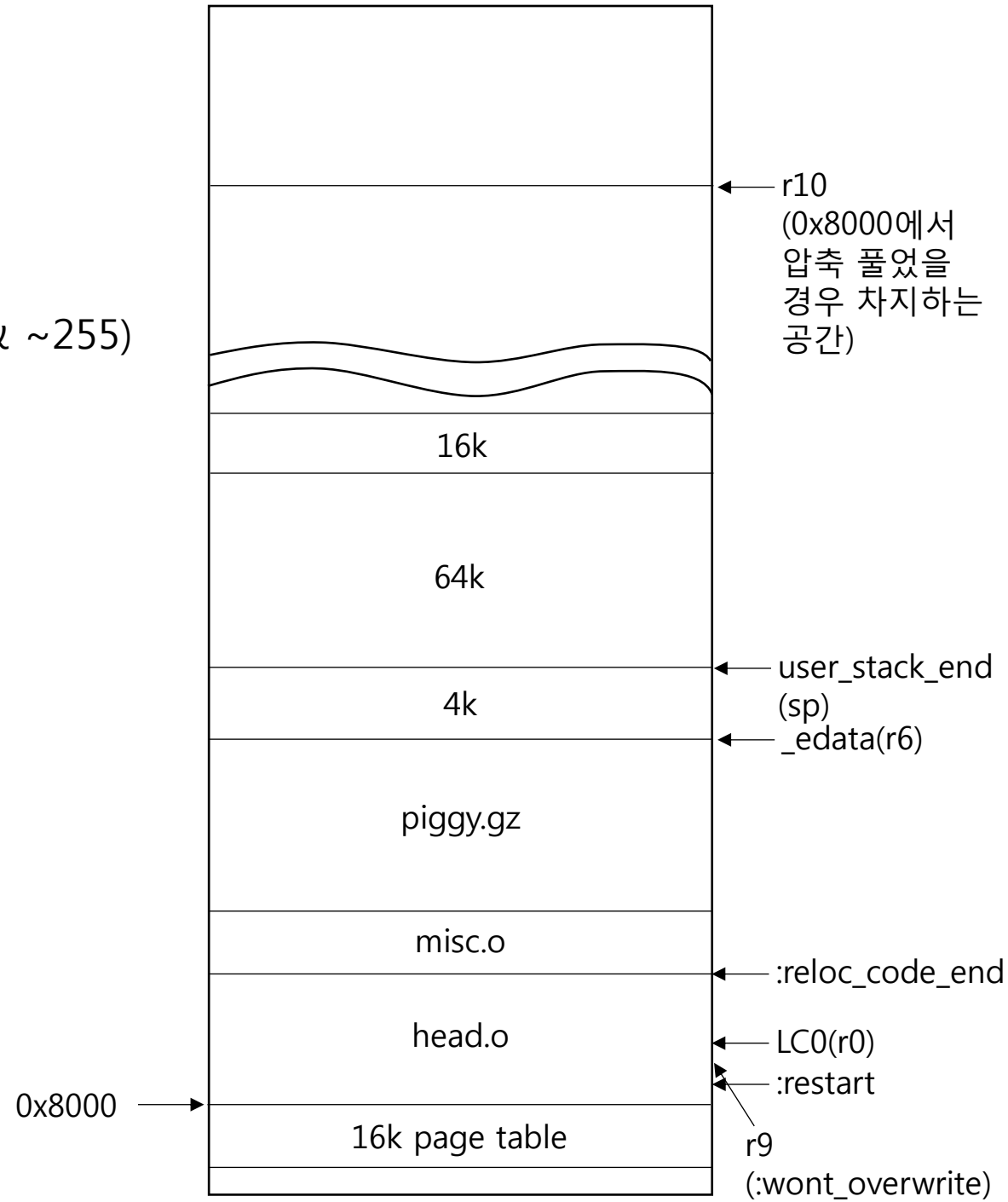
```
add     r10, r10, #16384
cmp     r4, r10
bhs     wont_overwrite
add     r10, r4, r9
adr     r9, wont_overwrite
cmp     r10, r9
bls     wont_overwrite
```

압축 해제한 메모리 상한(r10)과 겹쳐도 상관없는 메모리(r9) 비교
즉, 압축을 해제하더라도 현재 실행중인 코드가 메모리 위쪽에 있어서 겹치지 않는지 확인
하지만, r10이 위에 있으므로 overwrite 발생하기에 branch 하지 않음



6

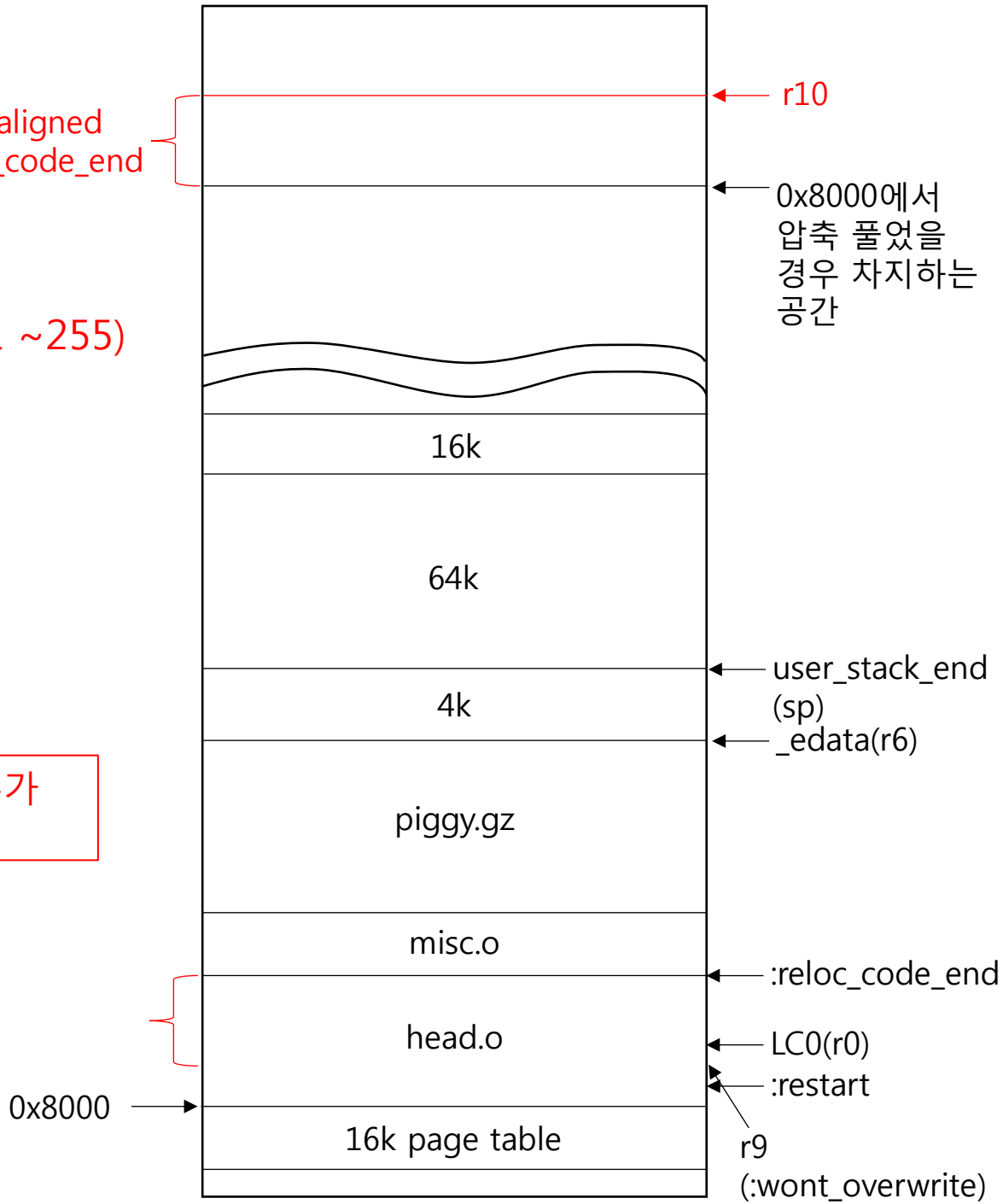
```
add     r10, r10, #((reloc_code_end - restart + 256) & ~255)
bic     r10, r10, #255

adr     r5, restart
bic     r5, r5, #31
```



6

```
add     r10, r10, #((reloc_code_end - restart + 256) & ~255)
bic     r10, r10, #255

adr     r5, restart
bic     r5, r5, #31
```

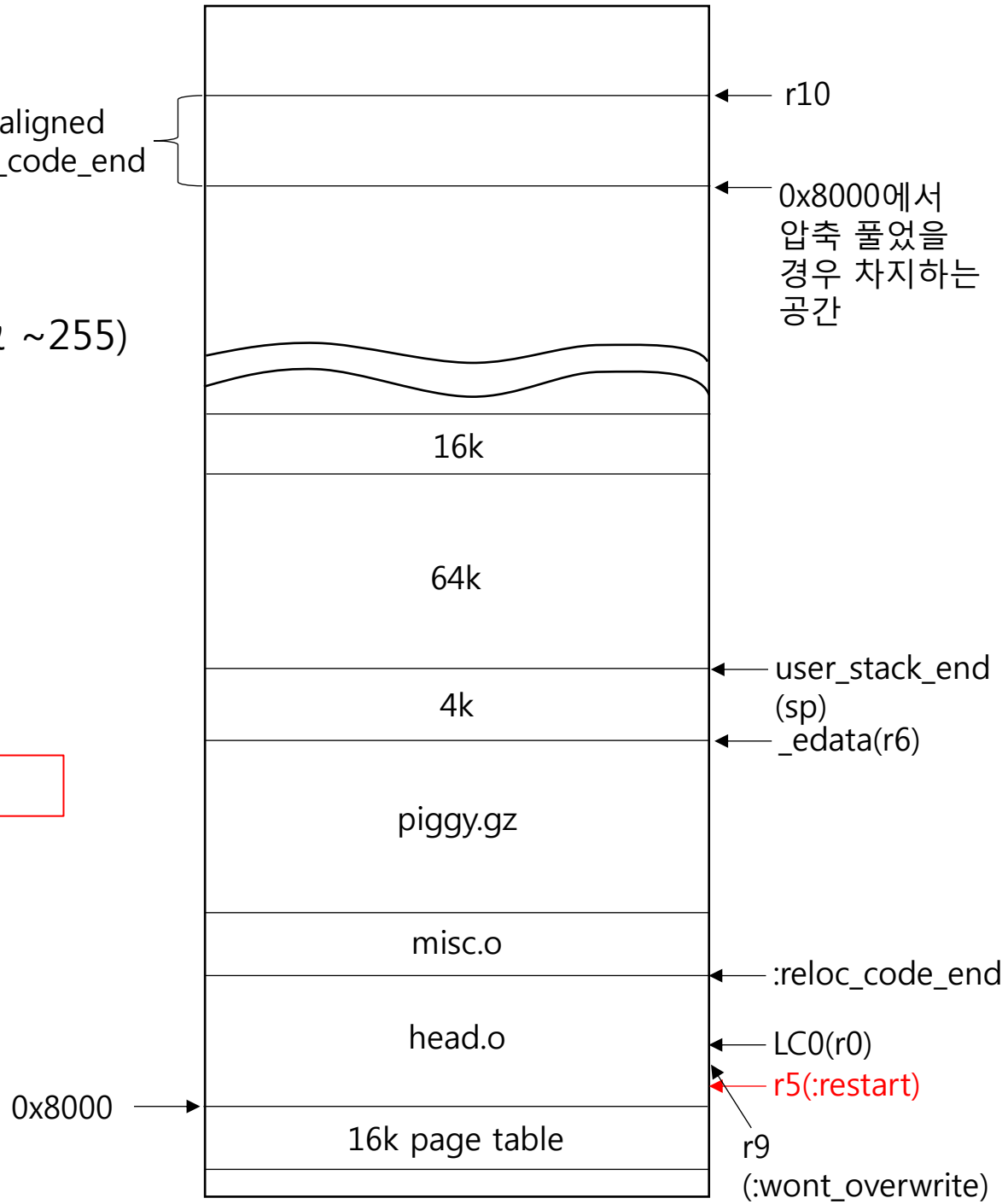
r10에 256 bytes align 시킨 restart~reloc_code_end 크기만큼 추가
이후, r10을 다시 256 bytes align



6

```
add     r10, r10, #((reloc_code_end - restart + 256) & ~255)
bic     r10, r10, #255

adr     r5, restart
bic     r5, r5, #31
```

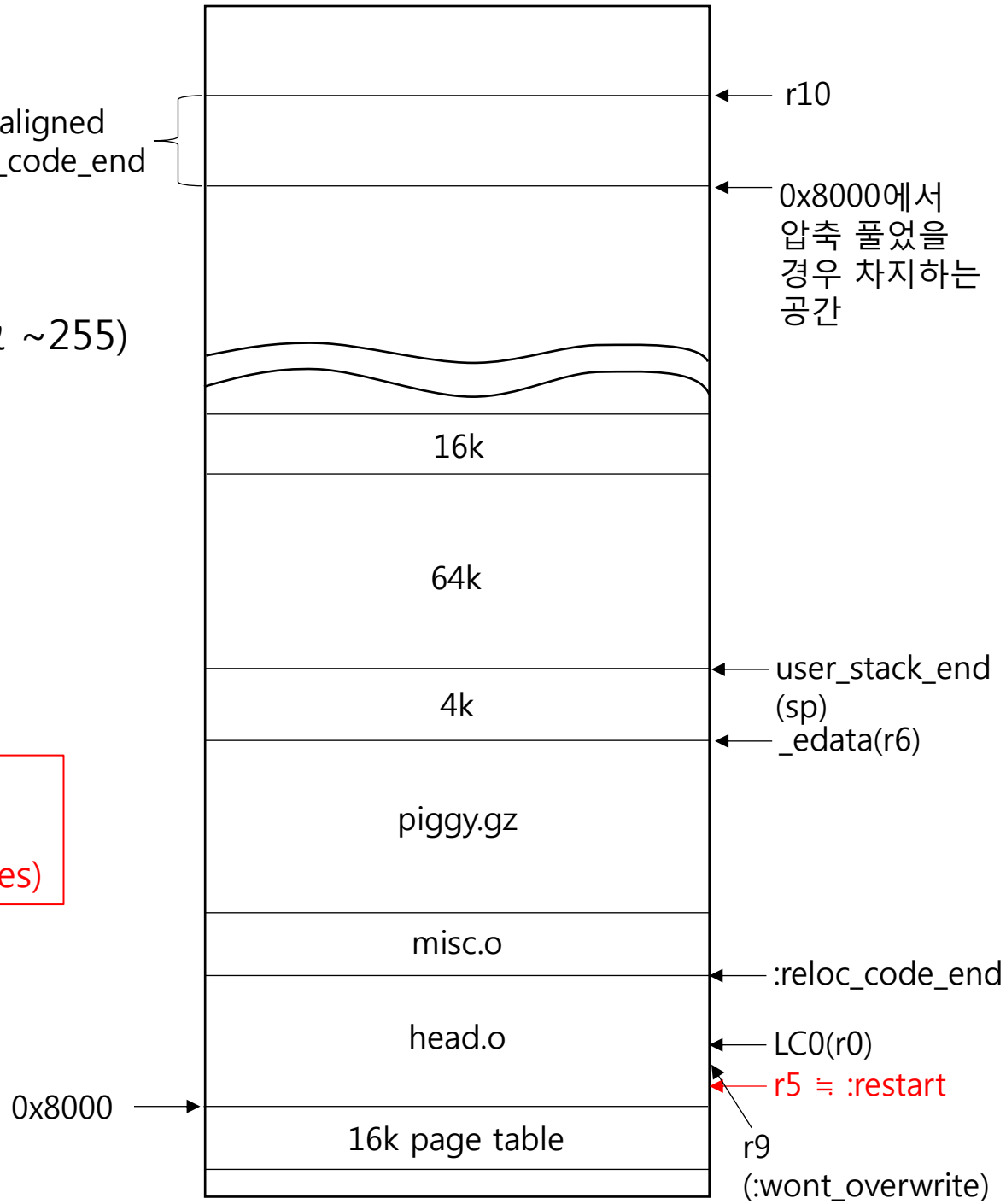
r5에 restart label의 pc 상대 주소 저장



6

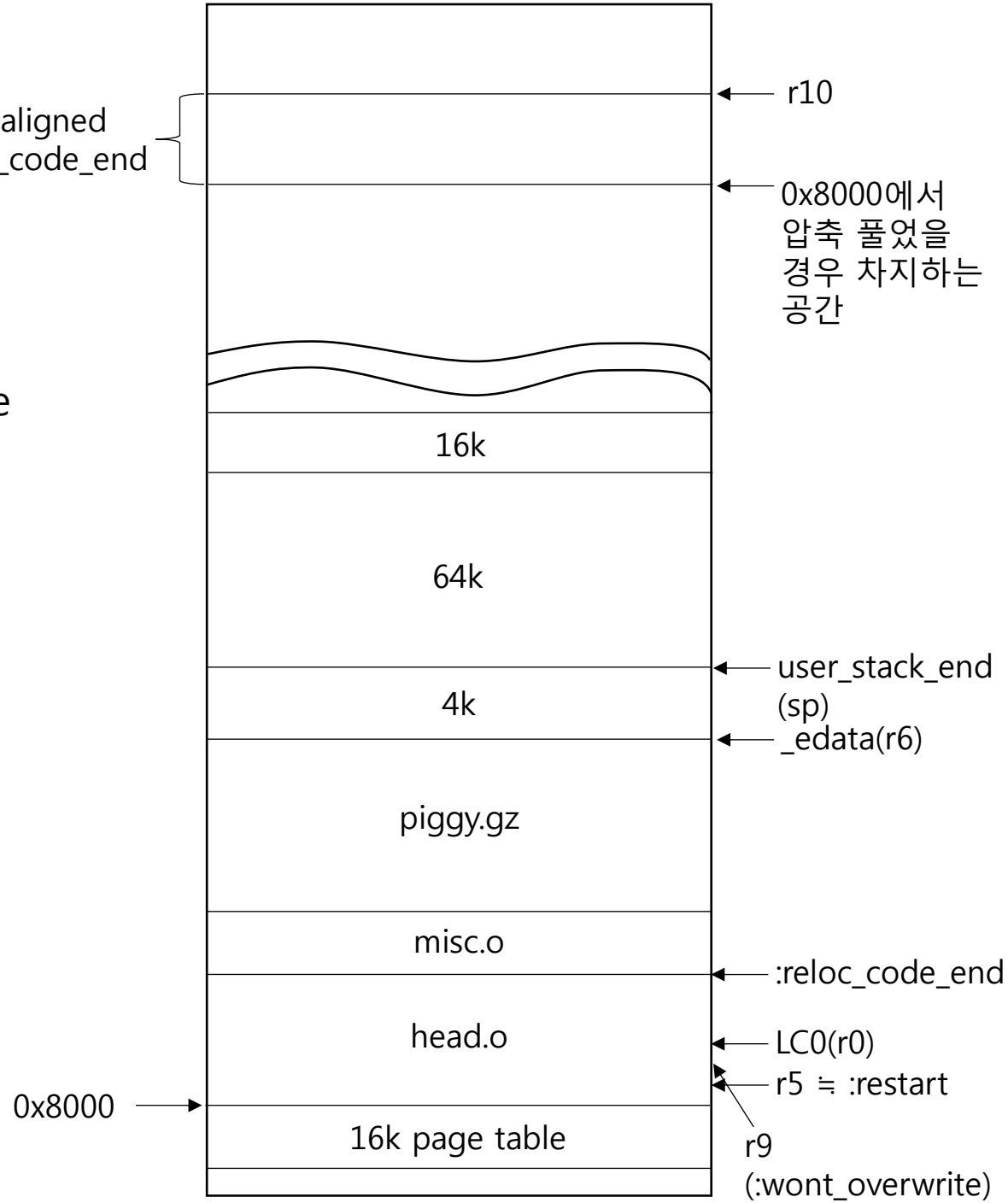
```
add     r10, r10, #((reloc_code_end - restart + 256) & ~255)
bic     r10, r10, #255

adr     r5, restart
bic     r5, r5, #31
```

r5의 하위 31비트 clear
즉, 32 bytes align
r5는 실제 restart label보다 조금 아래쪽일 수 있다. (최대 31 bytes)



7

```
sub     r9, r6, r5      @ size to copy
add     r9, r9, #31     @ rounded up to a multiple
bic     r9, r9, #31     @ ... of 32 bytes
add     r6, r9, r5
add     r9, r9, r10
```

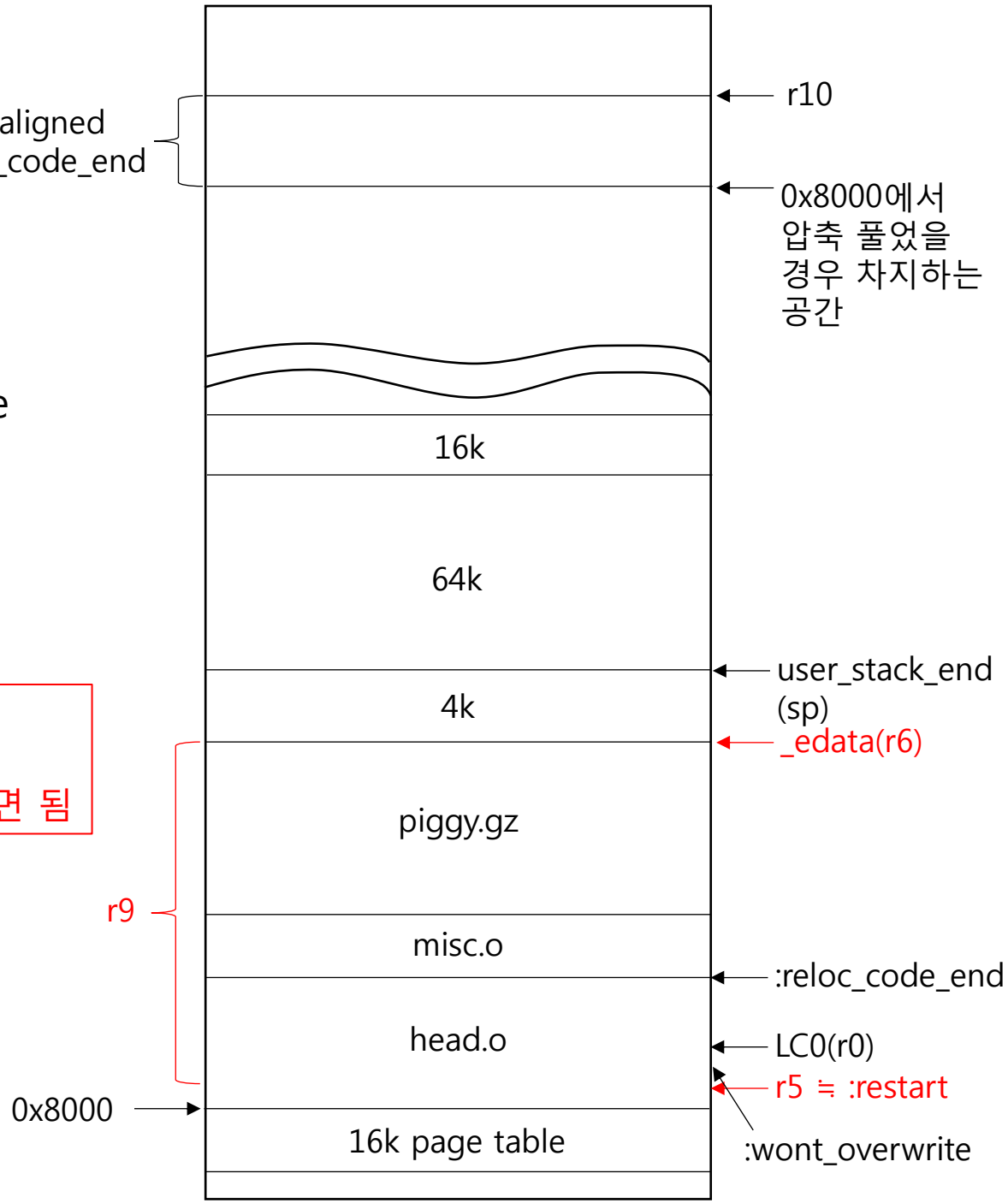


7

```
sub     r9, r6, r5      @ size to copy
add     r9, r9, #31     @ rounded up to a multiple
bic     r9, r9, #31     @ ... of 32 bytes
add     r6, r9, r5
add     r9, r9, r10
```

재배치할 크기 계산
restart label부터 _edata(piggydata+padding)까지 재 배치
그 위에 있는 4k, 64k 16k의 stack, heap은 재배치후 다시 확보 하면 됨

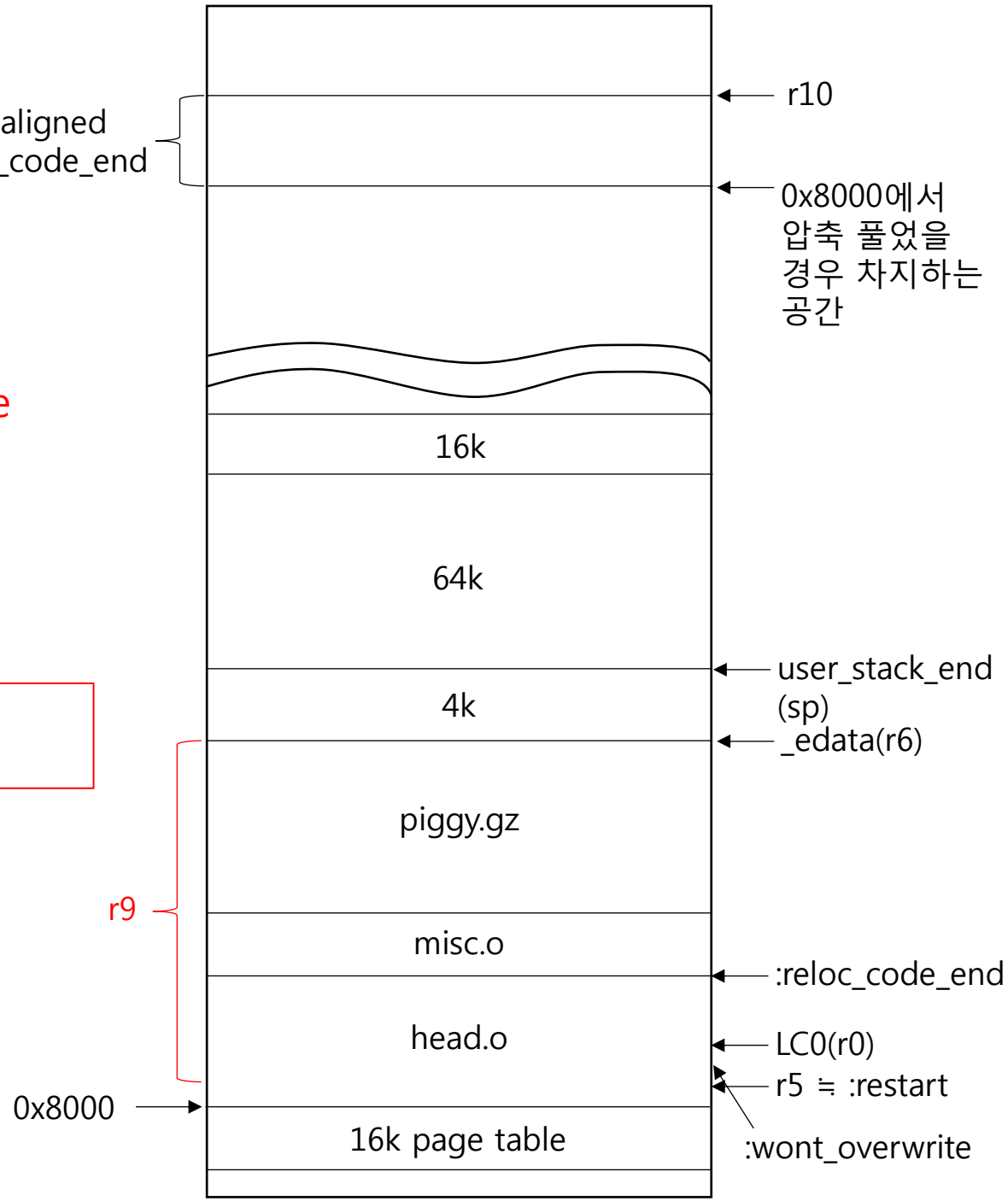
r9 = size to copy (재배치할 영역 크기)



7

```
sub     r9, r6, r5      @ size to copy
add     r9, r9, #31     @ rounded up to a multiple
bic     r9, r9, #31     @ ... Of 32 bytes
add     r6, r9, r5
add     r9, r9, r10
```

재배치할 크기(r9)를 32의 배수가 되도록 올림(rounded up)
조금 더(최대 31 bytes) copy 한다고 해서 문제 될 건 없음

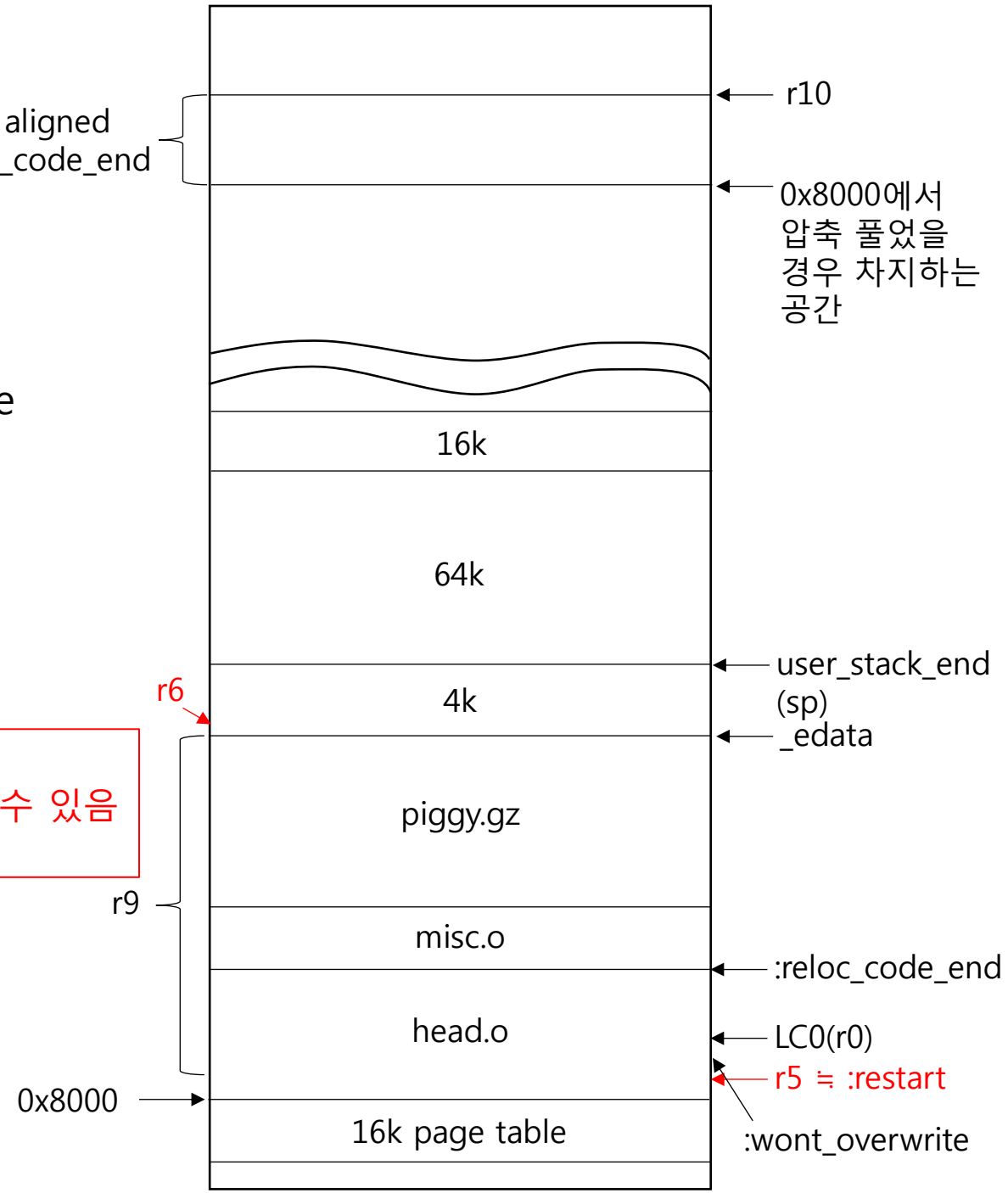


r9 = size to copy (재배치할 영역 크기)

7

```
sub     r9, r6, r5      @ size to copy
add     r9, r9, #31     @ rounded up to a multiple
bic     r9, r9, #31     @ ... Of 32 bytes
add     r6, r9, r5
add     r9, r9, r10
```

restart label부터 재배치할 영역 크기만큼 더한 값을 r6에 저장
r6은 앞에서 32 bytes 반올림 했기 때문에 _edata보다 약간 위가 될 수 있음
r6은 이후 asm memcopy 연산에서 src 주소로 사용



r9 = size to copy (재배치할 영역 크기)

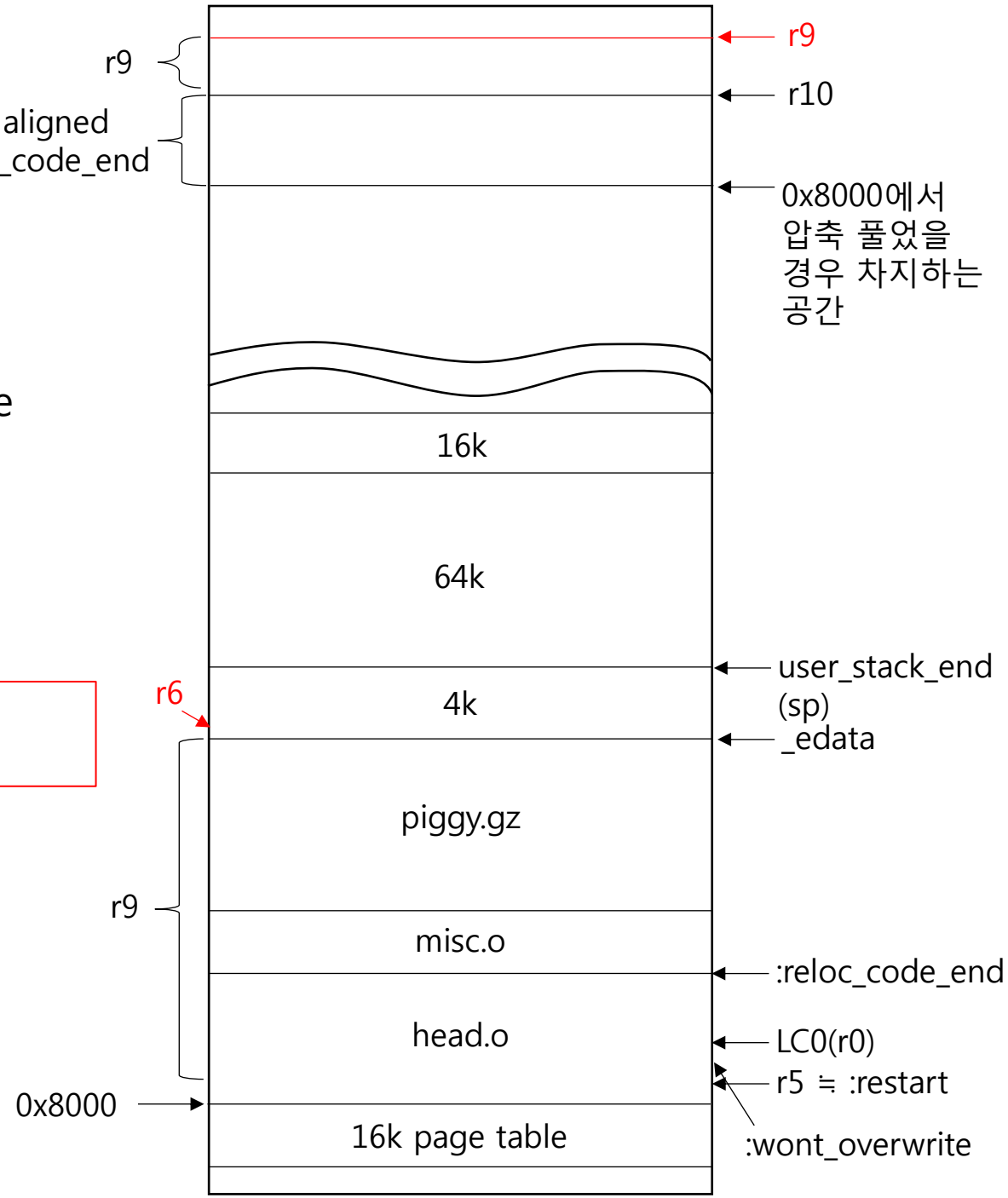
7

```
sub     r9, r6, r5      @ size to copy
add     r9, r9, #31     @ rounded up to a multiple
bic     r9, r9, #31     @ ... Of 32 bytes
add     r6, r9, r5
add     r9, r9, r10
```

r10에 재배치할 영역 크기(r9)만큼 더해서 r9에 저장
r9는 이후 asm memcopy 연산에서 dst 주소로 사용

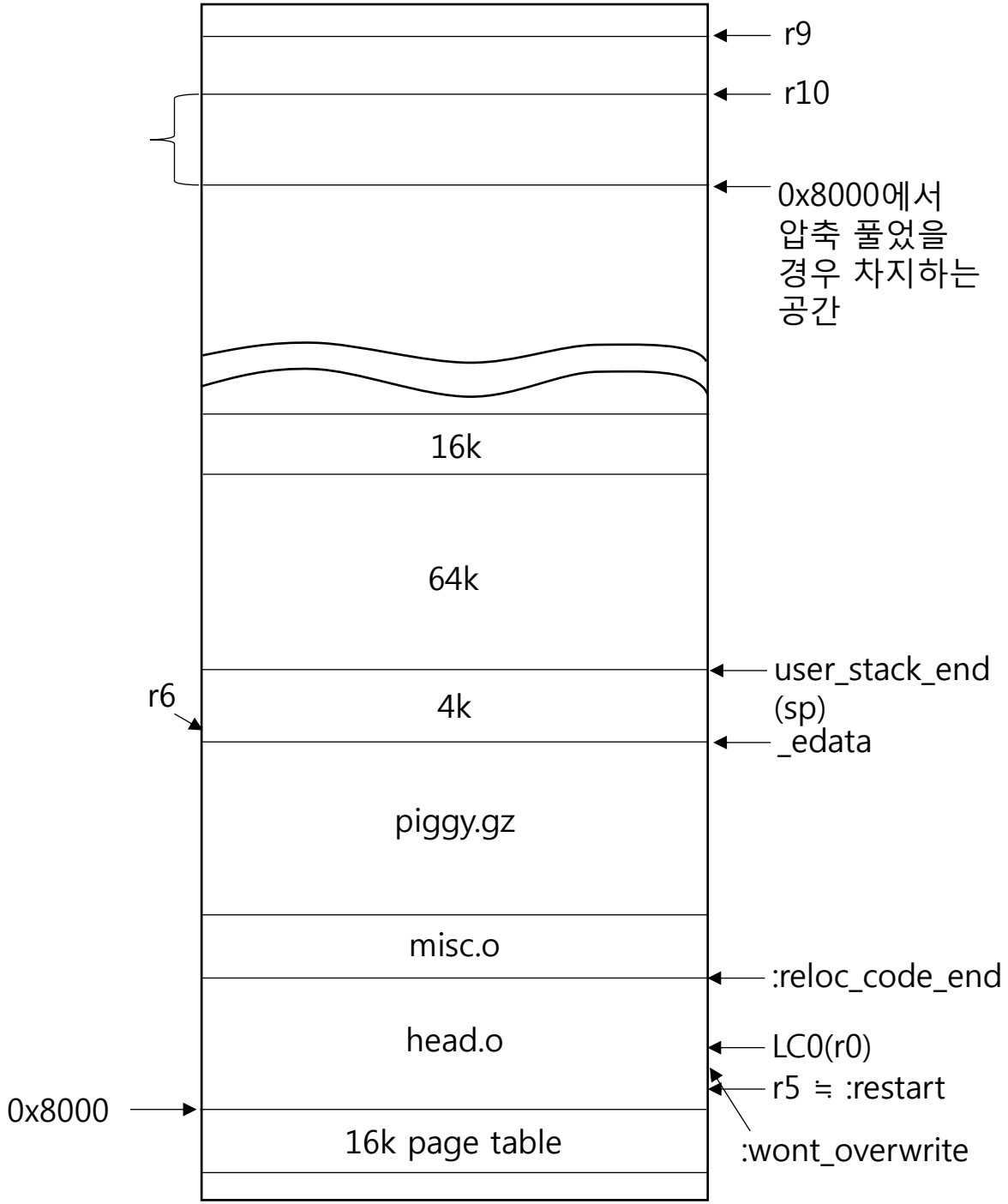
참고 : Memory layout에서 r9가 가리키는 재배치 공간(r9)과 r6가 가리키는 재배치 공간(r9)은 동일한 크기!

r9 = size to copy (재배치할 영역 크기)



8

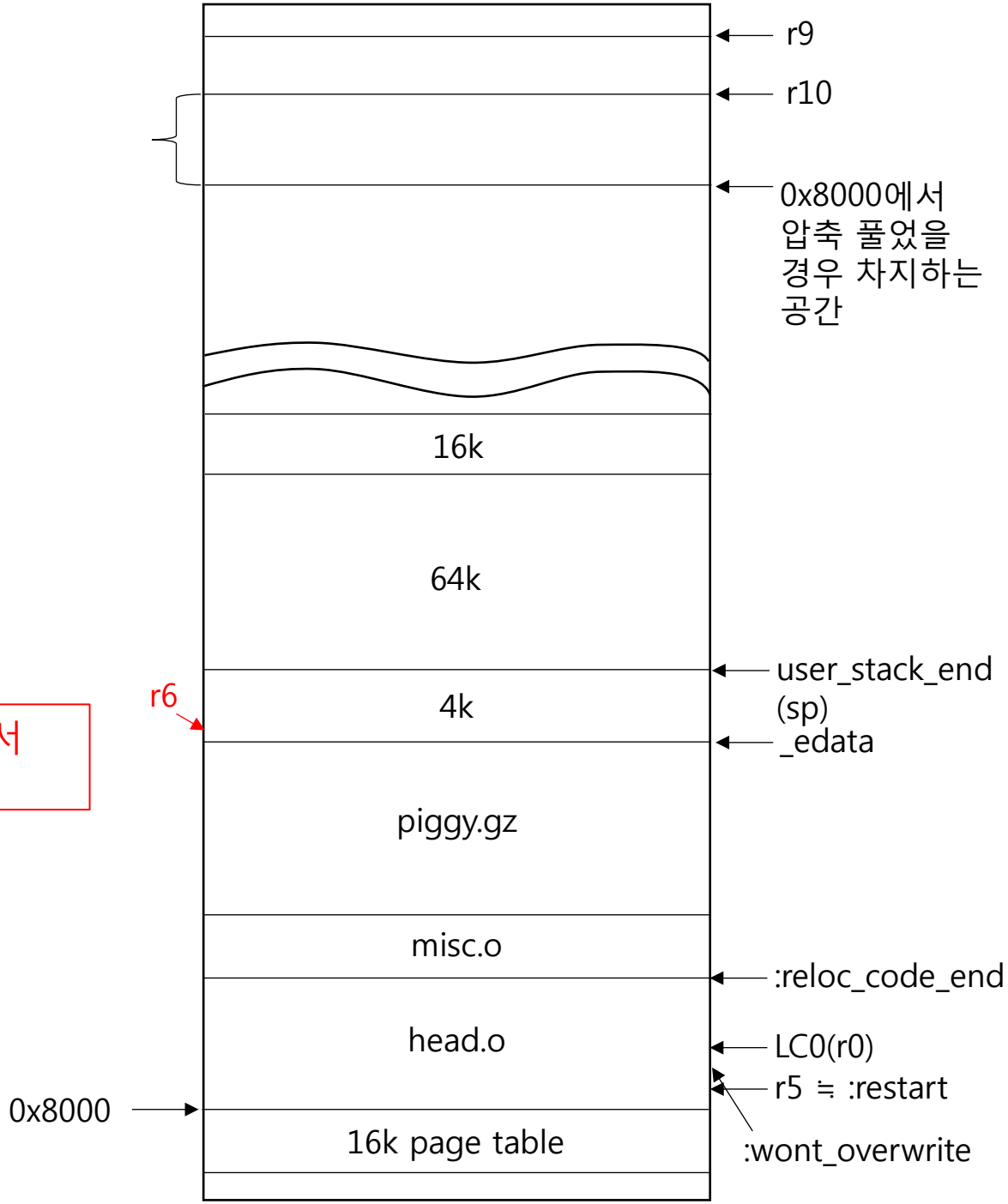
```
1:      ldmdb   r6!, {r0 - r3, r10 - r12, lr}
        cmp     r6, r5
        stmdb   r9!, {r0 - r3, r10 - r12, lr}
        bhi     1b
```



8

```
1:      ldmdb   r6!, {r0 - r3, r10 - r12, lr}
        cmp     r6, r5
        stmdb   r9!, {r0 - r3, r10 - r12, lr}
        bhi     1b
```

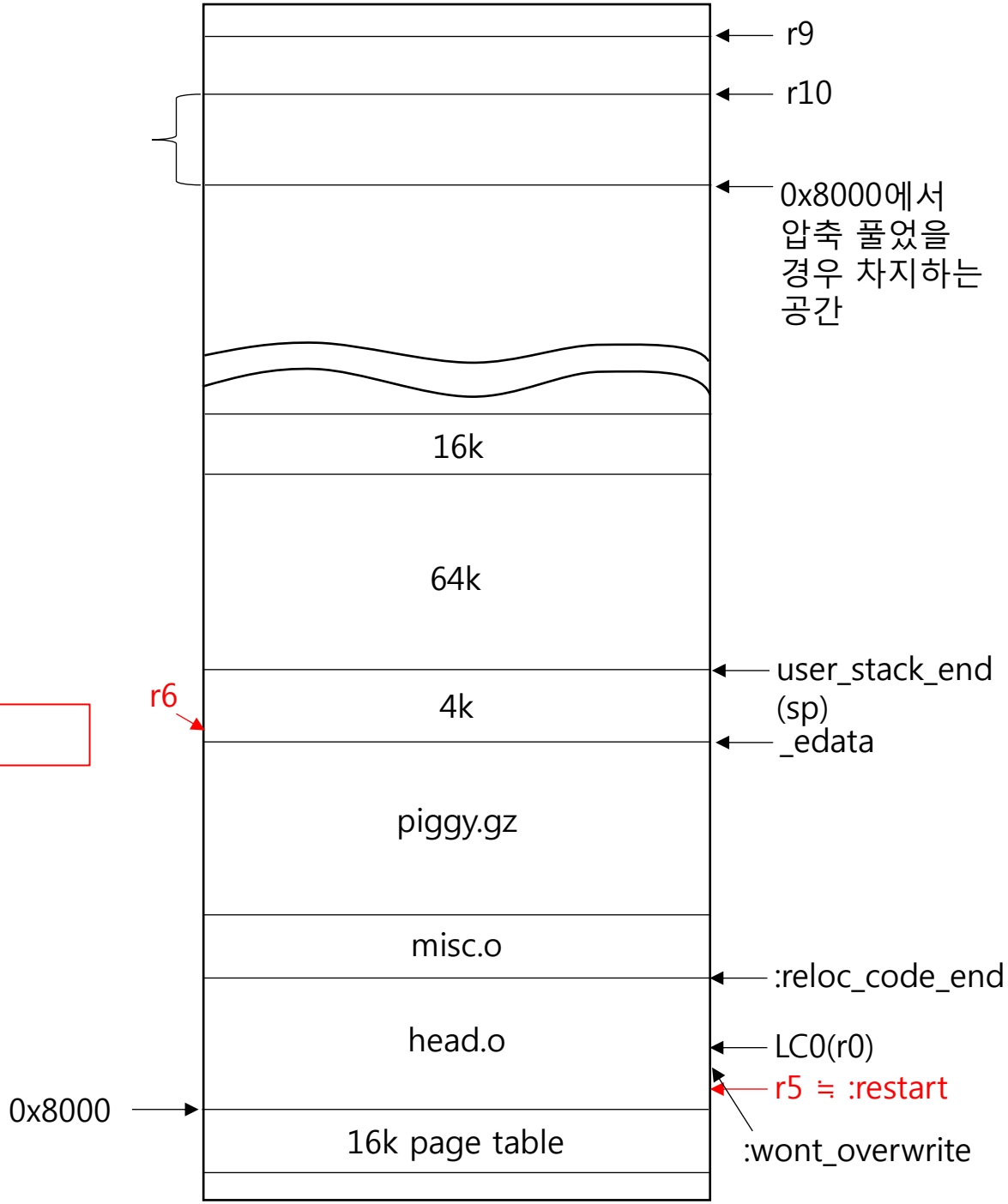
r6(src)를 시작으로 주소를 감소 시키면서 4byte씩 32bytes를 읽어서 r0-r3, r10-r12, lr에 저장



8

```
1:      ldmdb   r6!, {r0 – r3, r10 – r12, lr}
        cmp     r6, r5
        stmdb   r9!, {r0 – r3, r10 – r12, lr}
        bhi     1b
```

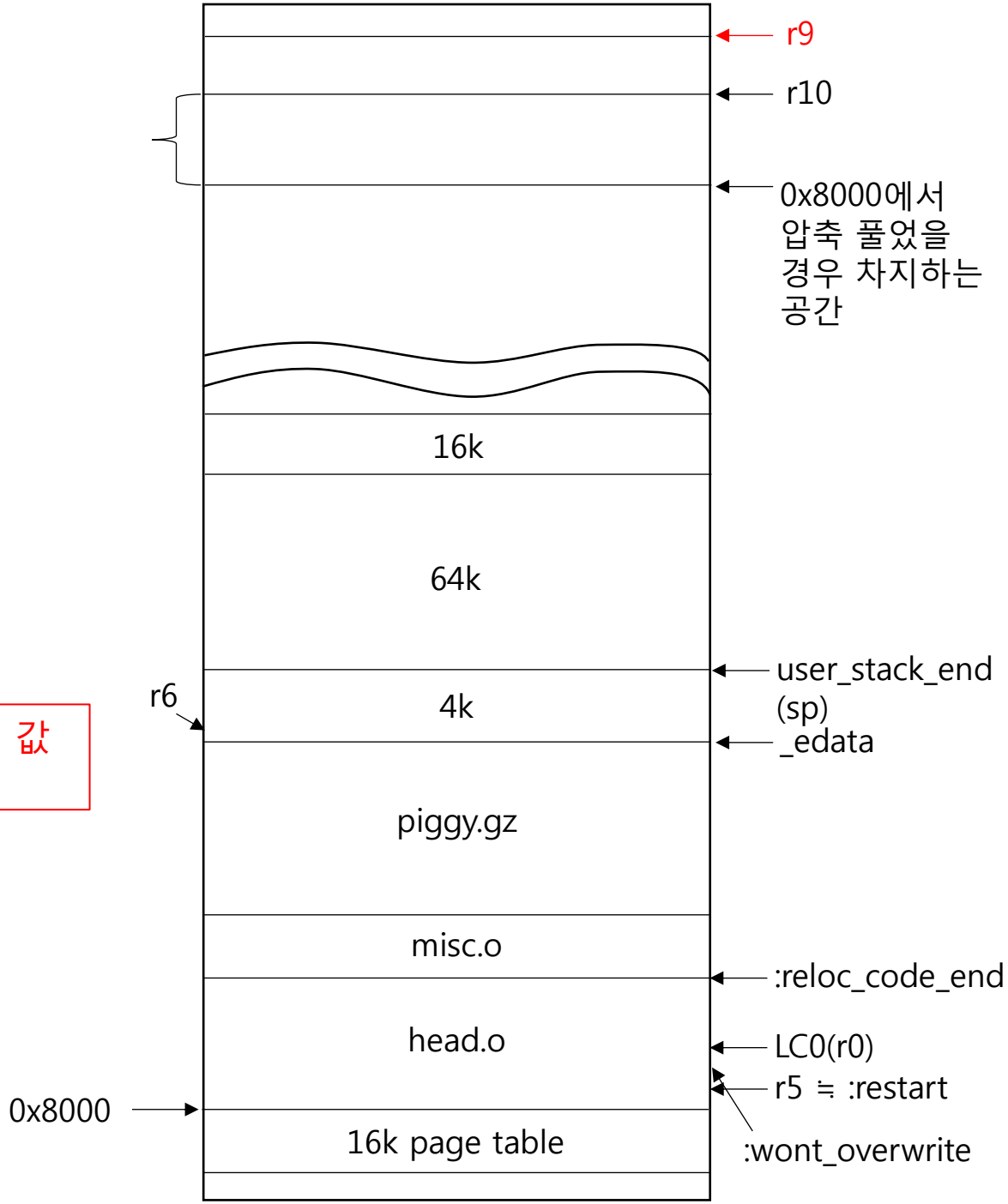
r6와 r5를 비교해서 r6 > r5 인지 (hi) 검사



8

```
1:      ldmdb   r6!, {r0 – r3, r10 – r12, lr}
        cmp     r6, r5
        stmdb   r9!, {r0 – r3, r10 – r12, lr}
        bhi     1b
```

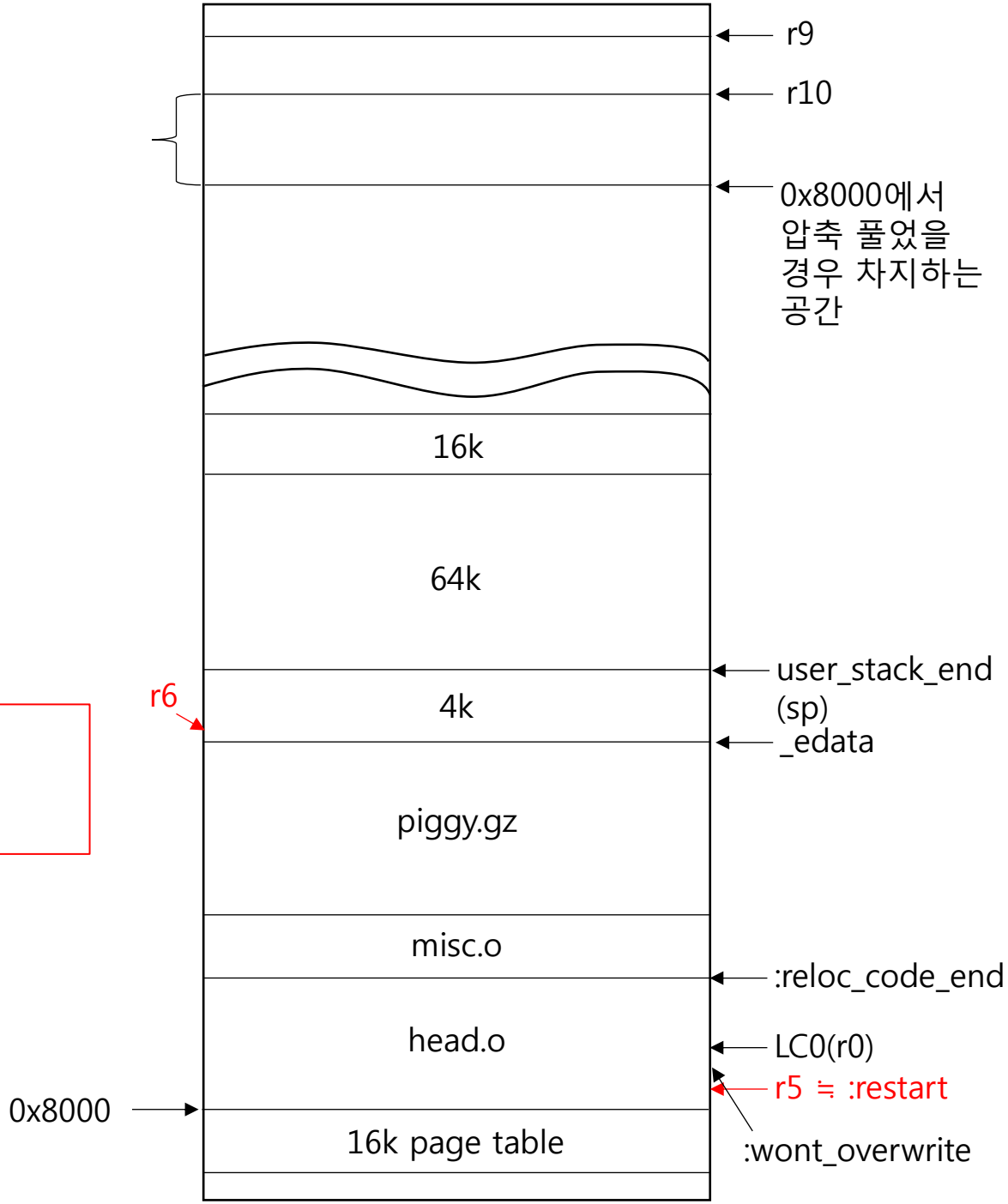
r9(dst)를 시작으로 주소를 감소 시키면서 r0-r3, r10-r12, lr 에 있는 값을 4byte씩 32bytes를 저장



8

```
1:      ldmdb   r6!, {r0 – r3, r10 – r12, lr}
        cmp     r6, r5
        stmdb   r9!, {r0 – r3, r10 – r12, lr}
        bhi     1b
```

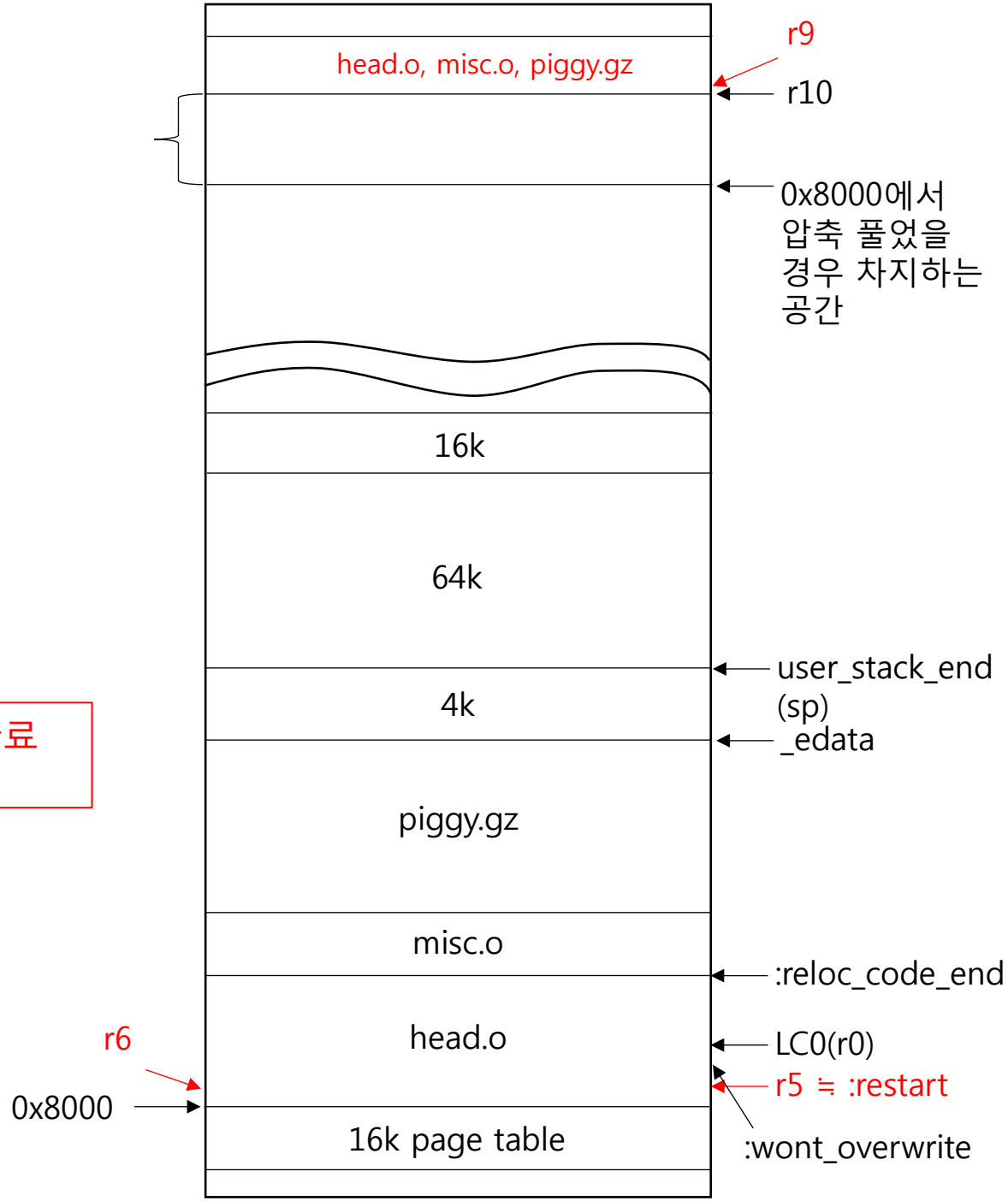
앞의 비교연산(cmp)에서 r6가 더 크기 때문에 (hi) 1b로 branch
이 과정을 r6가 r5보다 작아질 때 까지 반복
즉, asm의 memcopy로 재배치 하는 코드



8

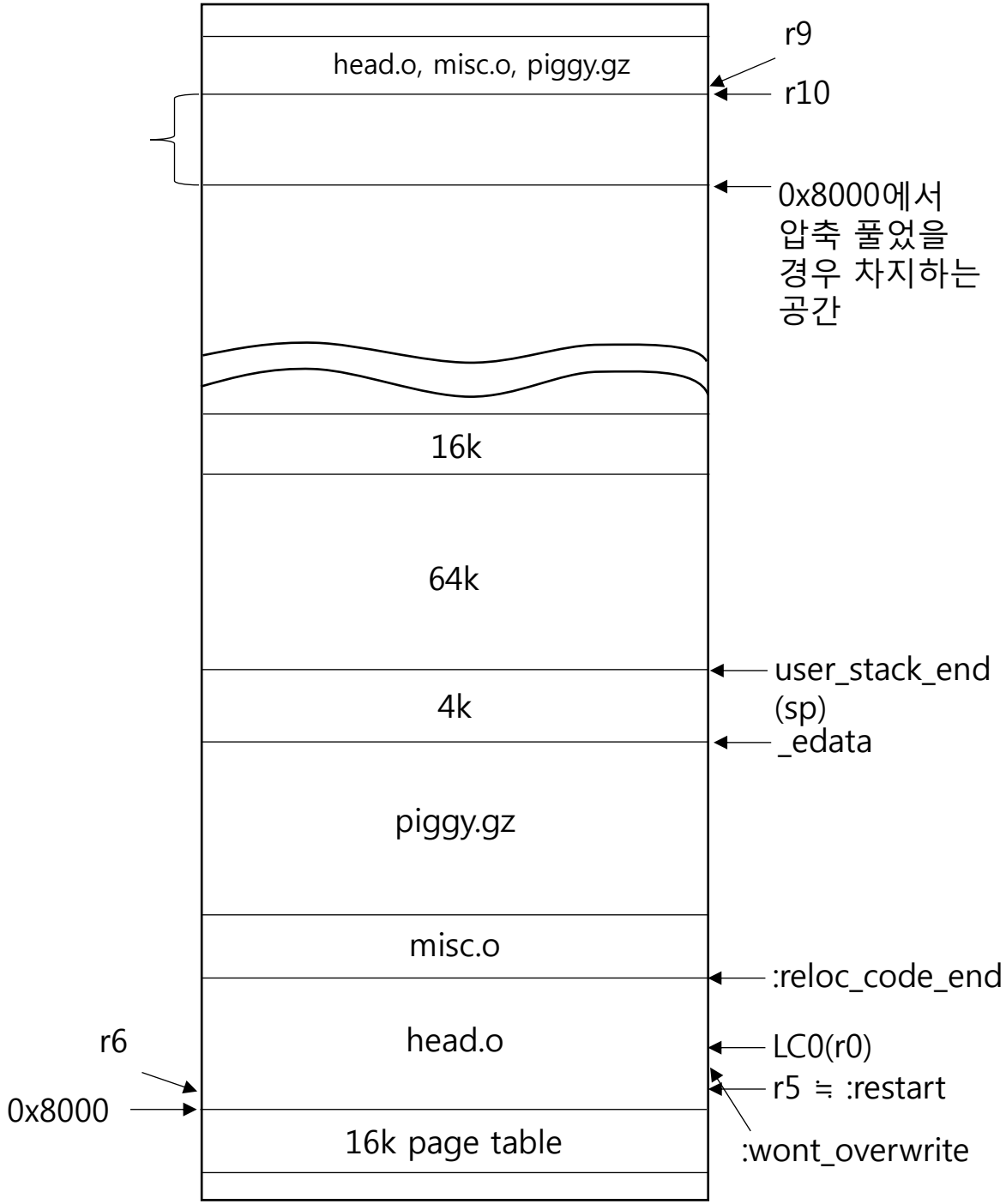
```
1:      ldmdb   r6!, {r0 – r3, r10 – r12, lr}
        cmp     r6, r5
        stmdb   r9!, {r0 – r3, r10 – r12, lr}
        bhi     1b
```

head.o(restart label 부터 시작), misc.o, piggy.gz를 모두 재배치 완료 하면 branch 종료



9

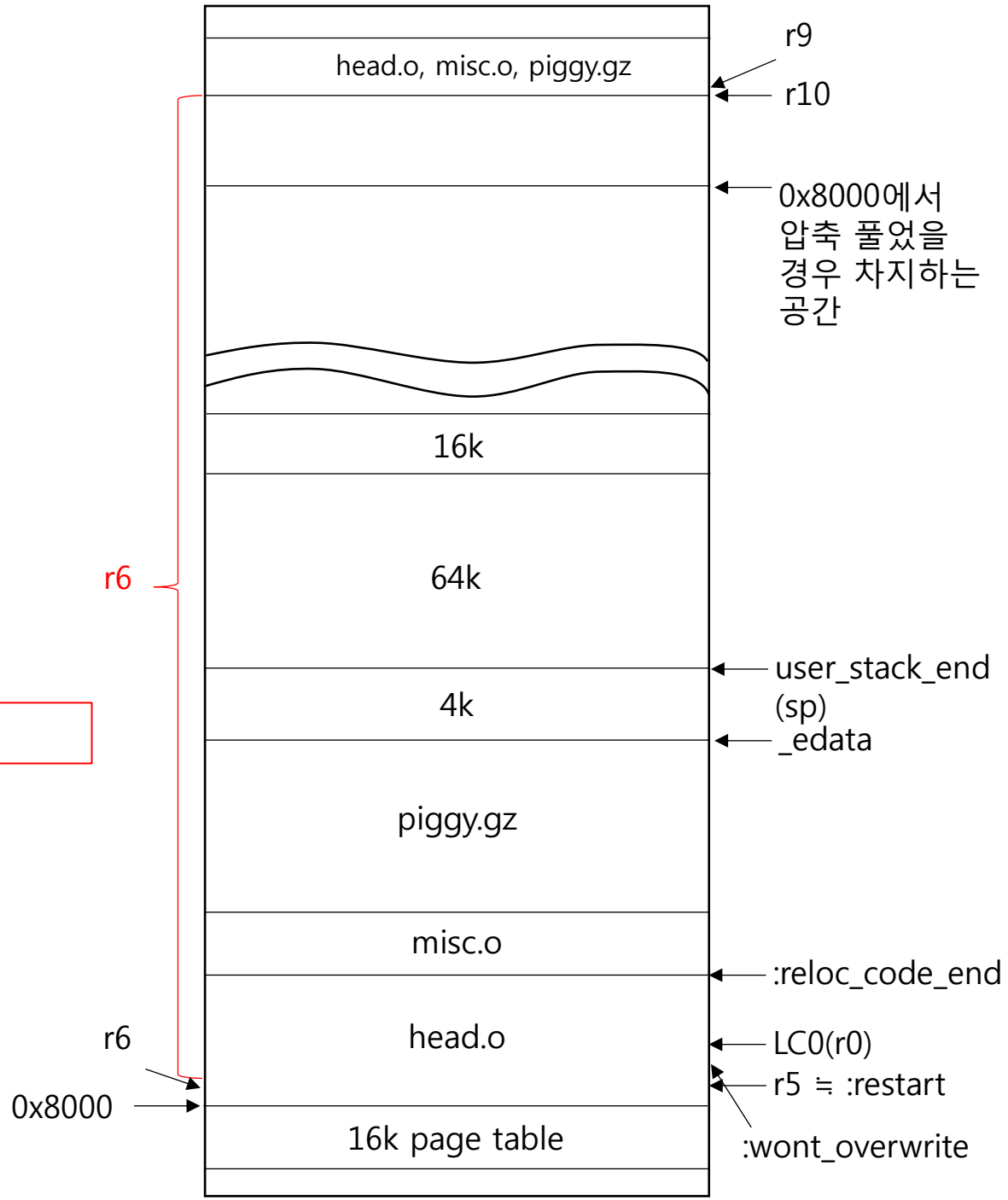
```
sub     r6, r9, r6
add     sp, sp, r6
bl      cache_clean_flush
```



9

```
sub     r6, r9, r6
add     sp, sp, r6
bl      cache_clean_flush
```

재배치 전/후 사이의 offset 계산해서 r6에 저장

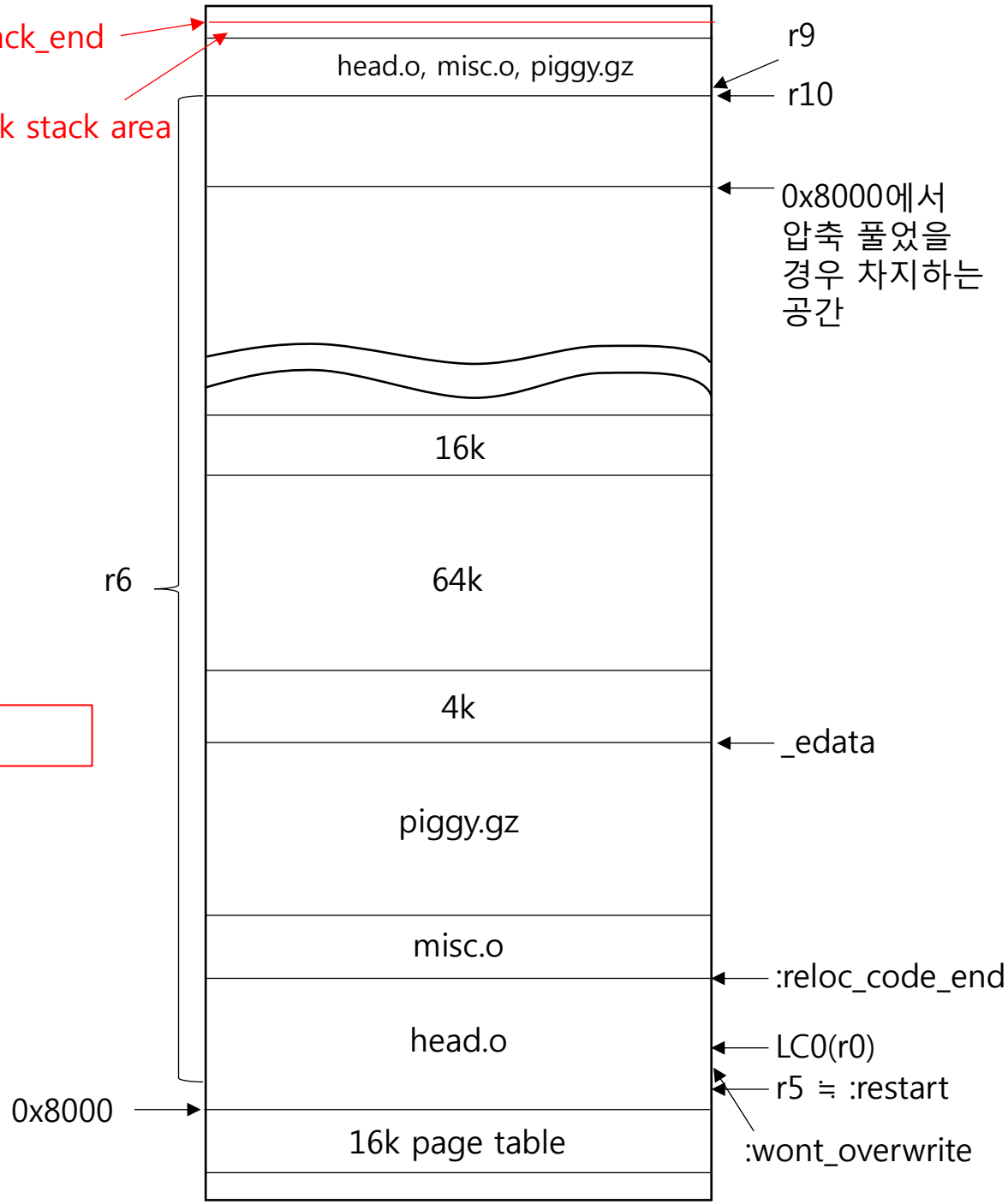


r6 = 재배치 전/후 사이의 offset

9

```
sub     r6, r9, r6
add     sp, sp, r6
bl      cache_clean_flush
```

현재 sp에 재배치 offset을 추가해서 sp 재조정



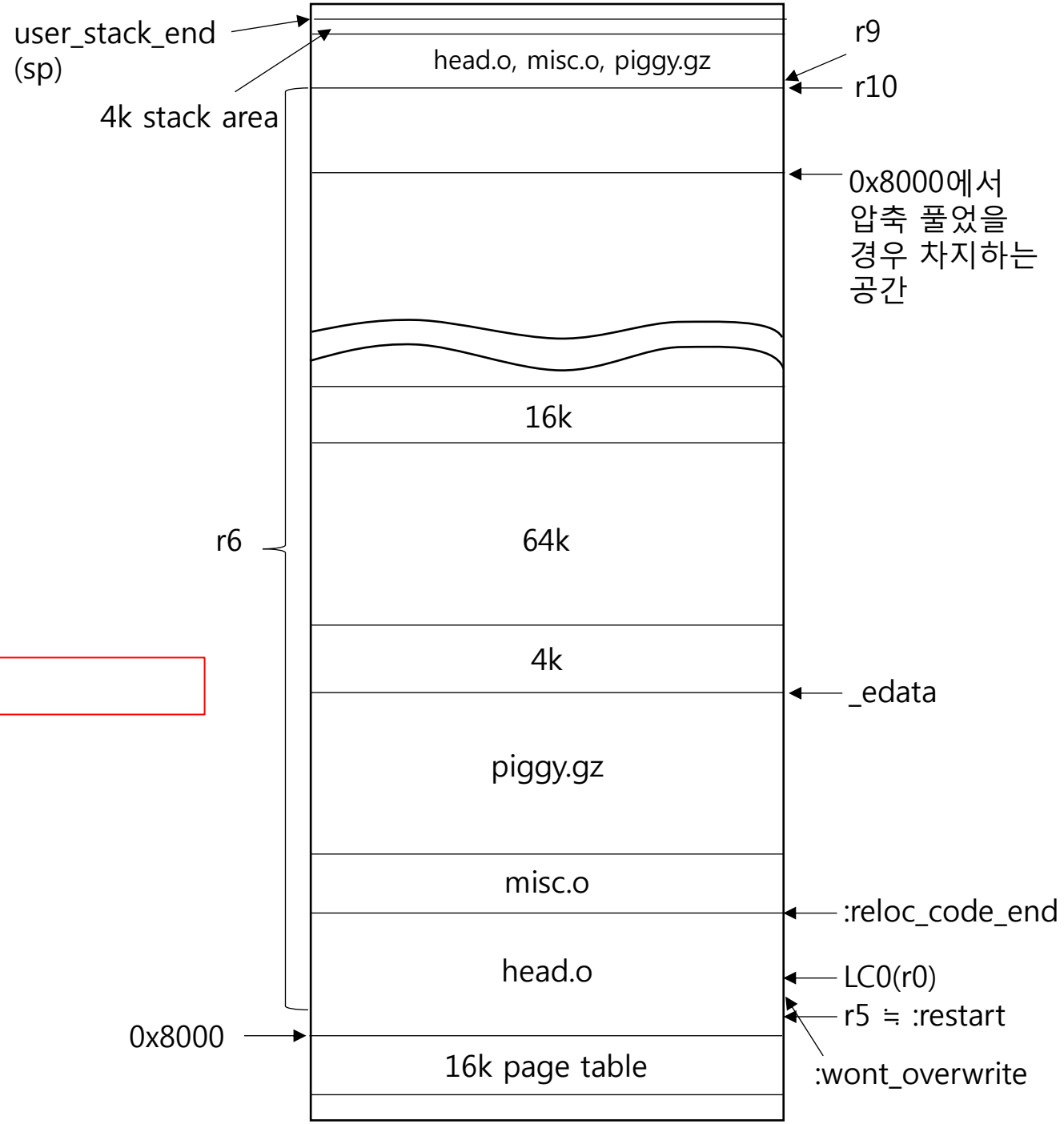
r6 = 재배치 전/후 사이의 offset

9

```
sub     r6, r9, r6
add     sp, sp, r6
bl      cache_clean_flush
```

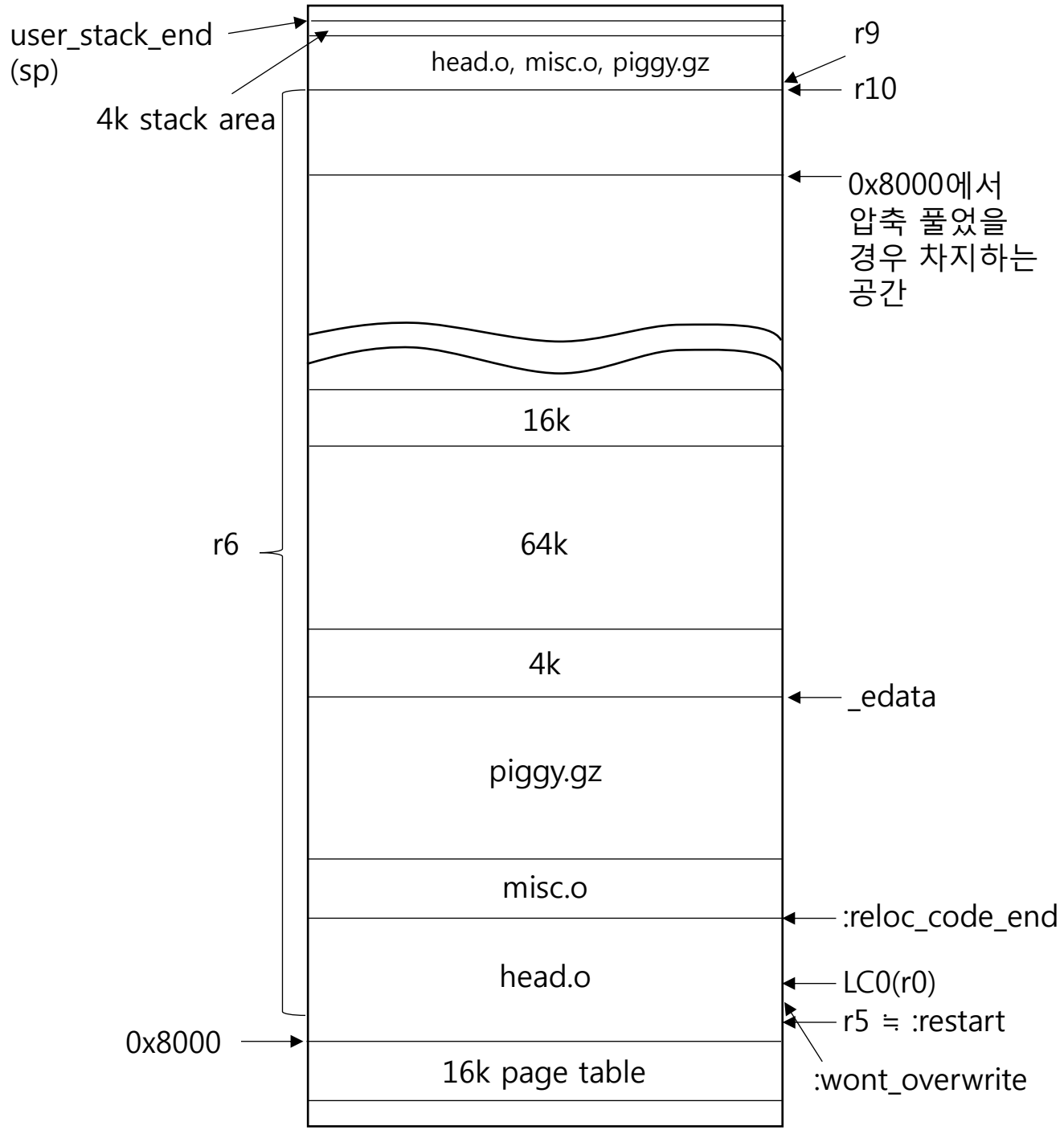
재배치한 곳에서 코드를 다시 수행 하기 위해 cache flush

r6 = 재배치 전/후 사이의 offset



10

```
adr     r0, BSYM(restart)
add     r0, r0, r6
mov     pc, r0
```



r6 = 재배치 전/후 사이의 offset

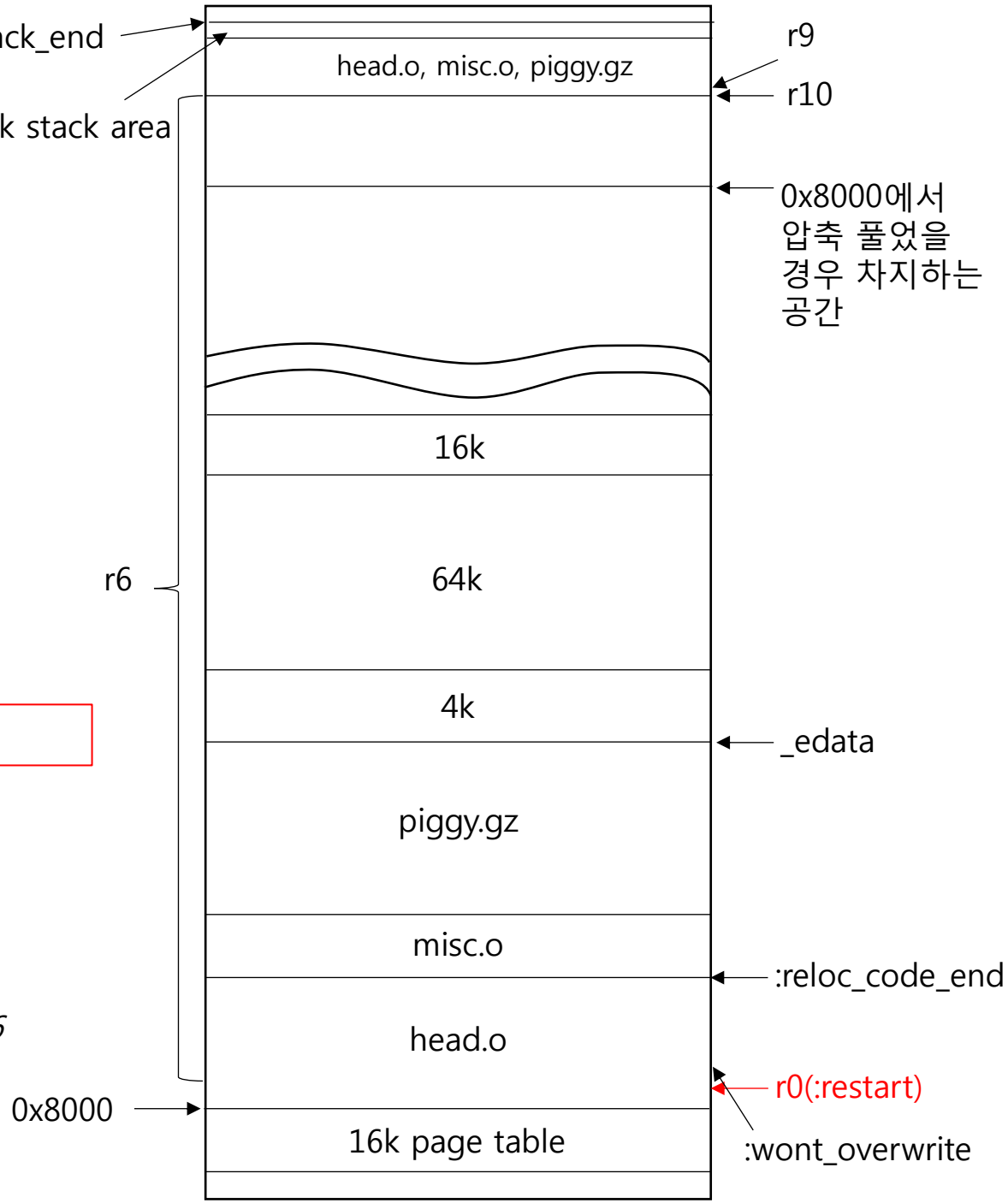
10

```
adr     r0, BSYM(restart)
add     r0, r0, r6
mov     pc, r0
```

restart label의 pc 상대 주소를 r0에 저장

참고 : BSYM macro는 badr macro로 대체 되었음
e.g. badr r0, restart
https://github.com/torvalds/linux/commit/14327c662822e5e874cb971a7162067519300ca8

r6 = 재배치 전/후 사이의 offset

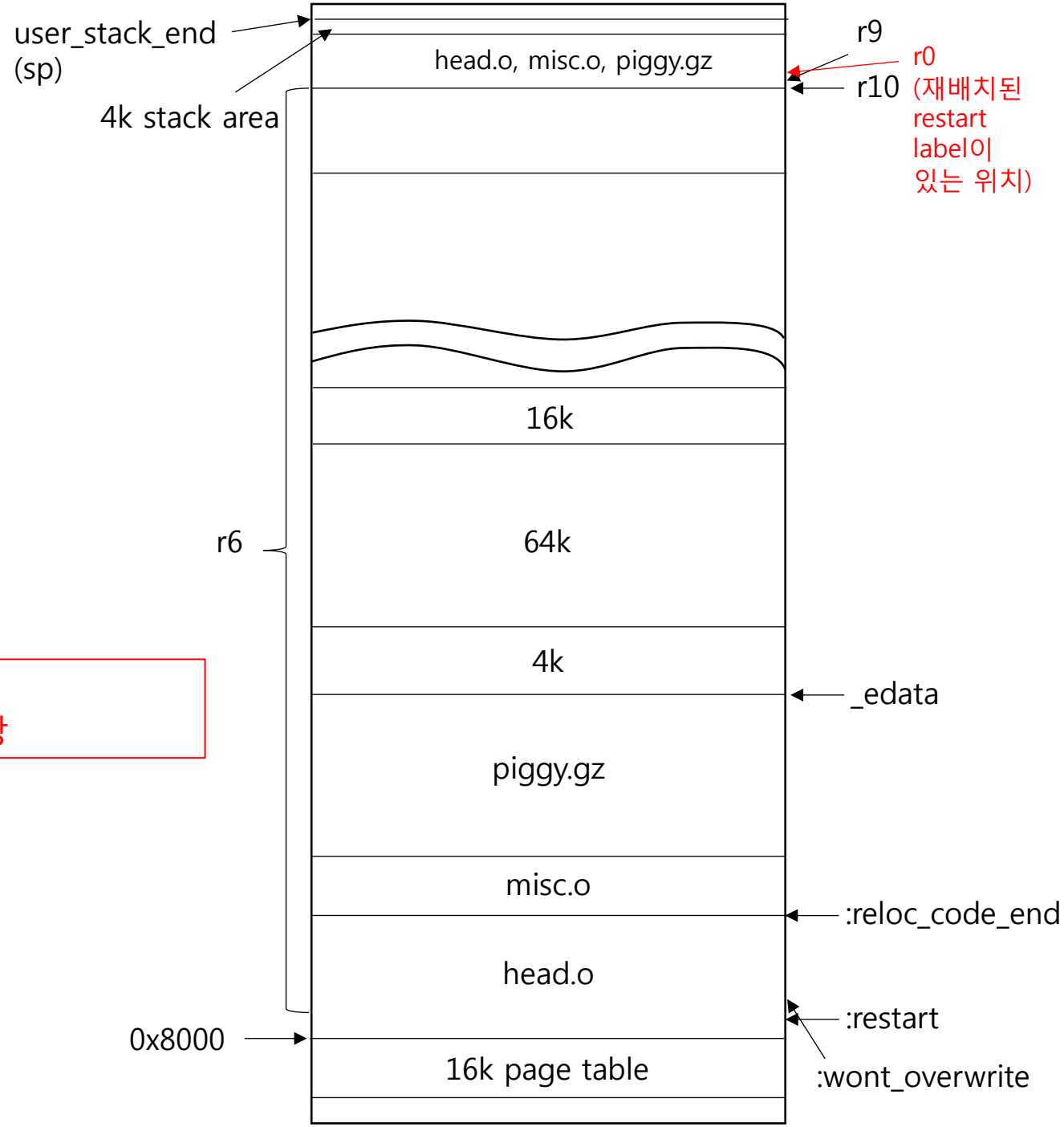


10

```
adr     r0, BSYM(restart)
add     r0, r0, r6
mov     pc, r0
```

r0에 재배치 offset을 추가
즉, 재배치 한 위치의 restart label이 있는 위치를 r0에 저장

r6 = 재배치 전/후 사이의 offset

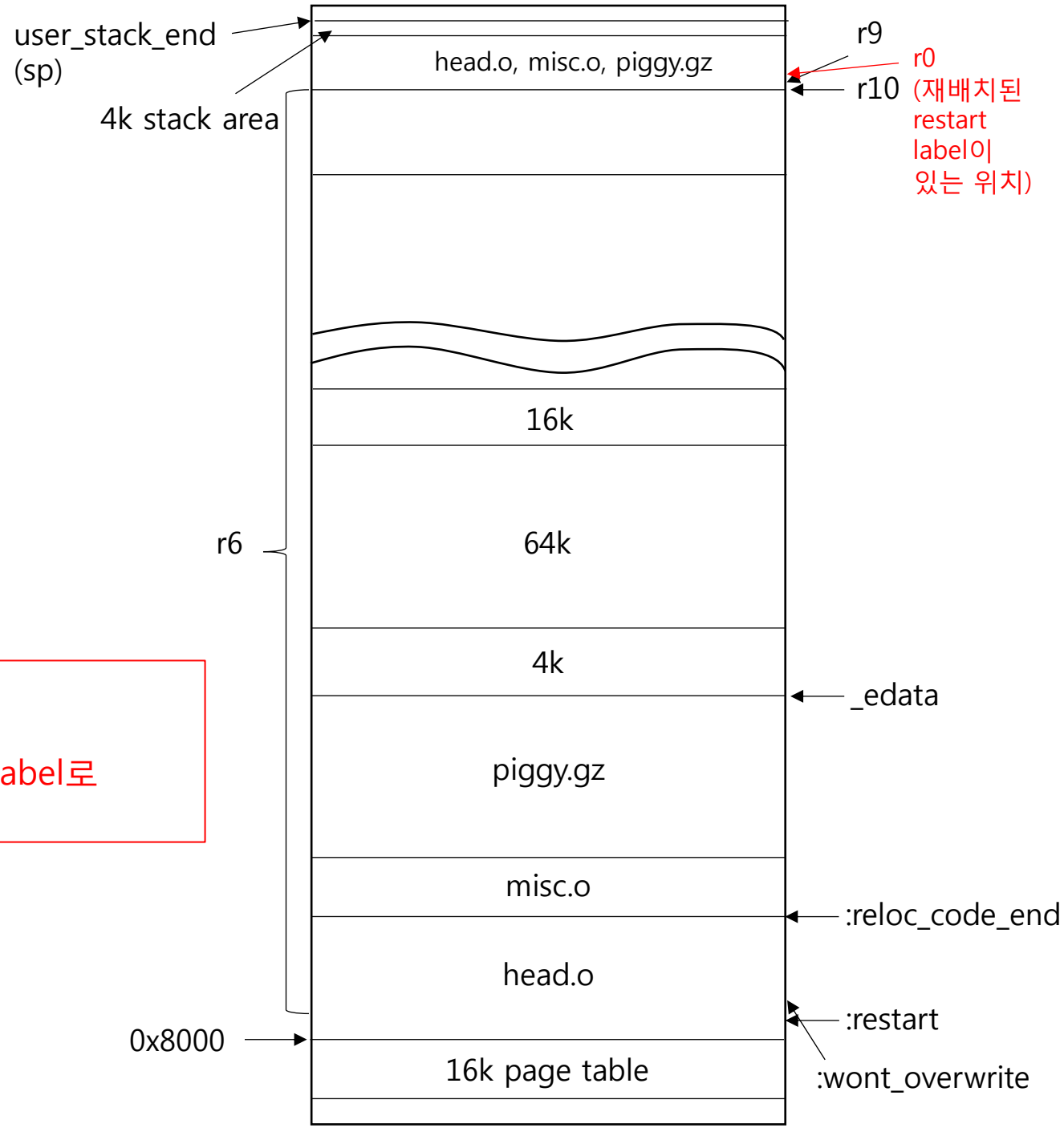


10

```
adr     r0, BSYM(restart)
add     r0, r0, r6
mov     pc, r0
```

pc를 r0로 설정
즉, 재배치된 restart label이 있는곳 부터 다시 코드를 실행
결과적으로 stack, heap을 다시 확보하고 wont_overwrite label로 branch하여 이후 코드(압축해제)를 실행

r6 = 재배치 전/후 사이의 offset



References

- https://github.com/torvalds/linux/blob/master/arch/arm/boot/compressed/head.S
- https://github.com/torvalds/linux/blob/master/arch/arm/boot/compressed/vmlinux.lds.S
- https://github.com/torvalds/linux/commit/14327c662822e5e874cb971a7162067519300ca8
- https://tools.ietf.org/html/rfc1952
- http://www.onicos.com/staff/iz/formats/gzip.html

Done !!