창립 50주년기념 특집 - 컴퓨터기술

마이크로 프로세서의 발달

박 명 순*
(*고려대 이과대 컴퓨터학과 교수)

1. 서 론

컴퓨터는 중앙에서 연산을 처리하는 마이크로 프로세서와 연산 처리를 위한 코드와 데이터를 저장하는 메모리 그리고 코드와 데이터를 입출력하는 주변 장치등으로 구성되어 있다. 이러한 컴퓨터의 처리 능력을 높이는 가장 효과적이고 현재까지 가장 많은 연구가 이루어져 온 부분이 마이크로 프로세서이다.

그동안 다양한 형태의 마이크로 프로세서들이 제안되었고, 사라져 갔다. 가장 초기의 마이크로 프로세서는 주어진 연산들을 hardwire된 로직으로 표현하는 방식으로 제작되었다. 그러나 이러한 방식은 프로세서의 복잡성을 높이고, 명령어의 추가가 어려운 단점을 가지고 있었다. 이러한 문제를 해결하기 위해 등장한 설계 방식이 CISC이다[3]. CISC 프로세서는 프로세서 내부에 마이크로 메모리와 마이크로 코드, 이를 처리하기 위한 처리 엔진을 가지고 있어서 새로운 명령어 추가시 적절한 마이크로 코드의 조합만을 추가하면 쉽게 확장할 수 있는 장점이 있다. 특히 CISC는 다양한 고수준의 명령어들을 제공하여 어셈블리 프로그램을 쉽게 작성할 수 있는 장점을 가진다. 그러나 많은 프로그램들이 C, FORTRAN등과 같은 고수준의 프로그램 언어(high-level programming language)로 작성되고, CISC를 가능하게 했던 메모리와 프로세서의 속도차이가 줄어듦에 따라 프로세서 자체의 속도를 높이기 위해 RISC가 제안되었다[3]. RISC는 기본적인 명령어들을 매우 빠르게 수행하도록 설계되었으며, 이를 위해 파이프라이닝이나 수퍼스칼라와 같은 다양한 성능향상 기법이 제안되었다. 특히 수퍼 스칼라는 동시에 다수의 명령어를 처리할 수 있으며, 현재 대다수의 프로세서에서 채용되고 있다[1, 3]. 그러나 수퍼 스칼라 프로세서에서 수행하는 명령어 스케줄링의 제한 및 부담등으로 인하여 90년대 들어서 VLIW라는 새로운 프로세서가 출현하게 되었다[2, 4].

본 글에서는 CISC, RISC, 수퍼스칼라, VLIW를 소개하고 각각의 프로세서가 출현하게된 배경과 특징, 장단점 등을 다룬다.

2. CISC

마이크로 프로세서의 성능 향상을 위한 노력은 다양한 마이크로 프로세서 구조로 나타나게 되었다. 이러한 노력의 결과 첫 번째로 나타난 마이크로 프로세서의 형태가 CISC이다. CISC 이전의 마이크로 프로세서는 주어진 명령에 대한 처리 과정을 하나의 hardwire된 로직으로 설계하였다. 그러나 이러한 경우 다양한 명령들에 대하여 일일이 설계하여야 하는 어려움이 있었으며 따라서 프로세서의 제작이 대단히 비효율적이었다. 이러한 어려움을 극복하고 보다 효과적으로 마이크로 프로세서를 설계하기 위해서 제안된 방식이 CISC이다[3].

CISC의 가장 큰 특징은 프로세서내에 또 다른 기억장소가 존재하여 프로세서 외부에서 주어진 명령어를 프로세서 내부에 저장된 정보에 따라 일련의 작은 여러 명령어, 즉 마이크로 코드들로 대치하여 수행하는 방식을 취한다는 것이다.

특히 프로세서내의 메모리는 외부 메모리 보다 10배 이상 빠른 접근 시간을 보이기 때문에 CISC 설계자들은 가능하면 많은 명령어들을 마이크로 코드로 제작하려고 하였다. 이때 마이크로 코드는 자주 사용됨에도 불구하고 느린 수행 속도를 보이는 명령어들을 위주로 선택되었다. 이밖에도 마이크로 코드를 사용하여 프로세서를 설계하는 것은 다음

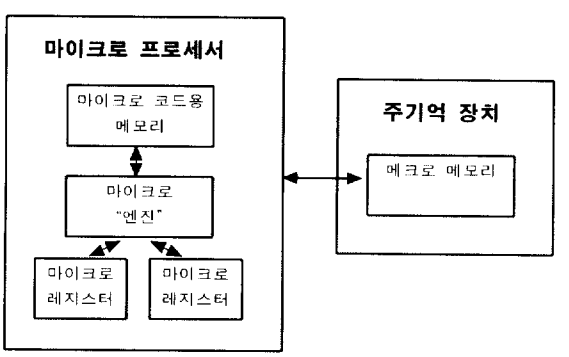

그림 1. CISC 프로세서의 구성

과 같은 많은 장점을 가지고 있다.

- 마이크로코드 메모리는 일반 메모리에 비해 빠른 접근 속도를 보이므로, 많은 명령어들이 순수하게 hirdwire된 구현에 비해 느리지 않게 마이크로코드로 구현될 수 있었다.
- hirdwire 방식에 비하여 마이크로 코드 방식은 작은 트랜스레이터를 요구하며, 쉬운 제작환경을 제공한다.
- 새로운 명령어 세트를 지원하도록 수정하기가 용이하다.

이러한 장점을 활용하여 IBM 360 시리즈는 같은 프로그래밍 모델을 서로 다른 하드웨어 상에 구축하였다. 또한 같은 명령어 세트임에도 불구하고 사용 목적에 따라 과학 계산용 명령어들 만을 혹은 상업 계산용 명령어들 만을 최적화하여 구현하여, 특정 목적에 빠른 속도를 보이는 머신이 다른 목적의 프로그램에 대한 호환성도 확보할 수 있도록 설계하기도 하였다.

CISC의 또 다른 특징은 복잡한 기능이 하나의 명령어로 표현되며 점점 프로그래머에게 친숙한 명령어 세트를 지원하였다. 이러한 특징은 프로그램의 크기를 줄이고 어셈블리 프로그램의 작성을 용이하게 만들었다.

이와 같은 특징을 가지는 CISC는 다음과 같은 장점을 갖는다.

- hardwiring 기법에 비해 확장이 용이하다.
- 어셈블리 언어 프로그래밍이 쉽다.
- 과거 머신과 호환성을 갖춘 머신을 설계하기가 쉽다.
- 주어진 작업을 완수하는데 적은 수의 명령어가 필요하다.
- 상위 수준 언어에서 사용할 수 있는 명령어가 존재하므로 컴파일러가 단순하다.

반면 컴퓨터의 CISC는 제작 환경이 변함에 따라 아래와 같은 문제점을 노출하였다.

- 새로운 머신이 나옴에 따라 확장된 명령어 세트를 처리하기 위한 H/W가 점점 복잡해지기 시작했다.
- 명령어들의 다양한 수행 시간이 시스템의 성능을 떨어뜨린다.
- 많은 특수 목적 명령어들은 거의 사용되지 않는다. 일반적인 프로그램은 약 20% 정도의 명령어만을 사용한다.

3. RISC

1970년대 수퍼컨덕터의 기술이 발달함에 따라 메모리와 프로세서간의 속도 차이가 줄어들게 되었다. 메모리 접근 속도의 향상으로 인해 많은 프로그램이 고수준 프로그래밍 언어로 제작되게 되었고, CISC의 많은 장점들 예를 들어 어셈블리 프로그래밍의 용이함, 작은 프로그램 크기 등으로 인한 유리한 점이 사라지게 되었으며, 따라서 프로세서 설계자는 프로세서 자체의 성능 향상에 주목하게 되었다[3].

RISC(Reduced Instaruction Set Computers)를 출현하게 된 기본적인 생각은 복잡한 기능을 하는 명령어는 일련의 단순한 명령어들로 대치될 수 있으며, 이렇게 단순화 시킨 명령어들로 인하여 프로세서의 구조가 단순해 질 수 있다는 것이다. 이러한 RISC의 특징은 다음과 같다.

- 기본적이고 단순한 명령어 세트만을 지원한다.
- 모든 명령어의 길이는 동일하다.
  일정한 명령어 길이는 모든 명령어를 한 번의 fetch 와 decode로 인식할 수 있도록 만든다.
- 모든 명령어는 한 싸이클에 처리되도록 한다.
  모든 명령어를 단일 싸이클에 처리하도록 하는 특징은 프로세서가 동시에 여러 명령어를 처리할 수 있도록 하였다. 모든 명령어를 단일 싸이클에 처리하도록 하는 데에는 파이프라이닝(pipelining) 기술이 사용되었다.

이와 같은 특징을 만족시키고 보다 고성능의 프로세서를 구현하기 위해 여러 가지 기술들이 개발 적용되었다.

파이프라이닝은 RISC 프로세서를 구현하는데 핵심이 되는 기술이다. 파이프라이닝에 의하면 동시에 여러개의 명령어가 수행된다. 즉 한 명령어가 끝나기 전에 다음 명령어의 수행을 시작한다. 프로세서내의 명령어는 일반적으로 fetch, decode, execute, write의 네단계로 이루어진다. 파이프라이닝은 아래의 그림과 같이 한 명령어가 decode되고 있는 사이에 그 다음 명령어를 fetch하는 식으로 동시에 4개의 명령어를 수행하게 된다. 이때 각 단계는 한 싸이클이 소요된다. 따라서 평균 매 싸이클 마다 하나의 명령어가 실행되도록 하고 있다.

시간 --->

ADD R1, 3, 4				W	E	D	F
SUB R2, 4, 2			W	E	D	F	
MUL R3, 5, 6		W	E	D	F		
DIV R4, 10, 5	W	E	D	F			

하지만 모든 명령어가 이와 같이 이상적인 수행 형태를 보이는 것은 아니다. 예를 들어 메모리를 접근하는 명령어나 종속성이 존재하는 명령어들은 한 싸이클에 수행완료될 수 없는 경우가 발생한다. 이처럼 명령어가 한 싸이클에 완료되지 못하면 그 다음 명령어들은 현재의 상태에서 앞의 명령어가 수행되기를 기다려야 한다. 이 상황을 stall이라고 한다. 이러한 stall 상황을 해결하고자 다음과 같은 노력이 이루어져 왔다.

메모리 접근 속도는 일반적으로 프로세서의 속도에 비해 수배내지 수십배 이상의 시간이 소요된다. 따라서 메모리

접근을 하는 명령어가 수행될 경우 프로세서는 한동안 stall 되는 현상이 발생한다. 이러한 현상은 캐쉬를 사용하여 해결하고 있다. 캐쉬는 빠른 접근 속도를 보장하고 있으며, 특히 프로세서 내에 장착된 on-chip 캐쉬의 경우 한 싸이클에 접근을 완료할 수 있어서, 만일 접근하는 데이터(코드)가 on-chip 캐쉬에 저장될 경우 stall현상이 발생하지 않고 메모리 접근 명령어를 처리할 수 있다.

명령어간의 종속성은 종속된 두 명령어 사이에 수행 순서가 지켜져야 프로그램의 의미 변화가 없음을 나타낸다. 즉 다음과 같은 프로그램에서

```
ADD     R1, 3, 4        # R1 = 3+4
SUB     R2, R1, 4       # R2 = R1-4
```

두 번째 문장은 첫 번째 문장이 수행 완료되어 R1에 연산 결과를 쓰는 write단계 이후에야 수행 가능하다. 이러한 프로그램을 파이프라인으로 수행하면 다음과 같이 파이프라인의 효과를 전혀 거둘 수 없는 결과를 야기한다.

ADD R1, 3, 4
SUB R2, R1, 4

				W	E	D	F
W	E	D	F				

이러한 상황을 해결하기 위해서는 컴파일러가 이러한 상황이 발생하지 않도록 명령어들 사이의 종속성 관계를 파악 관련있는 명령어들을 멀리 떨어뜨리는 방법이 사용된다. 즉 아래와 같이 종속성이 존재하는 두 명령 사이에 종속성이 존재하지 않는 다른 명령들을 삽입함으로써 파이프라인에 stall현상이 발생하는 것을 미연에 방지한다.

```
ADD     R1, 3, 4        # R1 = 3+4
MUL     R3, 5, 6
DIV     R4, 10, 5
SUB     R2, R1, 4       # R2 = R1-4
```

이와 같이 RISC에서는 프로세서의 성능을 높여서 한 싸이클에 한 명령어가 수행되는데 주력하였다. 그러나 프로세서 설계자들은 이에 만족하지 않고 한 싸이클에 두 개 이상의 명령어를 수행하는 기법들을 개발하였다. 그중 하나는 수퍼 파이프라이닝이다. 이 기법은 외부 클락보다 2배 빠른 프로세서 내부 클락을 사용하여 파이프라인의 각 단계를 다시 2개의 소단계로 나누어 총 8단계로 처리함으로써 명령어 처리 속도를 2배 향상시키는 기법이다. 그러나 이 기법은 파이프라인을 stall시키는 것과 같은 원인들로 인해 2배의 처리속도를 보장하지는 못하고 있다.

또 다른 기법은 수퍼 스칼라이다. 이 기법은 명령어 처리 모듈을 두 개로 늘림으로써 처리속도의 향상을 기대한다.
이러한 RISC는 CISC에 비하여 다음과 같은 장점을 갖는다.

- 단순해진 명령어가 파이프라이닝과 수퍼스칼라를 적용가능하게 하기 때문에, RISC는 CISC에 비하여 약 2배에서 4배까지의 성능 향상을 얻을 수 있다.
- 단순해진 하드웨어로 인하여 칩의 영역을 절약할 수 있으며, 절약된 부분을 매모리 영역이나, 실수 연산 모듈로 활용함으로써 성능 향상을 꾀할 수 있다.
- 단순해진 구조로 인하여 CISC에 비하여 설계하기가 용이하다.

반면 RISC는 성능 향상을 위해 코드의 스케쥴링이 반드시 필요하며, 이로 인해 여러 가지 문제를 야기한다. 다음은 RISC를 사용하는데 반드시 고려해야하는 부분이다.

첫째, RISC에서는 파이프라이닝 단계에서 stall이 발생하지 않도록 하기위해 컴파일러의 도움이 절대적으로 필요하다.

둘째, 파이프라이닝과 명령어 스케쥴링 때문에 프로그램의 디버깅이 어렵다.

셋째, CISC를 위한 코드 보다 RISC를 위한 코드는 더 큰 메모리를 요구한다.

넷째, 단일 싸이클에 메모리 접근을 완료하기 위해서는 커다란 캐쉬 메모리가 필수적이다.

4. 수퍼스칼라

수퍼스칼라는 RISC 프로세서의 성능 향상을 위하여 제안되었다. 수퍼스칼라의 기본적인 생각은 다수의 연산 유닛을 두고 병렬 수행 가능한 명령어들을 최대한 병렬 수행함으로써 수행 속도를 높이는데 있다[1, 3].

이를 위해서는 순차 수행을 위해 정렬된 일련의 명령어들을 병렬 수행 가능한 부분과 그렇지 못한 부분으로 분리하는 작업이 필요하다. 일련의 명령어들중 반드시 순차 수행되어야 하는 부분은 기본적으로 명령어들간의 종속성으로 표현된다. 따라서 서로 종속되지 않은 명령어들을 동시에 수행함으로써 성능 향상을 꾀한다.

동시에 수행하도록 명령어들을 재배열하는 방법에는 크게 3가지 방식이 있다. 첫 번째 방식은 "in-order issues and in-order completion" 방식으로 수행 명령어의 순서는 바꾸지 않고 단지 인접된 명령어가 종속되지 않을 경우 이들을 동시에 수행하는 방식이다. 두 번째는 "in-order issues and out-of-order completion"으로 fetch와 decode는 순서대로 하되, 실행시 수행 시간이나 명령어간의 종속성 때문에 늦게 수행되는 것은 제외하고 나머지 명령어들만을 먼저 수행하는 방식이다. 이럴 경우 write되는 명령의 순서는 원래의 순서와 바뀌게 된다. 마지막은 "out-of-order issues and out-of-order completion"으로 일정 갯수의 명령어들-이 범위를 윈도우라 부른다-을 실행 이전에 미리 fetch하여 이들의 종속성 관계를 고려하여 병렬 수행 가능한 명령어들을 스케쥴링하여 실행하는 방식으로 실행 자체의 순서가 바뀌어 버린다. 이 방식은 가장 일반적인 방식으로 윈도우의

범위가 크면 클수록 병렬로 수행될 수 있는 명령어의 수가 늘어나게 된다. 가장 이상적으로는 프로그램 전체가 윈도우 안에 들어갈 경우 모든 병렬성을 추출할 수 있다.

이와 같이 수퍼스칼라에서는 프로세서가 실행되는 명령어들의 순서를 스케쥴링하여 병렬로 수행될 명령어들을 추출한다. 그러나 이러한 스케줄링을 위한 부분의 복잡성은 프로세서 설계의 어려움을 가중시킬 뿐만 아니라 윈도우 크기 또한 제한되어 많은 병렬성을 추출하지 못하는 문제가 있다.

5. VLIW

이러한 수퍼스칼라의 문제를 해결하고 프로그램내에 있는 모든 병렬성을 추출하기 위한 목적으로 제작된 머신이 VLIW(Very Long Instruction Word)이다. VLIW는 명령어 스케쥴링 과정을 컴파일러에게 일임함으로써 프로세서의 복잡성을 크게 줄이면서도 프로그램내의 모든 명령어 수준 병렬성을 추출하여 병렬로 수행할 수 있게 하여 성능 향상을 이루도록 한다[2, 4].

VLIW 프로세서의 기본 구조는 그림 2와 같다. VLIW 프로세서의 구조에서 다른 프로세서들과 다른 가장 큰 특징은 명령어(instruction word)가 하나의 기능만을 하는 것이 아니라 여러 기능을 하는 단위 명령들의 집합으로 이루어진다는 것이다. 따라서 명령어의 길이가 다른 프로세서 보다 길다. 또한 이러한 단위 명령들은 다수의 명령어 처리 유닛들(IU 또는 FU)에 의해 병렬로 처리된다.

이러한 구조의 프로세서에서는 컴파일러의 역할이 더욱 중요하다. 컴파일러는 프로그램을 분석하여 병렬로 수행될 수 있는 기본 명령어들을 하나의 긴 명령어로 재구성하는 작업을 수행해야 한다. 이를 위해서 기존의 수퍼스칼라 프로세서에서 프로세서 내부에서 수행되었던 명령어 스케쥴링 작업을 이제는 컴파일러에서 수행해야 한다. 이에 따라 수퍼 스칼라에서 문제가 되었던 윈도우 크기의 제한이 사라지면서, 프로그램 전체를 대상으로 병렬성을 추출할 수 있으며 보다 정교한 병렬성 추출 방법을 적용할 수 있게 되었다.

반면, VLIW는 대단히 정교하고 복잡한 컴파일러를 요구한다. VLIW 프로세서의 성능을 좌우하는 것은 컴파일러에서 생성되는 코드의 품질에 달려 있다. 이에 따라 VLIW

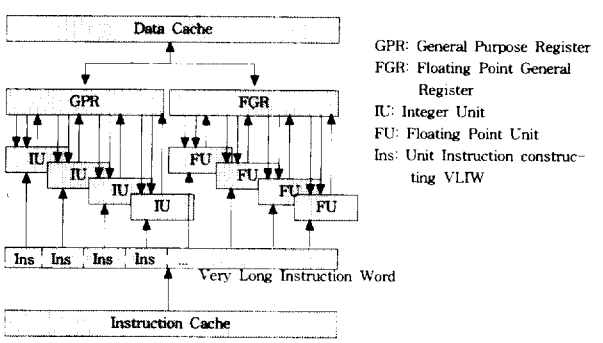

그림 2. VLIW 프로세서의 구조

컴파일러는 다양한 최적화 및 병렬화 알고리즘을 동원하여 기계어 수준의 병렬성을 추출하는데 주력하고 있다.

컴파일러가 보다 높은 품질의 코드를 생성하기 위해서 컴파일러는 컴파일을 수행하는 시점에서 머신에서 실행 시점에 발생하는 모든 상황을 예측하여야 한다. 따라서 컴파일러에서 생성되는 코드는 목적 머신에 종속적이되고 머신간의 코드 호환성이 떨어지게 된다.

6. 결 론

본 글에서는 컴퓨터의 성능을 좌우하는 가장 중요한 요소인 프로세서의 변천 과정에 따른 다양한 프로세서들에 대하여 알아보았다.

이러한 다양한 형태의 프로세서중 특정 프로세서 형태가 가장 좋다고 말할 수는 없다. 예를 들어 RISC는 CISC의 단점을 극복한 것이지만 아직까지도 많은 프로세서들은 CISC의 특징을 가지고 있다. 가장 좋은 프로세서의 형태는 주어진 환경과 시스템의 사용 목적 등에 따라 바뀔 수 있으며, 또한 현재의 프로세서 성능 향상을 유지하기 위해서는 계속적으로 새로운 프로세서의 형태가 연구되고 제안되어야 한다.

참 고 문 헌

[1] Mike Johnson, *Superscalar Microprocessor Design*, Prentice Hall, 1990.

[2] B. Ramakrishna Rau and Joseph A. Fisher, "Instruction-Level Parallel Processing: History, Overview, and Perspective," *The Journal of Supercomputing*, pp.9-50, 1993.

[3] Gabriel Acosta-Lopez, Richard Clark, and Anne Wysocki, "Introduction to RISC Technology," *http://kandor.isi.edu/aliases/PowerPC_Programming_Info/intro_to_risc/irt3_technology0.html*, 1997

[4] 박명순, 김석일, 전문석, "VLIW 병렬 컴퓨터의 개발과 디지털 선호 처리에의 응용," 2차년도 중간 보고서, 한국과학재단, 1996.

저 자 소 개

박명순(朴明淳)

1952년 4월 7일생. 1975년 서울대 공대 전자공학과 졸업. 1982년 Univ. of Utah 전기공학과 졸업(석사). 1985년 Univ. of Iowa 전기전산공학과 졸업(공박). 1975년-80년 국방과학연구소 연구원. 1985년-87년 Marquette 대학교 조교수. 1987년-88년 포항공대 전자전기공학과 조교수. 1988년-현재 고려대 이과대 컴퓨터학과 교수.