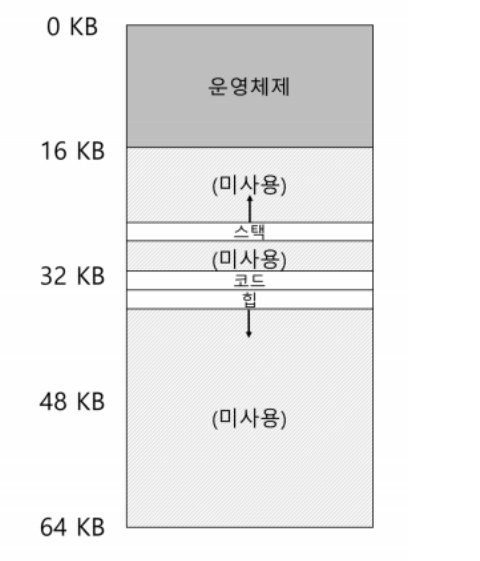
CDA0017: Operating Systems

Donghyun Kang (donghyun@changwon.ac.kr)

NOSLab (https://noslab.github.io)

Changwon National University

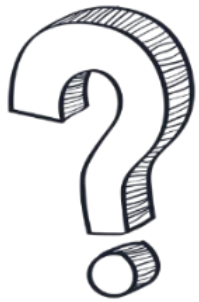
Summary



〈그림 19.2〉 물리 메모리에 세그멘트 배치하기

세그멘트	베이스	크기
코드	32 KB	2 KB
힙	34 KB	2 KB
스택	28 KB	2 KB

〈그림 19.3〉 세그멘트 레지스터의 값



세그먼트 기반 메모리 관리는 완벽한가?

페이징 (paging)

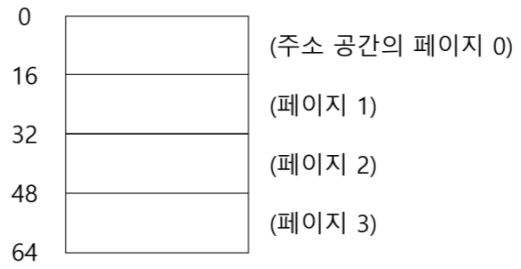
- 가변 크기의 논리 세그먼트로 나누는 것이 아니라 **고정 크기 단위**로 할당
  - 가변 크기의 논리 세그먼트: 코드, 힙, 스택

- 페이지 vs. 페이지 프레임
  - 가상 주소 공간을 페이지 (paging)라고 부름
  - 페이지에 상응하는 물리 메모리는 페이지 프레임 (page frame)이라고 부름

- 페이징의 장점
  - 효율적인 주소 공간 개념 지원
    - 힙, 스택의 방향에 대한 고려를 하지 않아도 됨
  - 공간관리의 단순함 제공

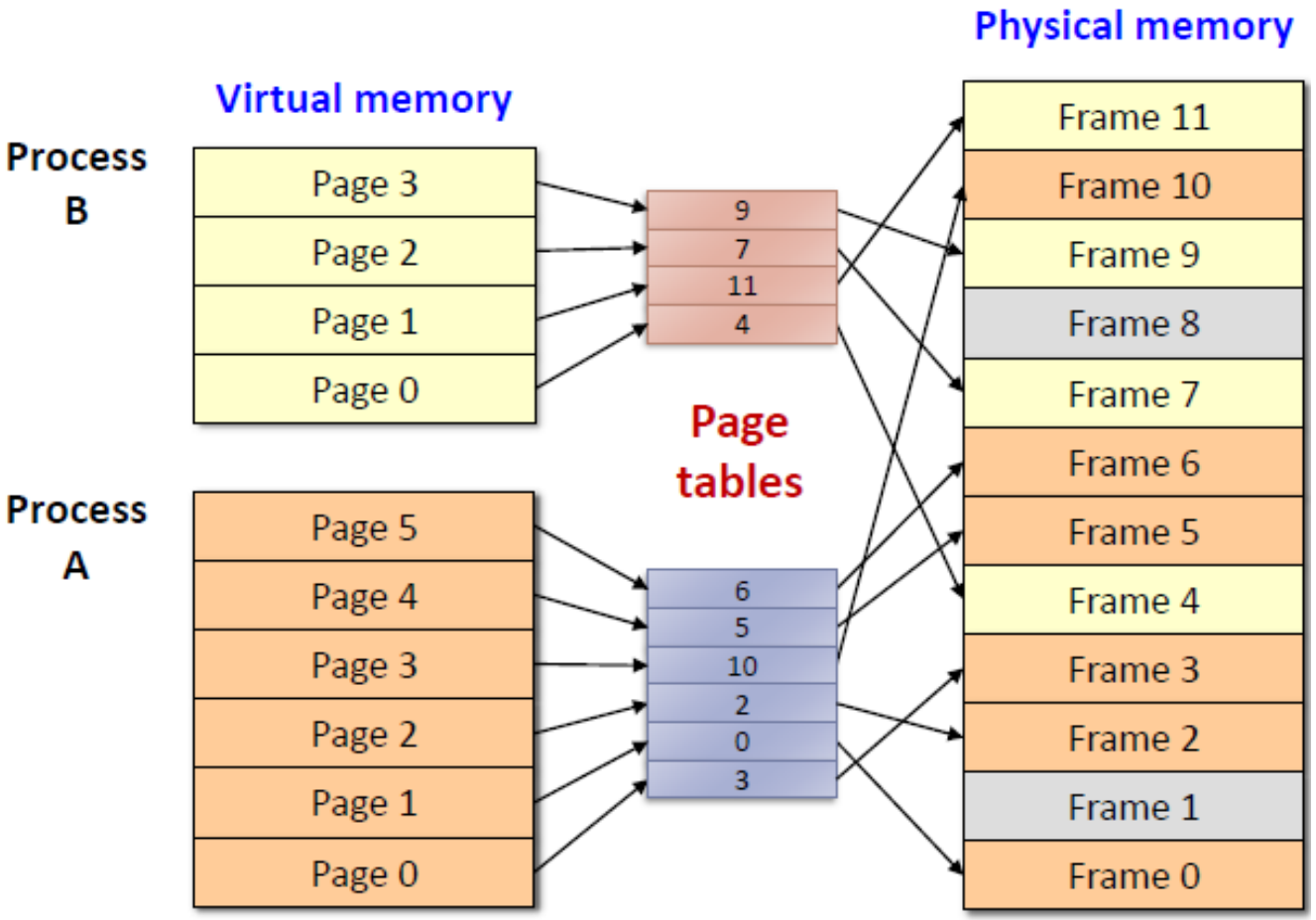
페이징 (paging) 예제

- 가상 주소 공간의 크기: 64 바이트
  - 한 페이지당 크기: 16 바이트



- 물리 주소 공간: 128 바이트로 구성

페이징 (paging) 예제



페이지 테이블 (page table)

- 가상 주소는 **운영체제**와 **하드웨어** 모두를 통해 물리 주소로 변환함
- 주소 공간의 가상 페이지에 대한 물리 메모리 위치 기록
  - 즉, 주소변환 (address translation) 정보 저장
- 각 프로세스마다 별도의 **페이지 테이블 (page table)** 자료 구조를 가짐

- 예
  - VP0 → PF3
  - VP1 → PF7
  - VP2 → PF5
  - VP3 → PF2

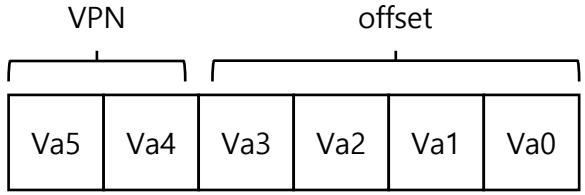
주소 변환

- 주소 변환을 위해 가상 주소를 2개의 파트로 분리함
  - 가상 페이지 번호 (virtual page number, VPN)
  - 오프셋 (offset)

- 64 바이트의 가상 주소 공간이 필요한 비트 수는?

주소 변환

- 가상 주소 공간의 관리
  - VPN: 가상 페이지 번호
    - 페이지 테이블의 인덱스(Index) 번호
  - Offset: 페이지에서의 위치

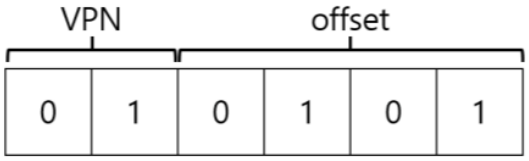


- 예제

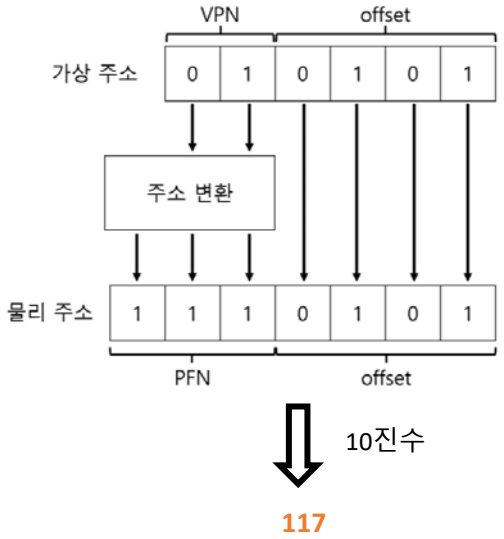
가상 주소

```
movl 21, %eax
```

2진수

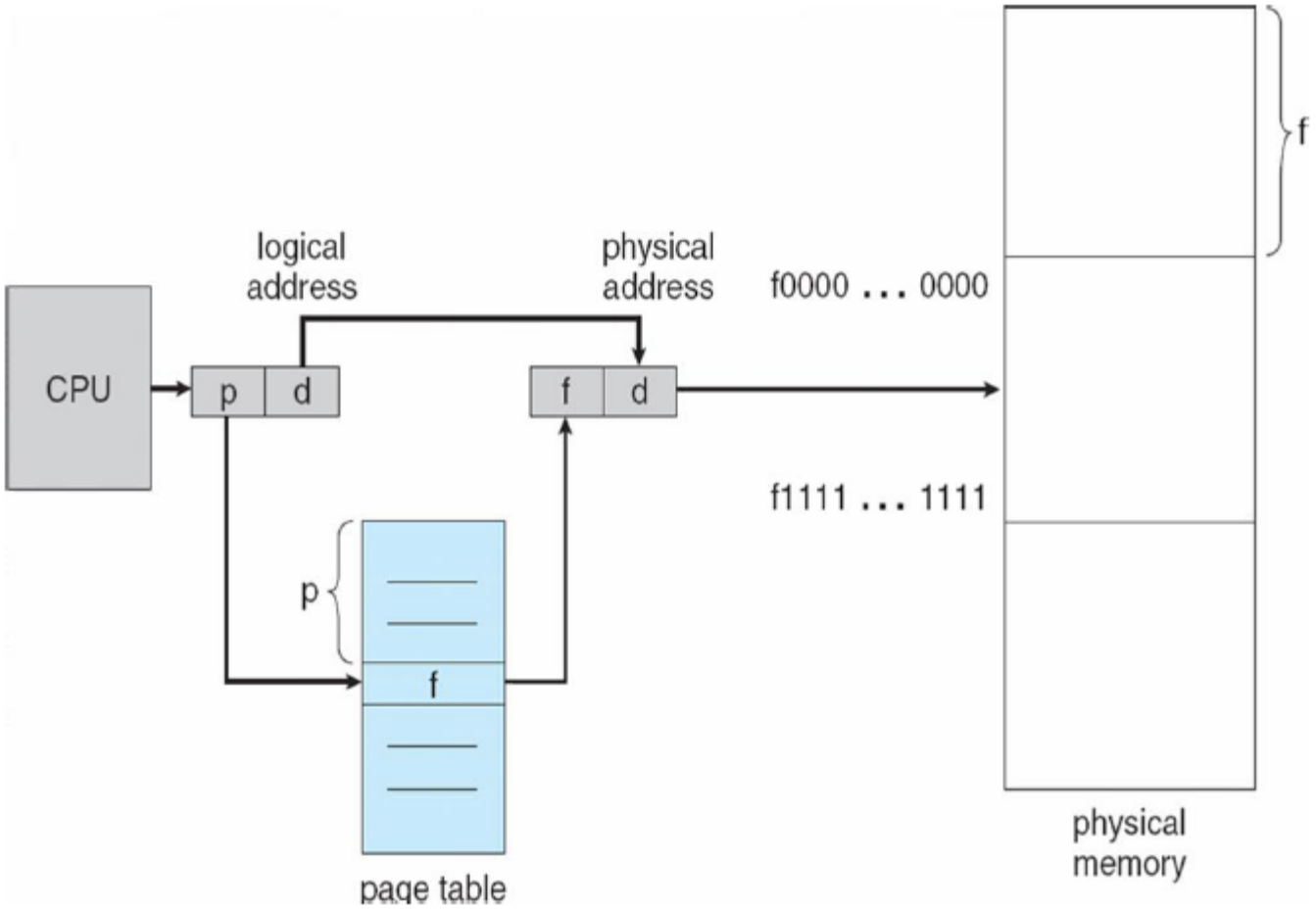


물리 주소



117

주소 변환


CPU
logical address
p
d
physical address
f
d
f0000 ... 0000
f1111 ... 1111
f
p
f
page table
physical memory

페이지 테이블 항목 (PTE)

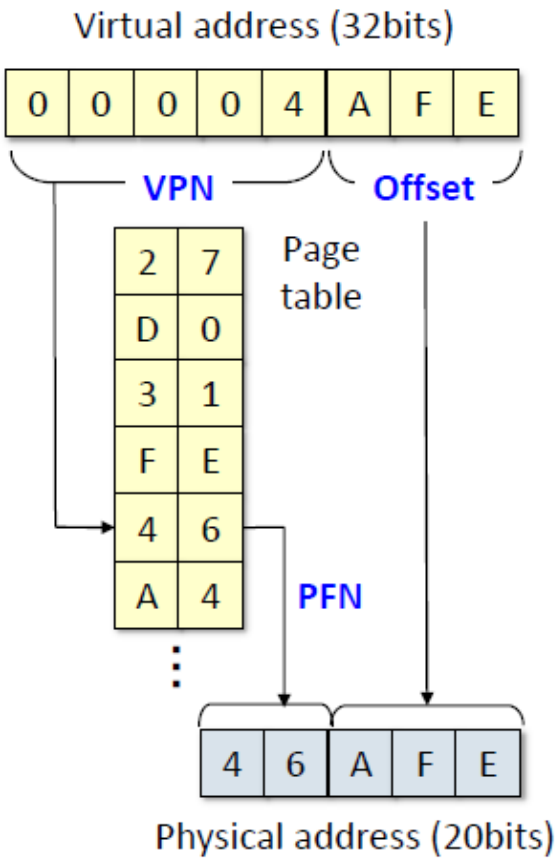
- 페이지 테이블 항목 (page table entry, PTE)
  - 물리 주소 변환 정보 저장
  - 기타 페이지의 다른 정보 저장

- 페이지 테이블 항목은 몇 개 인가?

페이지 테이블의 크기

- 예제 (가정)
  - 가상 주소: 32 bits
  - 물리 주소: 20 bits
  - 페이지 크기: 4KB

  - 오프셋: 12 bits
  - VPN: 20 bits
  - 페이지 테이블 엔트리: 2^{20}
  - 페이지 테이블의 크기는?

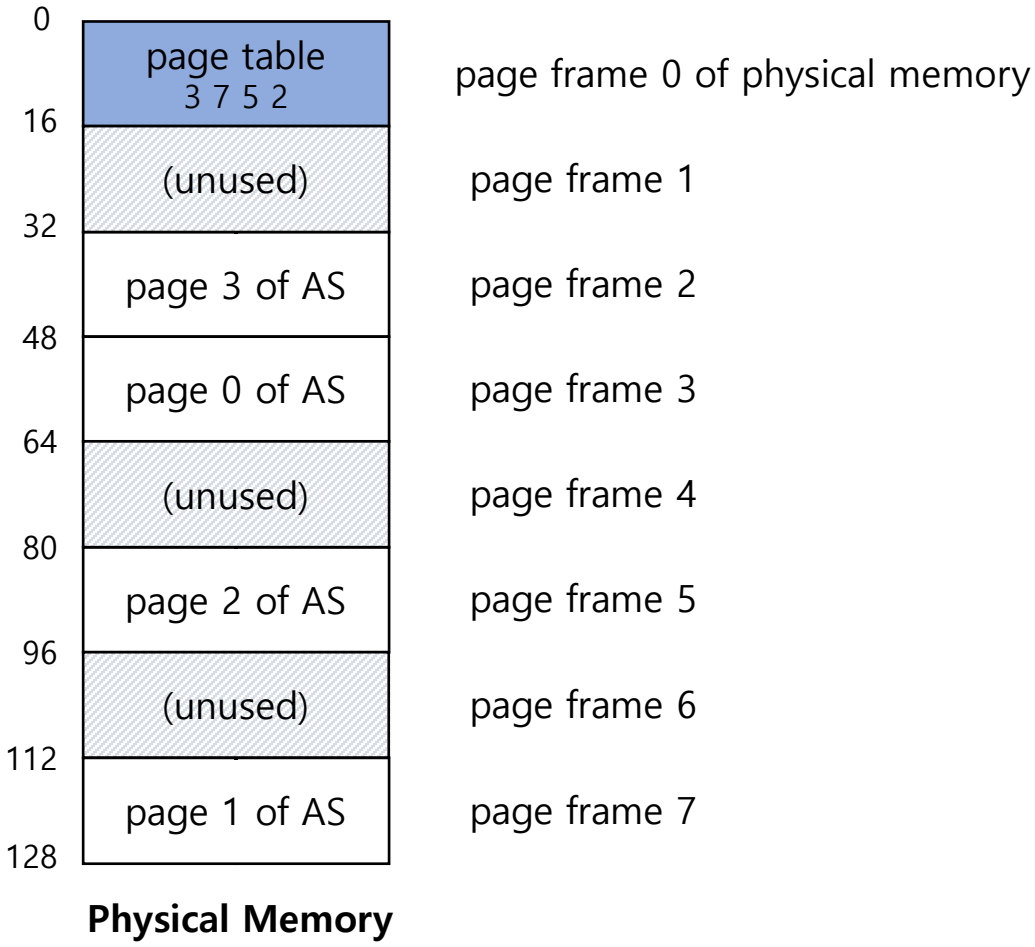


페이지 테이블의 저장

- 각 프로세스를 위한 페이지 테이블은 메모리에 저장됨

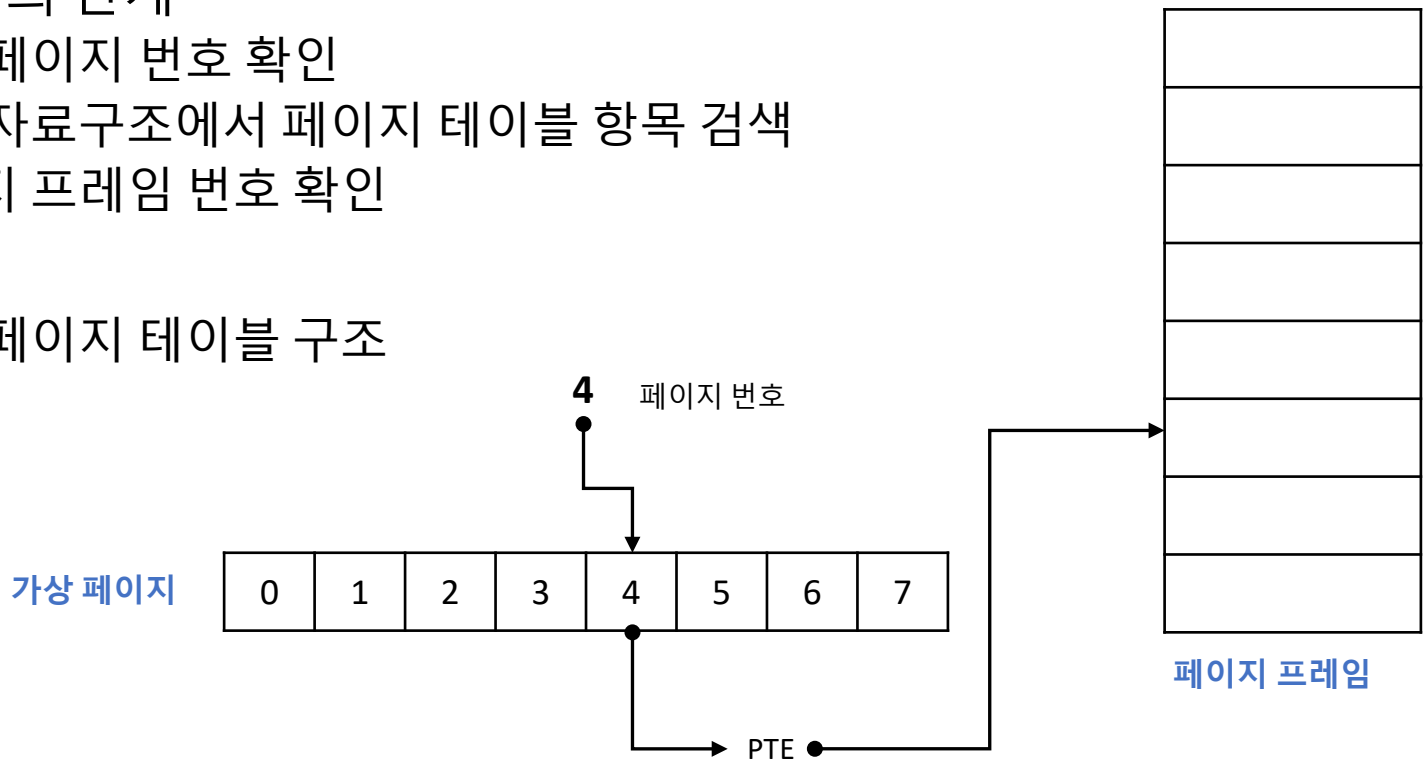
- 예제
  - 32-bit address space with 4-KB pages, 20 bits for VPN
    - $4MB = 2^{20} \, entries * 4 \, Bytes \, per \, page \, table \, entry$

가정 1: 페이지 테이블의 저장 (커널 메모리 영역)



페이지 테이블의 역할

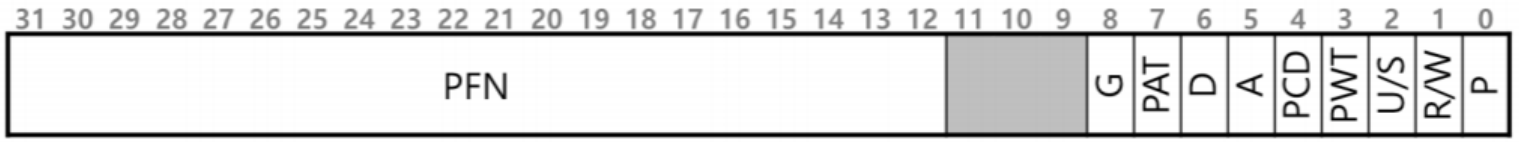
- 가상 주소를 물리 주소로 매핑
  - 즉, 가상 페이지 번호 → 페이지 프레임 번호
- 주소 매핑의 단계
  - 가상 페이지 번호 확인
  - 매핑 자료구조에서 페이지 테이블 항목 검색
  - 페이지 프레임 번호 확인
- 예
  - 선형 페이지 테이블 구조



페이지 테이블 항목 (PTE)

- X86의 가상 테이블 항목 (PTE)
  - Valid bit: 현재 물리 메모리를 사용하는지 판단하기 위한 비트
  - Protection bit: 읽기/쓰기/실행 권한을 설정하는 비트
  - Present bit: 현재 물리 메모리에 상주하는지 판단하기 위한 비트
  - Dirty bit: 메모리의 내용이 수정되었는지 판단하기 위한 비트
  - Reference bit: 페이지가 접근 되었는지 판단하기 위한 비트

- 예제



〈그림 21.5〉 x86 페이지 테이블 항목(PTE)

페이징을 위한 메모리 접근

```
1       // Extract the VPN from the virtual address
2       VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3
4       // Form the address of the page-table entry (PTE)
5       PTEAddr = PTBR + (VPN * sizeof(PTE))
6
7       // Fetch the PTE
8       PTE = AccessMemory(PTEAddr)
9
10      // Check if process can access the page
11      if (PTE.Valid == False)
12              RaiseException(SEGMENTATION_FAULT)
13      else if (CanAccess(PTE.ProtectBits) == False)
14              RaiseException(PROTECTION_FAULT)
15      else
16              // Access is OK: form physical address and fetch it
17              offset = VirtualAddress & OFFSET_MASK
18              PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19              Register = AccessMemory(PhysAddr)
```

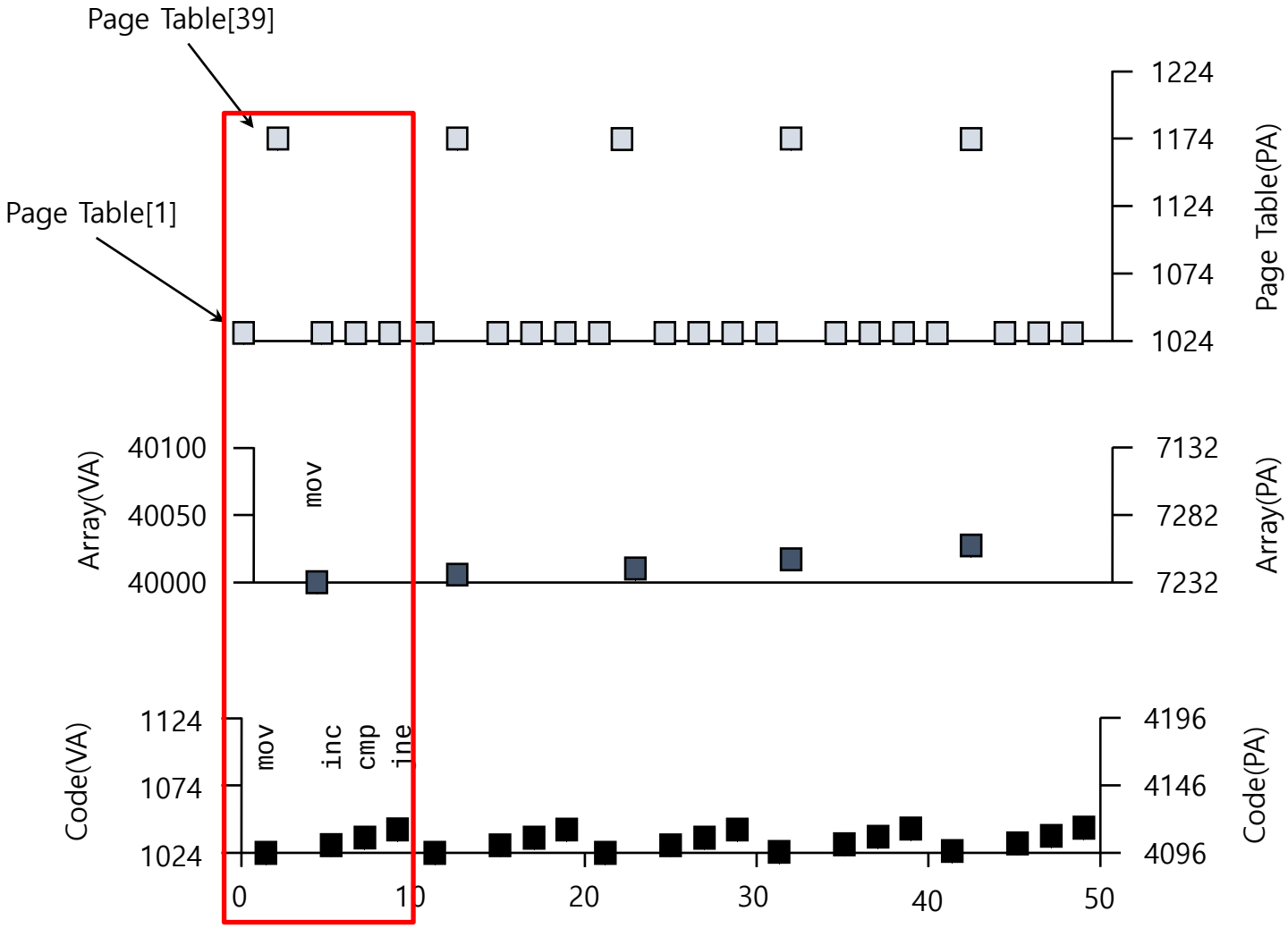
A Memory Trace

```
int array[1000];
...
for (i = 0; i < 1000; i++)
        array[i] = 0;
```

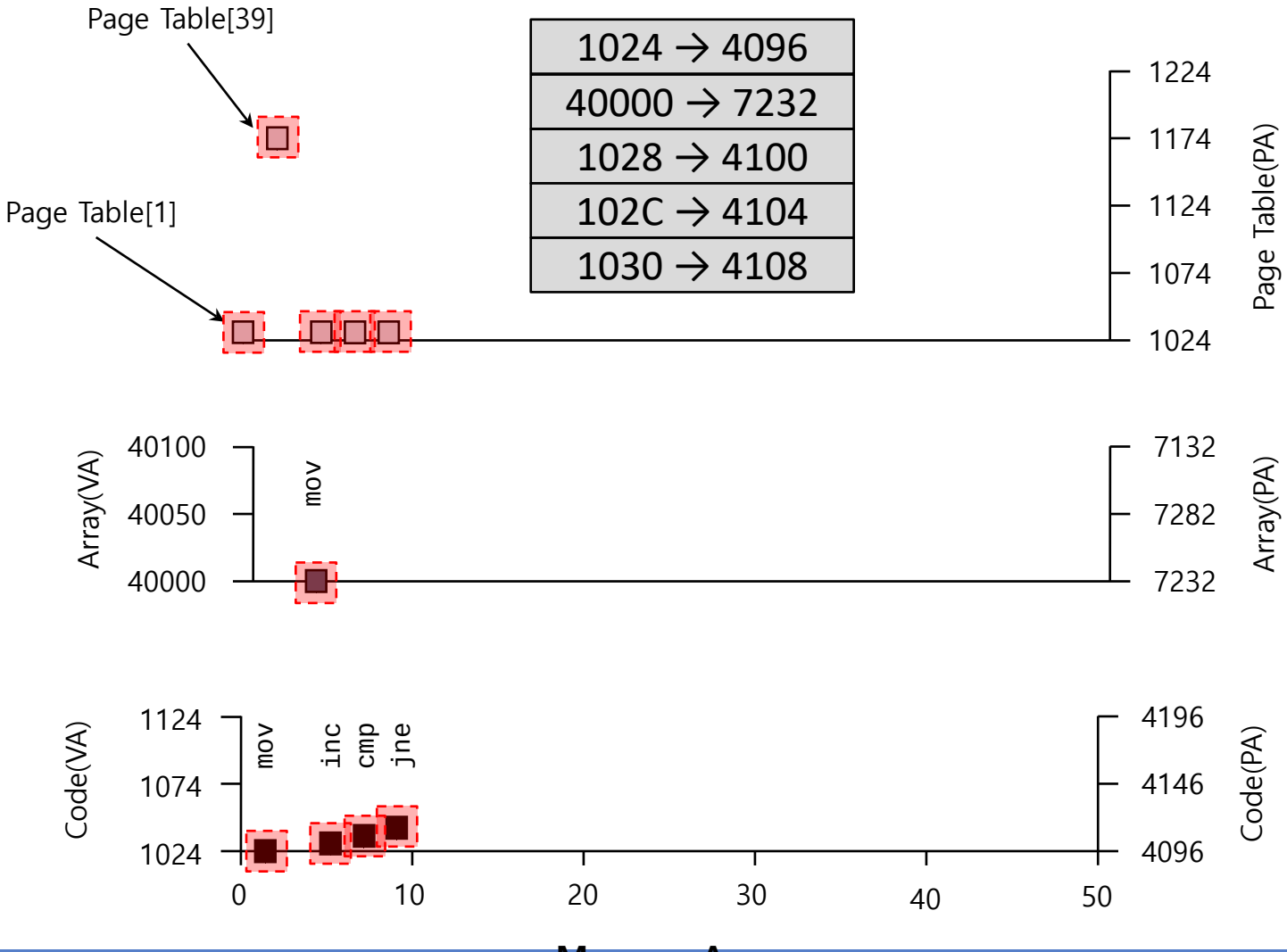
```
prompt> gcc –o array array.c –Wall –O
prompt>./array
```

```
0x1024 movl $0x0,(%edi,%eax,4)
0x1028 incl %eax
0x102c cmpl $0x03e8,%eax
0x1030 jne 0x1024
```

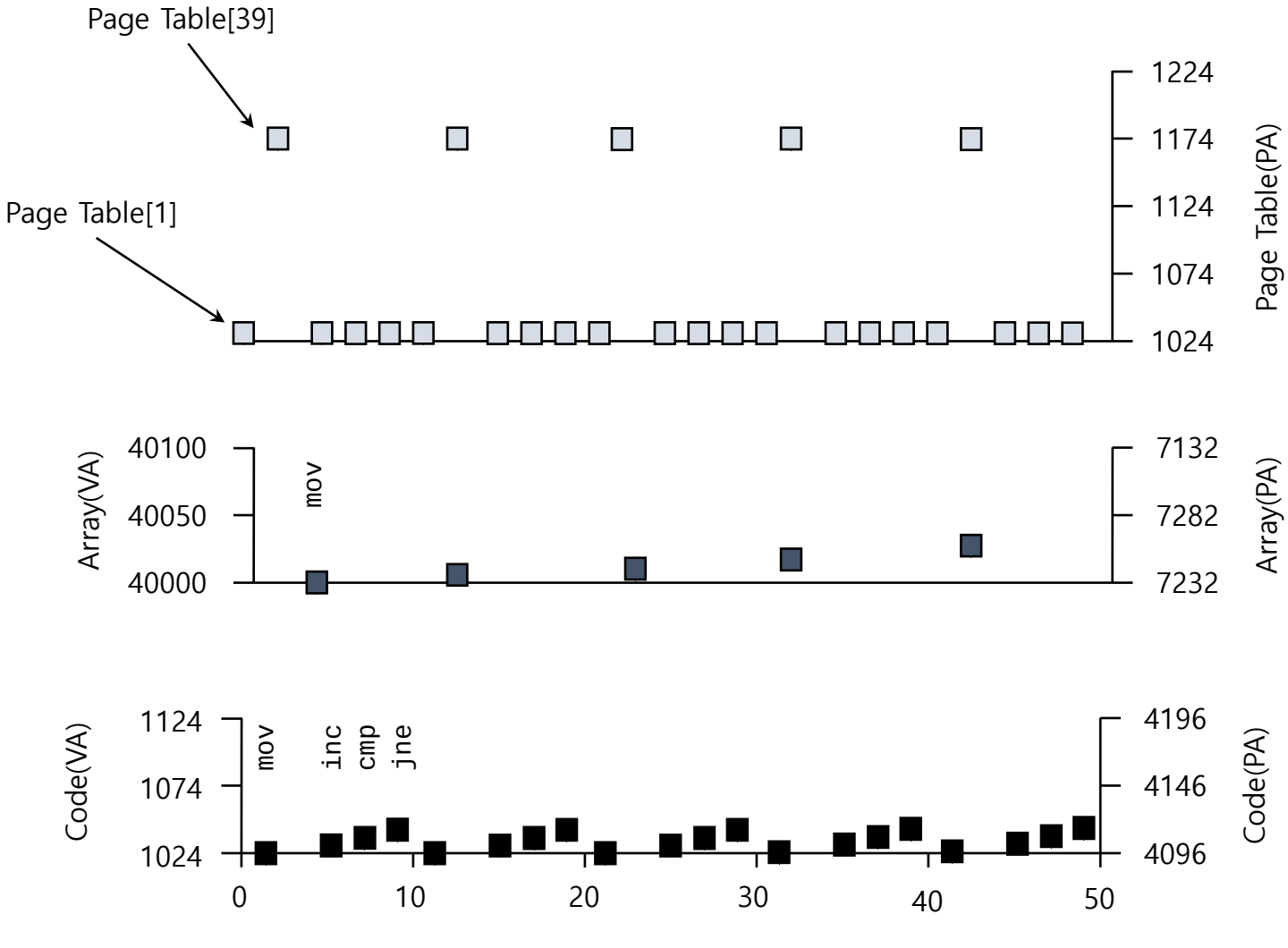
A Virtual(And Physical) Memory Trace



A Virtual(And Physical) Memory Trace



A Virtual(And Physical) Memory Trace



Q&A