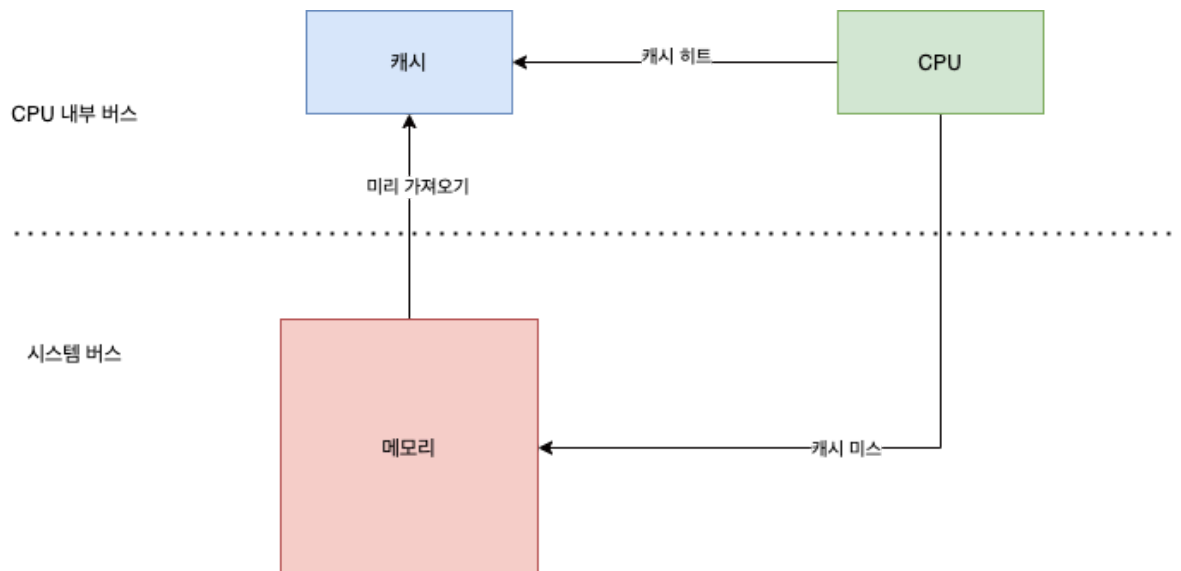


1. [레지스터] 데이터 흐름과 병목 현상

면접 질문: "CPU가 명령어를 인출(Fetch)하는 과정에서 **PC, MAR, MBR, IR** 레지스터가 순차적으로 관여합니다. 만약 시스템의 전체 속도를 높이기 위해 메모리 속도를 개선한다면, 위 레지스터들 중 어떤 레지스터와 연관된 병목(Bottleneck)이 가장 크게 해소될까요?"

- **답변 (Main Logic):** ****MBR(Memory Buffer Register)****과 연관된 병목이 가장 크게 해소됩니다. CPU 내부의 레지스터 간 이동 속도는 나노초(ns) 미만이지만, 외부 메모리(RAM)에 접근하여 데이터를 읽어오는 시간은 그보다 수십 배 이상 느립니다. MAR이 주소를 보낸 뒤 RAM으로부터 실제 데이터를 받아 MBR에 채우기까지 CPU는 '대기 상태'에 빠지는데, 메모리 속도가 빨라지면 이 데이터 수신 대기 시간이 직관적으로 줄어들기 때문입니다.
- **예상 반박 및 문제점:** "메모리 속도만 높인다고 해결될까요? CPU와 메모리를 연결하는 버스(Bus)의 대역폭이나 ****PC(Program Counter)****의 증가 속도가 낮으면 결국 전체 속도는 정체되지 않나요?"
- **보완점 (Deep Dive):** 맞는 지적입니다. 이를 보완하기 위해 현대 아키텍처는 메모리 속도 개선과 더불어 ****캐시 메모리(L1, L2)****를 CPU 내부에 두어 MBR로 들어올 데이터를 미리 인출해둡니다. 또한, 버스 병목을 줄이기 위해 데이터 버스의 폭을 넓히는 광대역 버스 설계를 병행하여 MBR의 데이터 처리 효율을 극대화합니다.



2. [인터럽트] 문맥 전환과 오버헤드의 실체

면접 질문: "인터럽트 발생 시 수행하는 '문맥 보존(**Context Save**)' 과정은 시스템 성능에 오버헤드를 발생시킵니다. 구체적으로 레지스터의 어떤 정보들이 저장되어야 하며, 하드웨어 엔지니어 입장에서 이 저장 시간을 줄이기 위해 설계적으로 어떤 대안을 제시할 수 있습니까?"

- **답변 (Main Logic): '새도 레지스터(Shadow Register)' 또는 '레지스터 뱅킹(Register Banking)'** 기술을 제안하겠습니다. 현재 사용 중인 레지스터 세트 외에 인터럽트 전용 레지스터 세트를 하드웨어적으로 추가 배치하는 방식입니다. 인터럽트 발생 시 메모리(스택)에 값을 일일이 쓰는 대신, 하드웨어 스위치만 변경하여 즉시 새로운 레지스터 세트를 사용하게 함으로써 저장 및 복구 시간을 0에 가깝게 줄일 수 있습니다.
- **예상 반박 및 문제점:** "레지스터는 CPU에서 가장 비싸고 면적을 많이 차지하는 자원입니다. 모든 인터럽트마다 새도 레지스터를 두는 것은 제조 원가 상승과 칩 크기 증가를 초래하여 비효율적이지 않나요?"
- **보완점 (Deep Dive):** 전적으로 동감합니다. 따라서 모든 레지스터가 아닌, 가장 핵심적인 **PC**, 상태 플래그, 범용 레지스터 일부에만 한정적으로 적용하거나, 실시간성이 극도로 중요한 임베디드/**RTOS** 환경의 전용 칩에 우선적으로 적용하는 식으로 '성능 대비 비용'을 최적화하는 전략이 필요합니다.

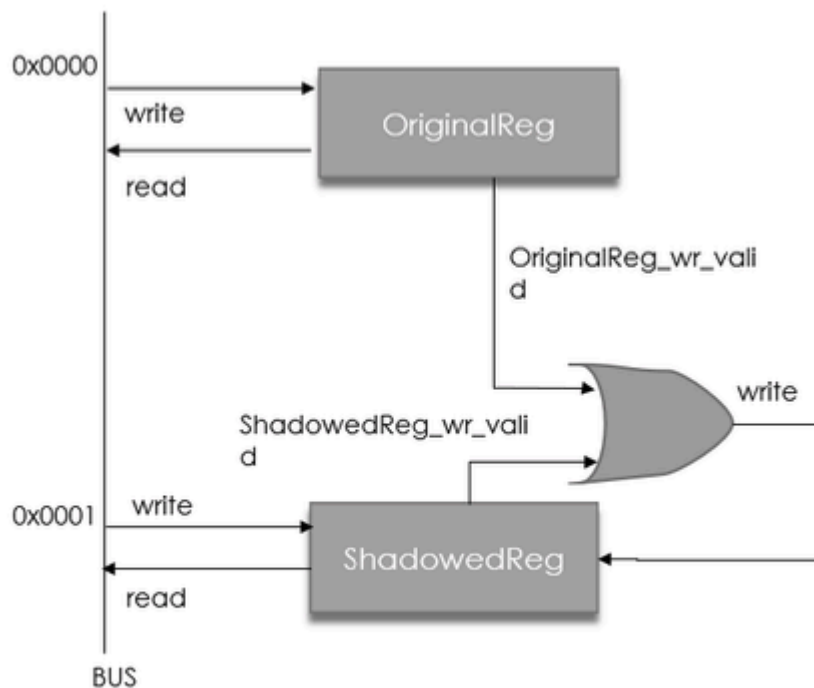


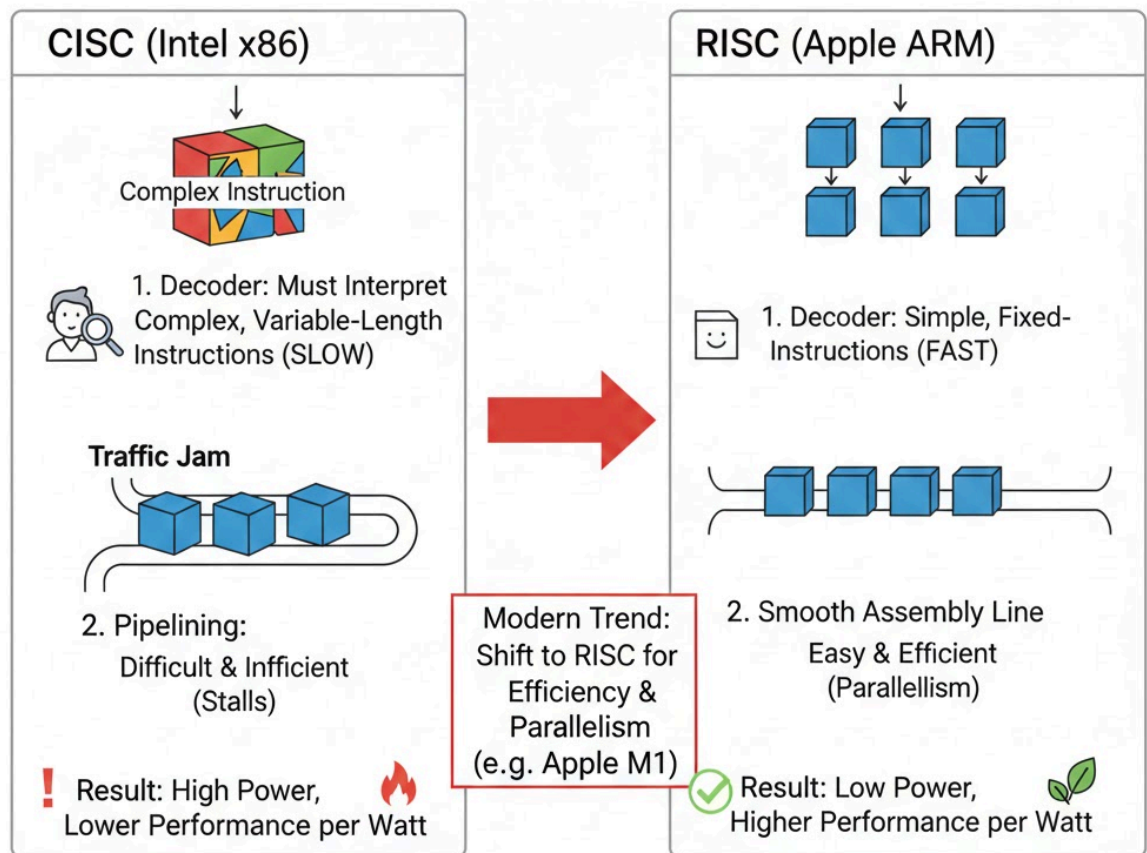
Figure 2: Shadow Register

3. [RISC vs CISC] 현대 아키텍처의 설계 효율성

면접 질문: "현대 **x86** 프로세서는 **CISC** 구조임에도 불구하고 내부적으로는 **RISC**의 이점인 파이프라이닝(**Pipelining**)을 극대화하여 사용합니다. **CISC** 명령어가 **RISC** 형태인 **u-ops(Micro-operations)**로 변환되는 과정이 성능과 전력 효율 측면에서 각각 어떤 트레이드오프(**Trade-off**)를 갖는지 설명하세요."

- **답변 (Main Logic):** 성능 면에서는 복잡한 명령어를 단순한 **u-ops**로 쪼개어 ****파이프라이닝과 비순차적 실행(OoO)****을 가능하게 하므로 처리 속도가 비약적으로 상승합니다. 하지만 전력 효율 면에서는 명령어를 실시간으로 해석하고 쪼개는 '**디코더(Decoder)**' 회로가 상시 작동해야 하므로 추가적인 전력 소모와 발열이 발생합니다.
.
- **예상 반박 및 문제점:** "최근에는 **Intel(CISC)**도 전성비가 좋아졌고, **ARM(RISC)**도 고성능화를 위해 구조가 복잡해지고 있습니다. 이제는 두 방식의 경계가 완전히 사라진 것 아닌가요?"
- **보완점 (Deep Dive):** 겉모습은 비슷해 보이지만 '태생적 한계'는 명확합니다. **x86**은 수십 년간 쌓인 가변 길이 명령어를 해석하기 위해 여전히 거대한 디코더를 포기하지 못하며, 이는 모바일처럼 극단적인 전력 효율이 필요한 분야에서 **ARM**을 이기지 못하는 결정적 이유가 됩니다. 결국 ****하위 호환성(CISC)****과 순수 효율성(**RISC**) 사이에서 어떤 가치를 우선하느냐의 선택 문제입니다.

3. RISC vs CISC: The Pipelining Trade-off



Key Point: Instruction Uniformity = Faster, More Efficient CPUS.