부채널 공격 방지 구현 기법

정보컴퓨터공학과 권혁동

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

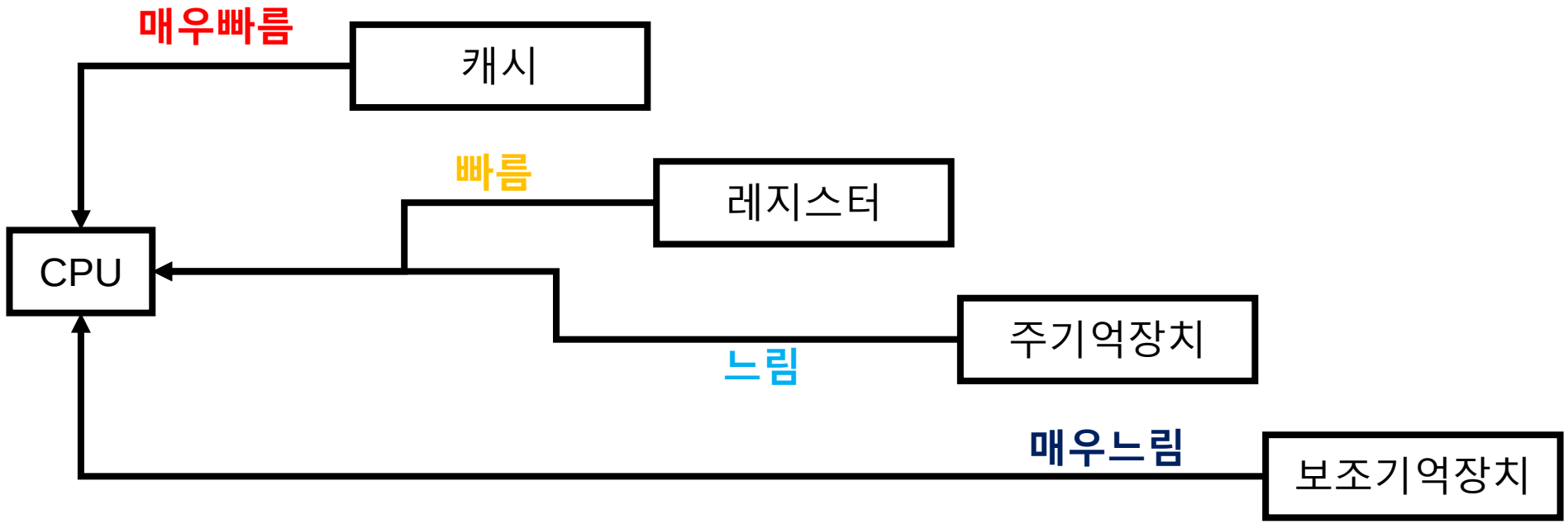
캐시 부채널 공격

타이밍 공격

결론

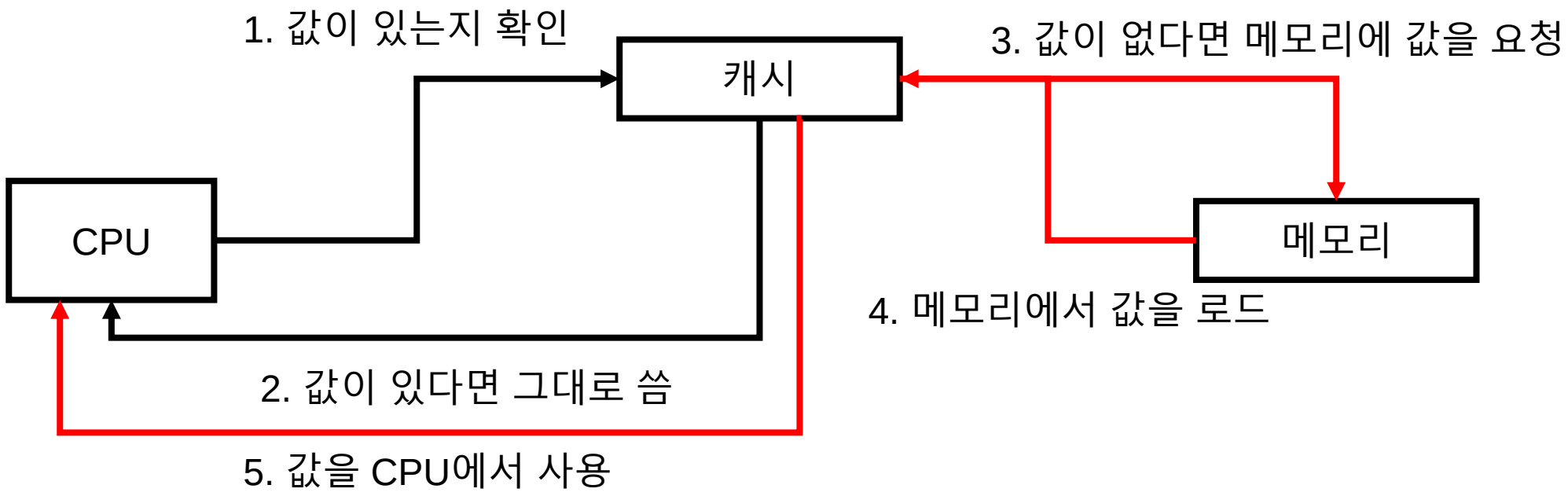
캐시 부채널 공격

- 컴퓨터에는 **캐시(cache)**라는 공간이 존재
  - CPU가 가장 빠르게 접근할 수 있는 메모리
  - 레지스터보다도 빠르지만 용량이 매우 적다
- 컴퓨터는 자주 사용하는 값을 캐시에 저장
  - 연산을 효율적으로 진행 가능



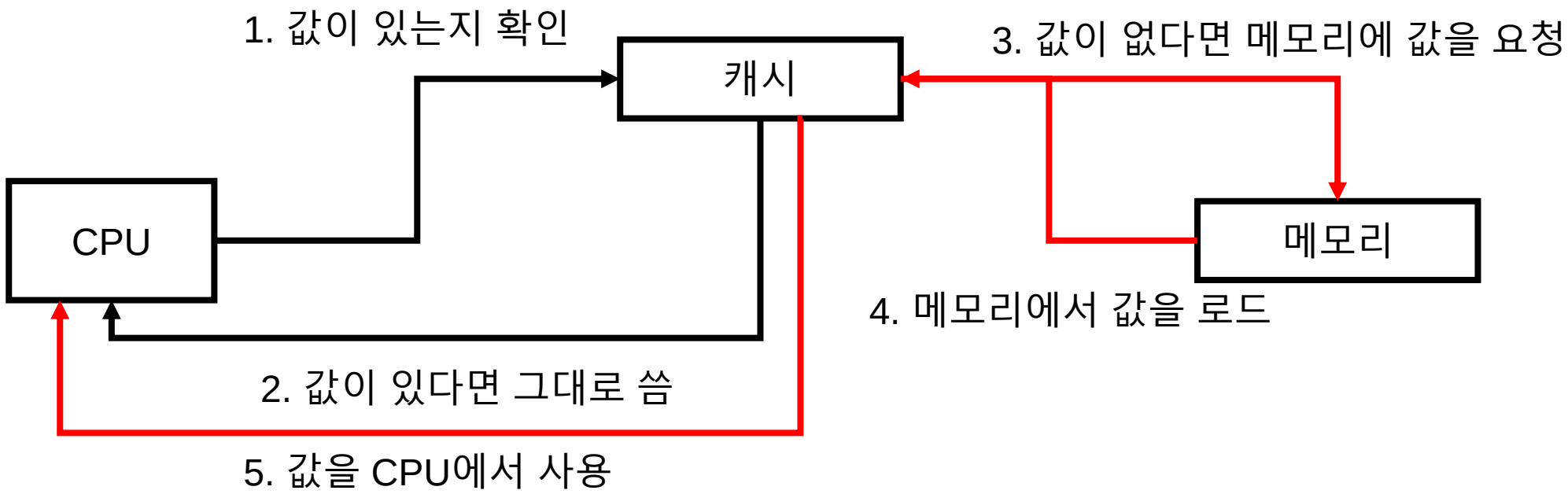
캐시 부채널 공격

- 캐시에 값을 탑재하는 순서
  1. CPU가 캐시에 필요한 값이 있는지 확인
  2. 필요한 값이 있다면 → 그대로 사용 (캐시 히트, cache hit)
  3. 필요한 값이 없다면 → 메모리에서 값 호출 (캐시 미스, cache miss)



캐시 부채널 공격

- 캐시 부채널 공격은 캐시 히트와 캐시 미스를 이용한 공격
  - **캐시 히트 시와 캐시 미스 시 발생하는 시간 차이**를 이용함
  - 멜트다운 공격이 이를 활용한 공격



캐시 부채널 공격

- 256-byte의 테이블에서 16-byte 값만 불러오는 코드
  - 값은 레지스터에 실리나, CPU의 빠른 연산을 위해 **캐시에도 값이 로드**
  - 공격자가 어떤 테이블을 사용했는지 알아보기 위해서 **연산 속도를 측정**

  → 256-byte 중 16-byte만 사용하기에 사용된 테이블은 조금 더 빠르게 연산

```
.balign 256
MUL_TABLE:
.byte 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf, \
0x0, 0x2, 0x3, 0x1, 0x8, 0xa, 0xb, 0x9, 0xc, 0xe, 0xf, 0xd, 0x4, 0x6, 0x7, 0x5, \
0x0, 0x3, 0x1, 0x2, 0xc, 0xf, 0xd, 0xe, 0x4, 0x7, 0x5, 0x6, 0x8, 0xb, 0x9, 0xa, \
0x0, 0x4, 0x8, 0xc, 0x6, 0x2, 0xe, 0xa, 0xb, 0xf, 0x3, 0x7, 0xd, 0x9, 0x5, 0x1, \
0x0, 0x5, 0xa, 0xf, 0x2, 0x7, 0x8, 0xd, 0x3, 0x6, 0x9, 0xc, 0x1, 0x4, 0xb, 0xe, \
0x0, 0x6, 0xb, 0xd, 0xe, 0x8, 0x5, 0x3, 0x7, 0x1, 0xc, 0xa, 0x9, 0xf, 0x2, 0x4, \
0x0, 0x7, 0x9, 0xe, 0xa, 0xd, 0x3, 0x4, 0xf, 0x8, 0x6, 0x1, 0x5, 0x2, 0xc, 0xb, \
0x0, 0x8, 0xc, 0x4, 0xb, 0x3, 0x7, 0xf, 0xd, 0x5, 0x1, 0x9, 0x6, 0xe, 0xa, 0x2, \
0x0, 0x9, 0xe, 0x7, 0xf, 0x6, 0x1, 0x8, 0x5, 0xc, 0xb, 0x2, 0xa, 0x3, 0x4, 0xd, \
0x0, 0xa, 0xf, 0x5, 0x3, 0x9, 0xc, 0x6, 0x1, 0xb, 0xe, 0x4, 0x2, 0x8, 0xd, 0x7, \
0x0, 0xb, 0xd, 0x6, 0x7, 0xc, 0xa, 0x1, 0x9, 0x2, 0x4, 0xf, 0xe, 0x5, 0x3, 0x8, \
0x0, 0xc, 0x4, 0x8, 0xd, 0x1, 0x9, 0x5, 0x6, 0xa, 0x2, 0xe, 0xb, 0x7, 0xf, 0x3, \
0x0, 0xd, 0x6, 0xb, 0x9, 0x4, 0xf, 0x2, 0xe, 0x3, 0x8, 0x5, 0x7, 0xa, 0x1, 0xc, \
0x0, 0xe, 0x7, 0x9, 0x5, 0xb, 0x2, 0xc, 0xa, 0x4, 0xd, 0x3, 0xf, 0x1, 0x8, 0x6, \
0x0, 0xf, 0x5, 0xa, 0x1, 0xe, 0x4, 0xb, 0x2, 0xd, 0x7, 0x8, 0x3, 0xc, 0x6, 0x9
```

```
ADR x4, MUL_TABLE
LSL w2, w2, #4
ADD x4, x4, x2

LD1.16b {v30}, [x4]
```

캐시 부채널 공격

- 이를 방지하기 위해서는 모든 테이블 값을 캐시에 담음
  - Apple M1의 캐시 라인은 128-byte로, 128-byte 단위로 캐시를 관리
  - 총 테이블 크기가 256-byte이므로, 2번 로드하여 모든 테이블을 저장 가능

```
.balign 256
MUL_TABLE:
.byte 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf, \
0x0, 0x2, 0x3, 0x1, 0x8, 0xa, 0xb, 0x9, 0xc, 0xe, 0xf, 0xd, 0x4, 0x6, 0x7, 0x5, \
0x0, 0x3, 0x1, 0x2, 0xc, 0xf, 0xd, 0xe, 0x4, 0x7, 0x5, 0x6, 0x8, 0xb, 0x9, 0xa, \
0x0, 0x4, 0x8, 0xc, 0x6, 0x2, 0xe, 0xa, 0xb, 0xf, 0x3, 0x7, 0xd, 0x9, 0x5, 0x1, \
0x0, 0x5, 0xa, 0xf, 0x2, 0x7, 0x8, 0xd, 0x3, 0x6, 0x9, 0xc, 0x1, 0x4, 0xb, 0xe, \
0x0, 0x6, 0xb, 0xd, 0xe, 0x8, 0x5, 0x3, 0x7, 0x1, 0xc, 0xa, 0x9, 0xf, 0x2, 0x4, \
0x0, 0x7, 0x9, 0xe, 0xa, 0xd, 0x3, 0x4, 0xf, 0x8, 0x6, 0x1, 0x5, 0x2, 0xc, 0xb, \
0x0, 0x8, 0xc, 0x4, 0xb, 0x3, 0x7, 0xf, 0xd, 0x5, 0x1, 0x9, 0x6, 0xe, 0xa, 0x2, \
0x0, 0x9, 0xe, 0x7, 0xf, 0x6, 0x1, 0x8, 0x5, 0xc, 0xb, 0x2, 0xa, 0x3, 0x4, 0xd, \
0x0, 0xa, 0xf, 0x5, 0x3, 0x9, 0xc, 0x6, 0x1, 0xb, 0xe, 0x4, 0x2, 0x8, 0xd, 0x7, \
0x0, 0xb, 0xd, 0x6, 0x7, 0xc, 0xa, 0x1, 0x9, 0x2, 0x4, 0xf, 0xe, 0x5, 0x3, 0x8, \
0x0, 0xc, 0x4, 0x8, 0xd, 0x1, 0x9, 0x5, 0x6, 0xa, 0x2, 0xe, 0xb, 0x7, 0xf, 0x3, \
0x0, 0xd, 0x6, 0xb, 0x9, 0x4, 0xf, 0x2, 0xe, 0x3, 0x8, 0x5, 0x7, 0xa, 0x1, 0xc, \
0x0, 0xe, 0x7, 0x9, 0x5, 0xb, 0x2, 0xc, 0xa, 0x4, 0xd, 0x3, 0xf, 0x1, 0x8, 0x6, \
0x0, 0xf, 0x5, 0xa, 0x1, 0xe, 0x4, 0xb, 0x2, 0xd, 0x7, 0x8, 0x3, 0xc, 0x6, 0x9
```

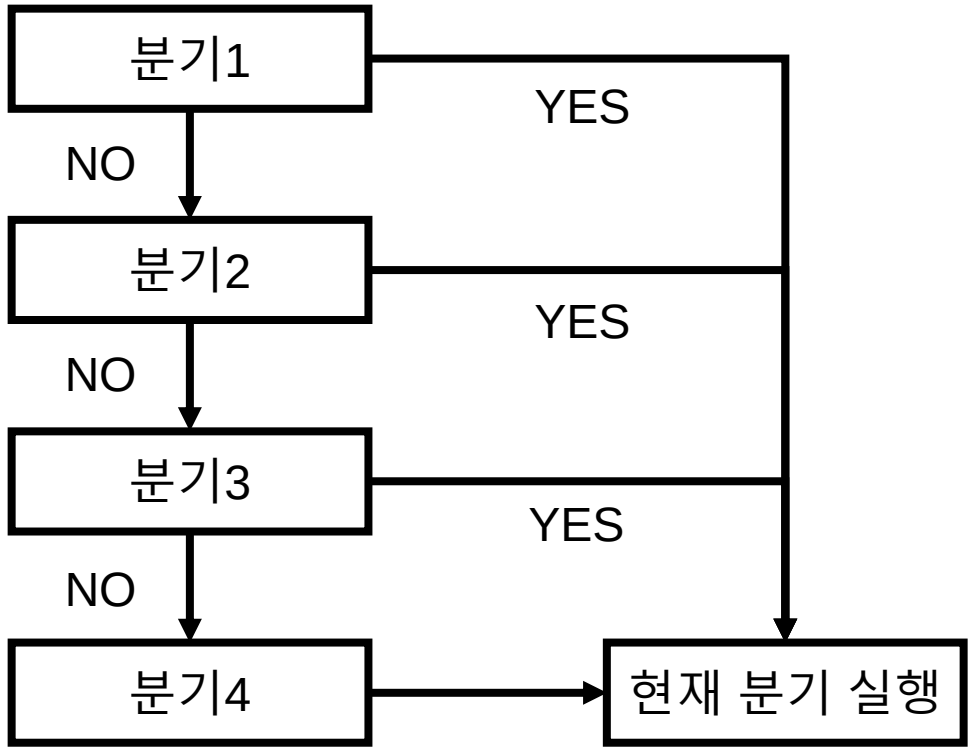
```
ADR x4, MUL_TABLE
LSL w2, w2, #4
ADD x4, x4, x2

LD1.16b {v30}, [x4]

SUB x4, x4, x2
ROR w2, w2, #4
XOR w2, w2, #8
LSL w2, w2, #4
ADD x4, x4, x2
LD1.16b {v27}, [x4]
```

타이밍 공격

- If-else 같은 분기문(branch)에서 발생하는 취약점
  - 캐시 부채널 공격과 비슷하게 시간을 측정
- 프로그래밍 언어 특성상 가장 첫 분기문 부터 실행
  - 분기 조건이 안맞으면 다음 분기로 이동
  - **분기가 많을 수록 실행 속도 차이가 커짐**



타이밍 공격

- 방지하기 위해서는 상수 시간 구현(constant time)이 필요
  - 하지만 조건을 따지기 시작하면, 상수 시간 구현이 어려워짐
  - **조건을 따지지 않고 실행**하는 형식으로 변경

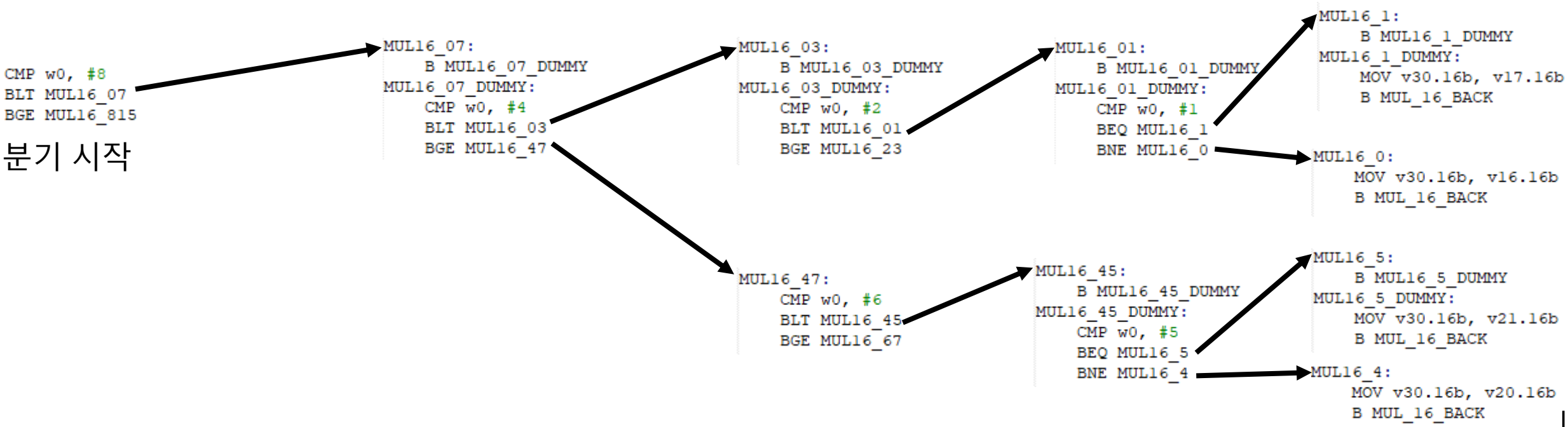
```
ADR x4, MUL_TABLE
LSL w2, w2, #4
ADD x4, x4, x2

LD1.16b {v30}, [x4]
```

1. 테이블 초기 주소 값을 가져와서
2. 전체 테이블 중 필요한 주소 값으로 이동
3. 해당 테이블 값을 로드

타이밍 공격

- 또는 모든 분기문을 원자화(atomic) 시키기
  - 어떤 분기에 접근하든 **명령어의 수와 사용한 명령어 종류가 항상 동일**
  - **실행 시간을 완전히 동일**하게 만들 수 있음
  - 아래의 예시는 모든 경우에서 14개 명령어(CMP 4, B 9, MOV 1)로 실행



결론

- 부채널 내성을 지니는 방법에 대해서 확인
  - 캐시 부채널: 캐시 히트와 미스를 이용한 공격

  → **캐시에 값을 상주**시키는 방법으로 방어
  - 타이밍 공격: 연산 시간의 차이를 이용한 공격

  → 모든 **연산 시간을 동일**하게 맞추는 것으로 방어
- 부채널 내성을 지니게 되면 연산 시간을 다소 희생
  - 각종 방어 기법과 연산 성능은 trade-off 관계