1. 문제: 다음 중 ALU(산술 논리 장치)와 관련된 설명으로 틀린 것은?

① ALU는 피연산자를 레지스터에서 받아 연산을 수행한다.

② ALU의 연산 결과는 항상 플래그 레지스터에 저장된다.

③ ALU는 연산 결과의 상태를 나타내기 위해 여러 가지 플래그를 사용한다.

④ 부호 플래그는 연산 결과의 부호를 나타내며, 제로 플래그는 결과가 0인지 여부를 나타낸다.

⑤ 캐리 플래그는 연산 중 올림이나 빌림이 발생했는지를 나타낸다.

2. 문제: ALU(산술 논리 장치)의 동작과 레지스터의 연관에 대한 설명으로 틀린 것은?

① ALU는 레지스터에서 피연산자를 읽어와 연산을 수행한다.

② ALU의 연산 결과는 레지스터에 저장된 후, 제어장치의 지시에 따라 메모리로 전송될 수 있다.

③ ALU는 제어장치로부터 받은 제어 신호에 따라 다양한 산술 및 논리 연산을 수행할 수 있다.

④ 레지스터는 ALU의 피연산자를 일시적으로 저장하는 역할을 하며, 여러 개의 레지스터가 함께 사용될 수 있다.

⑤ ALU는 연산을 수행하는 동안 레지스터의 값을 변경할 수 없으며, 결과적으로 새로운 값을 생성한다.

3. 문제: 레지스터와 주소 지정 방식에 대한 설명으로 틀린 것은?

① 프로그램 카운터는 다음에 실행할 명령어의 메모리 주소를 저장하고, 명령어를 읽기 전에 이 값을 증가시킨다.

② 명령어 레지스터는 메모리에서 읽어들인 명령어를 저장하며, ALU가 이 값을 기반으로 연산을 수행한다.

③ 스택 주소 지정 방식에서 스택 포인터는 스택의 꼭대기를 가리키며, 스택 프레임의 관리에 사용된다.

④ 변위 주소 지정 방식은 오퍼랜드의 값과 스택 포인터의 값을 더해 유효 주소를 계산한다.

⑤ 베이스 레지스터 주소 지정 방식은 오퍼랜드와 베이스 레지스터의 값을 더해 유효 주소를 얻으며, 데이터의 위치를 동적으로 조정할 수 있다.

4. 문제: CPU의 명령어 사이클 및 인터럽트 처리 메커니즘에 관한 다음 설명 중 틀린 것을 고르시오.

① 명령어 사이클은 인출(Fetch) 사이클, 실행(Execution) 사이클, 그리고 필요에 따라 간접(Indirect) 사이클을 포함하며, CPU는 이들 사이클을 반복 수행하여 프로그램의 명령어를 처리한다.

② 인출 사이클 동안, CPU는 메모리에서 명령어를 읽어 명령어 레지스터(IR)에 저장하고, 동시에 프로그램 카운터(PC)를 자동 증가시켜 다음 명령어의 주소를 준비한다.

③ 간접 사이클은 오퍼랜드 필드에 저장된 값이 항상 유효주소의 주소인 경우에만 수행되며, 모든 명령어에서 반드시 실행되는 단계이다.

④ 비동기 인터럽트는 주로 입출력 장치 등 외부 요인에 의해 발생하며, CPU는 실행 사이클이 종료된 후 인터럽트 플래그를 확인하고, 허용 상태일 경우 현재 작업 상태를 스택에 백업한 후 인터럽트 벡터에 따라 인터럽트 서비스 루틴(ISR)을 실행한다.

⑤ 동기 인터럽트는 CPU 내부에서 발생하는 예외 상황에 대응하기 위한 것으로, 폴트, 트랩, 중단, 소프트웨어 인터럽트 등 여러 유형으로 구분되며, 인터럽트 처리 후 원래의 실행 흐름으로 복귀된다.

5. 문제: 다음 중 CPU가 예외 상황에 직면했을 때의 처리 방식과 실제 발생 예시를 올바르게 연결한 것은 무엇인가?

① 페이지 폴트 (Fault): 프로그램이 메모리에서 요구한 페이지가 존재하지 않을 경우, CPU는 페이지 폴트 예외를 발생시킨다. 예외 처리 루틴이 필요한 페이지를 메모리에 로드한 후, 예외가 발생한 동일한 명령어부터 재실행한다.

② 브레이크 포인트 (Trap): 디버깅 과정에서 사용자가 브레이크 포인트를 설정하면, CPU는 트랩 예외를 발생시켜 예외 처리 후 해당 명령어의 다음 명령어부터 실행을 재개한다.

③ 접근 위반 (Abort): 프로그램이 존재하지 않거나 접근 권한이 없는 메모리 영역에 접근할 경우, CPU는 접근 위반 예외를 발생시키며, 심각한 오류로 판단되어 예외 처리 후 프로그램을 강제 종료시킨다.

④ 산술 연산 – 0으로 나누기 (Fault): 프로그램이 산술 연산 중 0으로 나누기를 시도하면, CPU는 폴트 예외를 발생시켜 예외 처리 후 같은 명령어를 다시 실행하여 정상 처리를 시도한다.

⑤ 산술 연산 – 오버플로우 (Trap): 연산 결과가 표현 범위를 초과할 경우, CPU는 오버플로우 트랩 예외를 발생시켜, 예외 처리 후 해당 명령어의 다음 명령어부터 실행을 재개한다.

1. 정답: ② ALU의 연산 결과는 항상 플래그 레지스터에 저장된다.

해설: ALU의 연산 결과는 일반적으로 레지스터에 저장되며, 결과의 상태를 나타내기 위해 플래그 레지스터에 플래그가 설정됩니다. 따라서 ALU의 연산 결과가 직접 플래그 레지스터에 저장된다고 하는 것은 틀린 설명입니다. 나머지 선택지는 ALU의 기능 및 플래그의 역할에 대한 정확한 설명입니다.

2. 정답: ⑤ ALU는 연산을 수행하는 동안 레지스터의 값을 변경할 수 없으며, 결과적으로 새로운 값을 생성한다.

해설: ALU는 연산을 수행하는 동안 레지스터의 값을 읽어와서 연산을 진행하고, 그 결과를 다시 레지스터에 저장할 수 있습니다. 따라서 ALU가 연산을 수행하는 동안 레지스터의 값을 변경할 수 없다는 설명은 틀립니다. 나머지 선택지는 ALU의 동작 및 레지스터의 역할에 대한 정확한 설명입니다.

3. 정답: ④ 변위 주소 지정 방식은 오퍼랜드의 값과 스택 포인터의 값을 더해 유효 주소를 계산한다.

해설: 변위 주소 지정 방식은 오퍼랜드의 값과 특정 레지스터(일반적으로 베이스 레지스터 또는 인덱스 레지스터)의 값을 더해 유효 주소를 계산합니다. 스택 포인터는 스택 주소 지정 방식에서 사용되며, 변위 주소 지정 방식에서는 주로 다른 레지스터가 사용됩니다. 나머지 선택지는 레지스터와 주소 지정 방식에 대한 올바른 설명입니다.

4. 정답: ③

해설: 간접 사이클은 간접 주소 지정 방식을 사용하는 명령어에 대해 추가적인 메모리 접근이 필요한 경우에만 수행됩니다. 즉, 모든 명령어가 간접 사이클을 거치는 것이 아니라, 오퍼랜드 필드에 실제 데이터가 아닌 유효주소의 주소가 저장된 경우에 한정됩니다. 따라서 ③번의 "모든 명령어에서 반드시 실행되는 단계"라는 설명은 틀렸습니다.

5. 정답: ④

해설: 0으로 나누기 오류는 회복 가능한 상황이 아니라, 조건이 변하지 않는 한 동일한 명령어를 재실행해도 계속 오류가 발생한다. 일반적으로 산술 연산에서 0으로 나누는 상황은 치명적인 오류로 간주되어, 예외 처리 후 해당 명령어를 재실행하는 폴트 방식으로 처리되지 않고 프로그램을 중단(Abort)하거나, 별도의 처리 루틴에서 오류를 보고하는 방식(트랩)으로 처리하는 것이 적절하다. 따라서 ④번의 설명은 올바르지 않습니다.