마이크로프로세서 설계 무작정 따라하기 part-II (1)

부제: 명령어 시뮬레이터, 어셈블러, 컴파일러의 개발

KAIST 전자전산학과 박사과정 배영돈(donny@ics.kaist.ac.kr), 이종열(jylee@ics.kaist.ac.kr)

지난 4 회 동안 마이크로프로세서 설계방법에 대하여 연재하였다. 본 강좌는 그 연장선 상에서 개발환경을 제작하는 방법에 대하여 설명하고자 한다. 지난 강좌에서 설계한 SimpleCore 의 명령어 시뮬레이터, 어셈블러, 컴파일러를 개발하여 SimpleCore 를 실제 시스템에 적용할 수 있는 수준으로 발전시키고, 이를 통해 효과적으로 개발 환경을 제작하는 방법을 설명하고자 한다.

1. 마이크로프로세서 개발 환경(developer's environment)

마이크로프로세서를 이용하여 복잡한 시스템의 설계를 하기 위해선 빠른 시뮬레이터가 필요하며, 성능을 최대화하기 위해서는 컴파일러의 역할이 매우 중요하다. 성능 좋은 마이크로프로세서를 개발 하더라도 편리하고 효율적인 개발 환경이 갖추어져 있지 않다면 경쟁력을 얻기 매우 어렵다. 따라서, 개발환경은 마이크로프로세서의 개발에 필수적인 부분이라고 할 수 있다.

가) 명령어 시뮬레이터 (Instruction Set Simulator)

지난 강좌에서 테스트프로그램을 Verilog 시뮬레이터를 이용하여 실행해 보았다. Verilog 시뮬레이터는 회로의 동작을 확인하기 위한 것이며 복잡하고 긴 프로그램을 실행하기에는 속도가 느리다. 따라서, 개발된 마이크로프로세서를 사용하여 응용 프로그램을 개발하고 검증하기 위해서는 명령어 수준에서 빠르게 실행 할 수 있는 명령어 시뮬레이터가 필요하다.

명령어 시뮬레이터는 응용프로그램 개발자 뿐만 아니라 마이크로프로세서 설계자에게도 매우 유용하다. 설계자가 구체적인 설계 이전에 기본적인 구조를 설계하고 이를 검증할 때 사용할 수 있으며, 각 명령어의 복잡한 동작의 참조 모델(reference model)로 회로의 동작을 검증할 때 매우 유용하게 사용할 수 있다.

나) 컴파일러 (Compiler)

컴파일러는 C 나 C++로 개발된 프로그램을 특정 마이크로프로세서에 최적화하는 역할을 한다. 컴파일러의 성능에 따라 적은 메모리 용량을 사용하면서도 빨리 수행되는 프로그램이 될 수도 있으며, 그 반대가 될 수도 있다. 컴파일러의 개발은 매우 어렵고 많은 시간을 필요로 한다. 따라서, 본 강좌에서는 GNU C Compiler(gcc)를 SimpleCore 에 포팅(porting)하고자 한다. gcc 는 비교적 쉽게 다양한 마이크로프로세서에 적용할 수 있게 만들어져 있으며 성능도 상용 컴파일러에 비하여 뒤지지 않는다.

다) 어셈블러 (Assembler)

컴파일러에서 생성된 assembly code 나 사용자가 직접 작성한 것을 마이크로프로세서에서 사용하는 object code 로 만들어주는 기능을 한다. 지난강좌에서 테스트프로그램의 바이너리 값을 모두 수작업으로 생성하였으나 어셈블러를 사용하면 이러한 과정을 자동화 할 수 있게 된다.
어셈블러 역시 GNU Assembler(gas)를 사용하면 적은 노력으로 좋은 어셈블러를 개발할 수 있다.

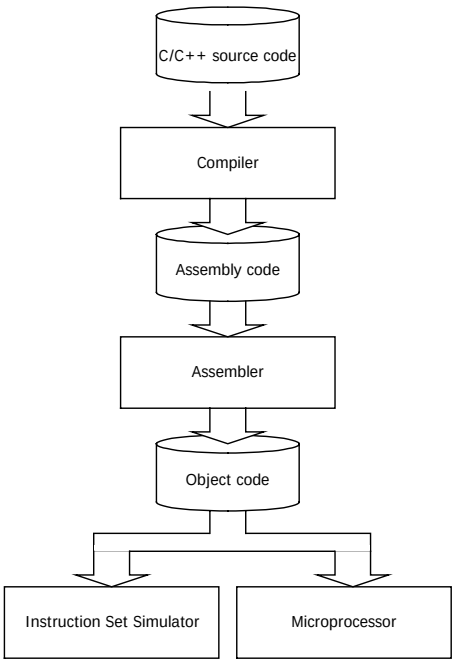

그림 1. 개발 환경

2. 명령어 시뮬레이터

명령어 시뮬레이터는 일반적으로 C 언어를 이용하여 대상 프로세서의 동작을 모방하여 만든 것이다. 간단히 설명하면 그림 2 와 같이 각 명령어에 대한 동작을 기술하면 된다. 이 때 결정해야 할 것은 얼마나 마이크로프로세서의 구조를 자세히 기술 할 것인가 이다. 그림 2 의 경우가 가장 간단한 형태라고 볼 수 있으며, 반대로 자세한 경우에는 파이프라인구조를 고려한 형태를 예로 들 수 있다. 이러한 시뮬레이터를 Cycle-accurate Simulator 라고 부른다. 이것은 상대적으로 제작하기가 어렵고 실행 속도도 느린 대신 보다 세밀한 동작을 시뮬레이션 할 수 있으며 프로세서의 구조와 관련된 시뮬레이션(architecture simulation)을 할 수 있으므로, 회로 설계에 도움이 된다. 처음에는 가장 간단한 형태의 시뮬레이터를 개발하는 것이 바람직하다. 프로세서의 구조를 반영할수록 시뮬레이터가 복잡해지고 그만큼 잘못된 동작을 할 가능성이 커진다. 명령어 시뮬레이터는 무엇보다 정확한 동작이 중요하다고 할 수 있다. 우선 명령어의 동작이 제대로 기술 되었는지 확인하고 단계별로 프로세서의 구조를 반영해가는 것이 좋다.
특히, 호환프로세서를 설계할 때는 명령어 시뮬레이터를 통해 명령어의 동작(호환성)을 정확히 검증하는 것이 중요하다.

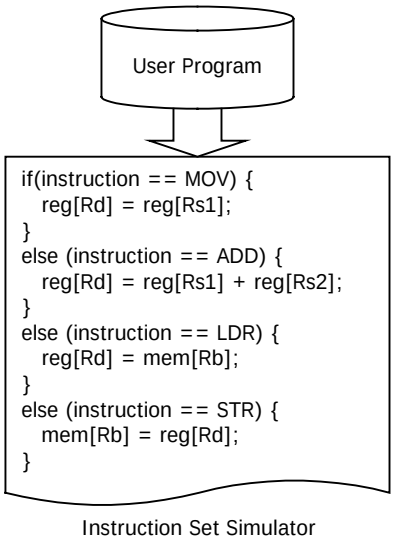

그림 2. 명령어 시뮬레이터의 동작 개요

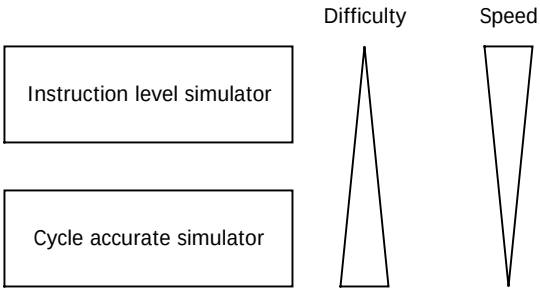

그림 3. 명령어 시뮬레이터의 종류

앞서 설명한 것과 같이 명령어 시뮬레이터는 응용프로그램을 개발 할 때 사용하는 것 이외에 마이크로프로세서의 설계 과정에서도 중요한 역할을 한다. 프로세서를 설계하고 시뮬레이션을 할 때, 복잡하고 긴 프로그램의 수행결과를 눈으로 확인하는 것은 매우 어렵다. 프로그램 수행 중에 발생한 오류를 찾으려면 수많은 파형을 관찰해야 하며 그 값이 맞는지 따져봐야 하기 때문이다. 따라서, 정확한 동작을 보장하는 명령어 시뮬레이터의 결과와 비교하면 그 문제를 쉽게 찾을 수 있으며, 이 비교과정을 자동화하면 매우 효율적으로 검증을 할 수 있게 된다. Verilog 시뮬레이터가 제공하는 program language interface(PLI)기능을 사용하면 사용자가 만든 프로그램과 Verilog 시뮬레이터를 연결 할 수 있다.

또한, 명령어 시뮬레이터에는 사용자 인터페이스(user interface)를 고려해야 한다. 즉, 사용자의 편의를 위해 여러 명령어를 사용하여 제어할 수 있어야 한다. 즉, 명령어 하나씩 진행하는 명령(step), 여러 개의 명령어를 진행하는 명령, 레지스터나 메모리 영역의 내용을 출력해주는 명령, 지정된 메모리 영역을 파일로 쓰거나 파일 내용을 메모리에 저장하는 명령, 특정 레지스터나 메모리의 내용이 변경되면 정지하는 기능, disassembly 기능, 등이 필요하며 그리고 최종적으로는 graphic user interface 까지 제공하는 것이 바람직하다.