RISC Architecture

전 주 식
(서울대 공대 전자계산기공학과 조교수)

1. 서 론

하드웨어의 가격이 하락하고 소프트웨어의 가격이 상대적으로 상승함에 따라, 컴퓨터 시스템에서 소프트웨어가 차지하는 비용이 큰 비중을 차지하게 되었다.

이러한 경향에 따라 강력하고 복잡한 기능을 제공하는 고급 프로그래밍 언어에 대한 필요성이 대두되었고, 프로그래머들은 이러한 언어를 사용하여 간결하게 알고리즘을 표현할 수 있었다. 그러나 고급 프로그래밍 언어에서 제공되는 연산들과 컴퓨터 구조에서 제공되는 명령어(instruction)들 사이의 의미상 간격(semantic gap)은 점점 더 벌어져 왔다. 이 간격은 기계어(machine language) 프로그램 크기를 증가시켜, 프로그램 수행 효율을 저하시켰고, 더욱 복잡한 컴파일러(compiler)를 구성하도록 요구해 왔다. 따라서 컴퓨터 시스템 설계자들은 이런 의미상 간격을 메우기 위해 명령어의 의미와 주소지정 방법(addressing mode)이 다양한 명령어 집합(instruction set)을 채택하였으며, 이러한 프로세서(processor)는 CISC(Complex Instruction Set Computer)라고 불리운다.

그러나 컴퓨터 시스템이 많은 명령어들을 처리하기 위해서는, 프로세서의 제어부(Control Unit) 구조가 복잡해질 뿐만 아니라, 기본 명령어 수행 시간이 증가하는 부담이 생기므로, 기계어 프로그램에 대한 수행 성능이 저하된다. 이는 프로그램 수행시에 많은 명령어들 중의 일부분을 차지하는 기본 명령어들이 주로 수행되기 때문이다. 더구나 복잡한 프로세서의 구성으로 인하여 실리콘으로 구현할 때(즉, VLSI 칩으로 구현할 때), 경제성, 성능, 설계 비용 등의 문제점이 야기된다.

위에서 열거한 문제점들을 해결하기 위해 1980년경에 새로운 설계 개념이 대두되었다. 이는 프로세서가, 자주 사용되고 간단한 소수의 명령어들만으로 구성된 명령어 집합(Reduced Instruction Set)[1) 2)]을 채택하자는 것이다.

이 경우 명령어들의 구조가 단순하므로 파이프라인(pipelining) 기법을 사용하여 수행 효율을 높일 수 있어서, 성능 향상에 직접적인 도움이 된다. 또한 전체 프로세서를 하나의 칩안에 쉽게 구현할 수 있다.

1980년에 Berkeley대학은 이러한 명령어집합을 채택함으로써 성능 향상을 꾀하는 새로운 방법을 제시하고, 이것을 RISC(Reduced Instruction Set Computer)[2;3]라고 명명했다. 이 대학은 RISC Ⅰ과 RISC Ⅱ를 VLSI 칩으로 구현했으며, 이들은 최근에 개발된 다른 RISC프로세서들의 설계에 큰 영향을 끼쳤다. RISCI프로세서에 대한 benchmark결과[4]는 RISC Ⅰ이 VAX-11/780이나 M68000에 비해 2배내지 4배만큼 빠르다는 것을 보여 주었다.

1984년에 MIPS회사는 Stanford대학에서 개발된 RISC 프로세서인 MIPS칩을 개량하여 새로운 RISC프로세서[5]를 개발했다. benchmark결과[6]에 의하면, 이 칩이 같은 클럭 크기의 VAX-11/780이나 M68020보다 5배 내지 10배 빠르다.

RISC개념에 근거한 칩을 생산한 최초의 기업인 Pyramid Technology Co. 은 1983년에 이 칩을 이용한 90X라는 수퍼 미니 컴퓨터를 발표하였다. 또한 IBM의 PC RT와 HP의 Spectrum 시리즈도 RISC 개념에 근거한 프로세서를 사용하고 있다.

최근 SUN에서 개발한 SPARC(Scalable Processor ARChitecture)도 RISC 구조로 개발되었으며, 이것은 현재 SUN-4 워크스테이션에서 사용되고 있다. 특히 SPARC 칩은 open architecture의 형태로 공개되고 있어서 RISC 구조의 표준화 경쟁에서 유리한 고지를 차지할 것으로 보인다.

현재까지 개발된 RISC 프로세서들은 다음과 같은 공통적 특징을 갖는다.

첫째, 기본적인 연산들은 모두 레지스터간에 이루어지며, 메모리에 접근하는 명령어로 LOAD와 STORE만이 제공된다. 명령어들의 기능이 간단하기 때문에 명령어 수행 주기(instruction execution cycle)가 짧다.

둘째, 연산 방식과 주소 지정 방식이 간단하다. 예를 들어 RISC I, II의 경우 주소 지정 방식은 indexed mode와 PC-relative mode만 허용되며, 그 밖의 다양한 주소 지정 방식은 위의 두 방식의 응용을 통하여 제공된다.

세째, 명령어 형식(instruction format)이 단순하다. 모든 명령어에 대하여 레지스터 피연산자(operand)의 위치가 고정되어 있으므로, 레지스터에 대한 접근과연산자(op code)에 대한 해독(decoding)이 동시에 수행될 수 있다.

네째, RISC에서 명령어 수행시 발생하는 분기는 명령어들의 파이프라인 방식의 수행을 파기하지 않는다. 이는 프로세서 수행 속도 향상에 큰 역할을 담당한다.

다섯째, CISC에 비하여 RISC의 하드웨어 구조가 단순하기 때문에, 칩 설계 주기가 짧다.

한편 최근까지 개발된 RISC 프로세서들은 CISC 프로세서들에 비해 다음과 같은 문제점들을 내포하고 있다.

첫째, RISC 개념은 소프트웨어들을 하드웨어로 대체하여 수행 성능을 향상시키려는 일반적 추세에 역행한다.

둘째, 자료 처리 및 과학 계산 등의 모든 응용에 효율적인 RISC 구조를 설정한다는 것은 쉽지 않다. 각 응용 분야의 특성에 따라 가장 효율적인 RISC 구조가 독특하게 설정될 수 있으므로, 마이크로 프로그램 기법을 사용하는 것이 훨씬 더 효율적일 수 있다.

세째, RISC 프로그램은 CISC 프로그램에 비해 훨씬 크다. 따라서 RISC는 훨씬 많은 메모리를 필요로 한다.

2. RISC 프로세서의 구조

이 장에서는 대표적인 RISC 프로세서로 미국 Berkeley 대학에서 개발한 RISC II에 대하여 설명하고, 기타 다른 RISC 프로세서에 대해 간략하게 비교 설명한다.

Berkeley대학이 개발한 RISC I은 소수의 단순한 명령어 집합을 수행하는 최초의 RISC 프로세서이다. LOAD 및 STORE를 제외한 모든 명령어들이 레지스터에 관한 연산으로 하나의 기본 주기에 수행되며, LOAD와 STORE 명령어만이 두 기본 주기를 사용하여 메모리에 접근한다. 제어장치는 하드와이어화되어 있으며, 지연 분기 명령어의 채택과 파이프라인 방식에 의거한 명령어 수행을 통하여 처리 속도를 개선시켰다. 또한 procedure 호출 및 복귀를 신속하게 처리하기 위해 레지스터 화일의 개념을 도입하였다. 이후 레지스터 화일 구조의 수정, 파이프라인 단계의 세분화, 연산자의 재구성을 통한 제어장치의 단순화 등에 의거하여 RISC I [7]을 확장하여 RISC II[3]를 개발하였다.

지금부터, RISC II의 주요 특징인 명령어 집합, 레지스터 화일 및 파이프라인기법에 대해 기술한다.

2-1. 명령어 집합

RISC II가 제공하는 명령어들은 기본적으로 3개의 피연산자를 갖는 레지스터간(register-to-register) 명령어들이며 아래와 같은 기본형식을 갖는다.

$R_d \leftarrow R_{S1}$ op S_2

명령어 집합은 39개의 명령어들로 구성되며, op code 영역이 7비트로 표현되기 때문에 충분한 확장성을 가지고 있다. 특히 모든 명령어들이 32bit로 구성되며, 그림 1처럼 두 가지 형식으로 표현된다.

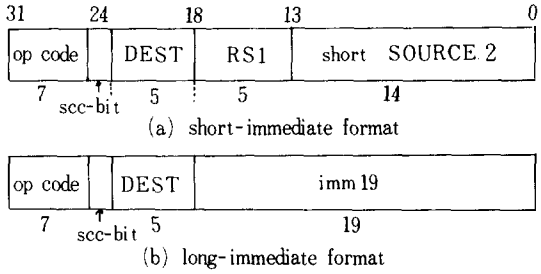

그림1 명령어 형식

여기서 short immediate와 long immediate는 각각 14 bit와 19bit로 구성되며, short immediate는 R_{S2}로 대치

될 수 있다.

RISCII가 제공하는 명령어들의 기능은 다음과 같다.

- 정수간의 덧셈 및 뺄셈
- 비트별 AND, OR, Exclusive-OR 연산
- 왼쪽 혹은 오른쪽으로의 논리적 자리이동(shift left/right-logical)
- 오른쪽으로의 산술적 자리이동(shift right-arithmetic)

RISCII에서는 LOAD와 STORE 명령어들만 메모리에 접근할 수 있으며, 이들은 다음과 같은 의미를 갖는다.

$R_d \rightleftarrows M\{R_{S1}+S2\}$

위의 메모리 주소 계산 방식은 R_{S1}이 PC 혹은 범용 레지스터 중 어느 것이 사용되느냐에 따라 PC-relative mode나 indexed mode의 주소 지정 방식으로 이해될 수 있다. 다른 명령어들과 달리 LOAD와 STORE 명령어들은 두 개의 기본 주기에 걸쳐 수행되는데, 첫번째 주기에서는 메모리 주소가 계산되고 두번째 주기 동안에 실제로 메모리에 접근하여 데이타를 읽거나 쓰게 된다.

2-2. 레지스터 화일

일반적으로 레지스터간 연산을 기본으로 하는 프로세서는 많은 레지스터들을 사용하게 때문에 procedure의 호출(call)과 복귀(return)시에 이러한 레지스터들을 저장(save)하거나 복원(restore)해야 하는 부담을 안고 있다.

이러한 부담을 줄이기 위하여 RISCII 프로세서는 32개의 레지스터들을 네 종류로 구분하여, 레지스터 화일을 구성한다. 첫번째 종류는 10개의 레지스터들로 구성되고, procedure의 호출 및 복귀시에 저장 및 복원될 필요가 없으며, 이들은 전역(global) 레지스터군으로 불리운다. 두번째 종류는 6개의 레지스터들로 구성되며, procedure를 호출하는 경우 인자를 넘겨 주거나, 결과를 되돌려받는데 사용한다. 세번째는 10개의 레지스터들로 구성되며, procedure의 호출 및 복귀시에 반드시 저장 또는 복귀되어야 한다. 네번째 종류는 6개의 레지스터로 구성되는데, 호출한 procedure로부터 인자를 전달받고, 결과를 되돌려 주는데 사용된다.

RISCII 프로세서는 두번째, 세번째, 네번째 종류에 해당하는 22개의 레지스터들로 구성된 레지스터 집합을 8개 사용하여, procedure의 호출 및 복귀 명령어를 수행할 때 자동적으로 해당 레지스터들을 저장 혹은 복원함으로써 프로세서의 수행 속도를 향상시킨다. 한편, 두번째와 네번째 종류의 레지스터들을 서로 공유시킴으로써, 전체 레지스터 화일 내의 레지스터 수를 줄일 수 있다.

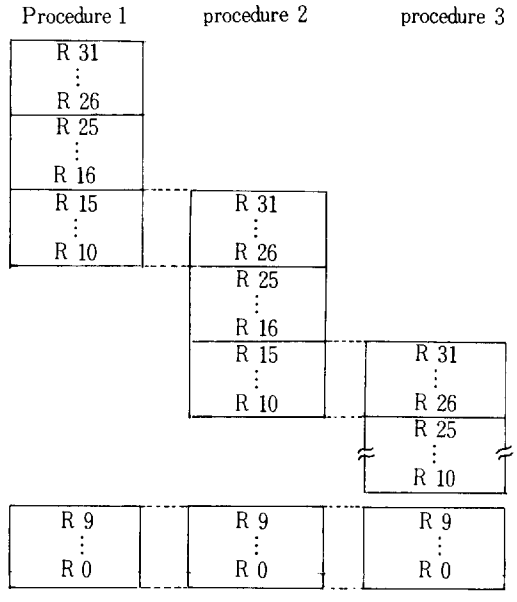

그림2 레지스터화일의 구조

RISCII는 이 방법에 따라 설계되었으며, 레지스터 화일은 138개의 레지스터로 구성되는데, 이 화일의 구조는 그림 2와 같다.

현재 수행중인 프로그램이 procedure1과 procedure2를 연속적으로 호출하여 procedure2가 수행되는 경우를 살펴보면, R0부터 R9까지가 첫번째 종류에, R10에서 R15 까지가 두번째 종류에, R16에서 R25까지가 세번째 종류에 해당되며, R26에서 R31까지의 레지스터가 네번째 종류로 사용된다. 또 R26부터 R31까지의 레지스터들은 procedure 1로부터 인자를 전달받아 결과를 되돌려 주는데 사용하기 위해 procedure1의 R10부터 R15까지의 레지스터들과 같은 레지스터들로 구성된다. 또 R10부터 R15까지의 레지스터들은 procedure3을 호출할 경우 인자를 전달하고 결과를 되돌려 받는데 사용된다.

2-3. 파이프라인 기법

컴퓨터 시스템의 성능 향상을 위해 파이프라인 수행 방식을 사용하기 시작한 것은 RISC 출현전의 일이다. 그러나 소수의 명령어 집합을 사용하고 그 명령어들의 수행 속도가 비슷한 경우, 그렇지 않는 경우보다 파이프라인의 효과가

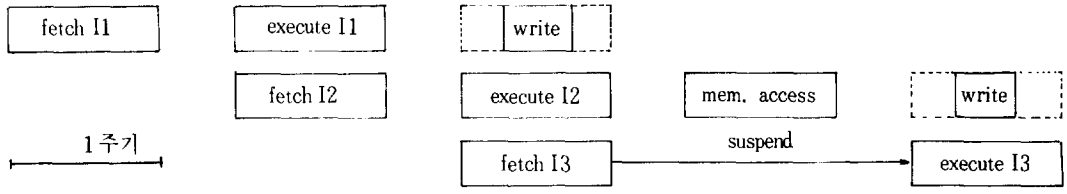

그림3 RISC Ⅱ 파이프라인

커진다. 이는 파이프라인 수행 시간이 각 단계 중에서 최대의 수행 시간을 갖는 단계에 좌우되기 때문이다. 따라서 RISCII 프로세서는 파이프라인 기법을 사용하여 명령 수행 속도를 개선하고 있다.

RISC I 이 두 단계의 파이프라인 기법을 사용하고 있는데 반해, RISCII는 그림 3과 같이 세 단계의 파이프라인 기법을 사용하여 명령어들을 수행한다.

명령어 I1은 레지스터간 명령으로 3단계에 걸쳐 수행되며, 명령어 I2는 메모리를 접근하는 LOAD 명령어로 4단계에 걸쳐 수행된다. 위의 예는 명령어 I3가 명령어 I2에 의해 읽혀진 피연산자를 사용하기 위해 중지된(suspend) 것을 보여준다.

일반적으로 파이프라인을 사용하는 컴퓨터 시스템에서 분기 명령어를 처리하여 실제로 분기가 발생하는 경우, 파이프라인내에 있던 이후의 명령어들은 더이상 수행되지 않는다. 이것은 분기 명령어의 의미(semantics)가 이후의 명령어들의 수행을 포기하고, 분기될 주소로부터 다시 명령어들을 가져 오는 것이기 때문이다.

RISCII 프로세서에서는 분기 명령어의 의미를 확장하여, 위에서 무시되었던 명령어들을 모두 처리함으로써 파이프라인 수행을 파괴하지 않고 명령어 수행 성능을 향상시킬 수 있다. 이 명령어는 지연 분기(delayed branch)라고 불리운다. 즉, 지연 분기 명령어에 의해 분기가 일어나는 경우라고 하더라도, 이미 파이프라인 방식으로 수행되고 있던 명령어들을 계속 수행하면서, 동시에 분기할 주소로부터 새로운 명령어를 가져온다는 것을 의미한다. 지연 분기 명령어 이후의 명령어들이 No-operation으로 대치된 경우는 CISC의 분기 명령 수행과 같아지지만, No-operation 대신에 다른 명령어들로 채움으로써 성능 향상을 꾀할 수 있다. 즉 사용자나 컴파일러가 분기와 상관없이 수행될 수 있는 명령어들을 지연 분기 명령어 이후에 배치함으로써, 해당 명령어에 대한 수행 시간을 단축시킬 수 있다.

위의 그림은 RISCII에서 지연 분기 명령어를 처리하는

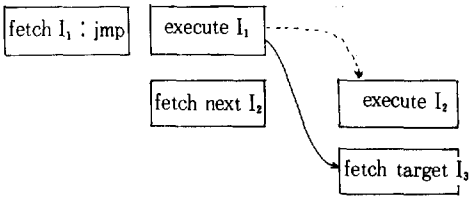

그림4 지연분기

방법을 보여준다. I1 명령어는 지연 분기 명령어로 2번째 주기에서 분기후에 수행될 명령어의 주소가 계산되고, 그와 동시에 다음 명령어 I2 를 가져온다. CISC의 분기 처리 방식은 세번째 주기에서 I2 명령어를 수행하지 않고 분기될 주소에 있는 명령어 I3 를 가져온다. 반면에 RISC의 지연 분기 수행 방식은 이미 가져온 I2 명령어를 수행하면서 분기될 주소에 있는 I3 명령어를 가져오게 되므로, I2 명령어를 수행하는 시간만큼 성능 향상을 기대할 수 있다. 물론 I2 명령어는 분기의 발생 여부와 관계없는 명령이어야 하며, 적절한 명령어가 없어서 No-operation으로 채운 경우는 CISC의 수행 방식과 같아진다.

다음은 CISC의 분기 명령 처리 방법과 RISC의 지연 분기 방법에 대한 기계어 프로그램이다.

표1 분기명령어 처리에 대한 비교

	CISC의 분기명령어		RISC의 지연분기		최적화된 지연분기	
100	LOAD	X, A	LOAD	X, A	LOAD	X, A
101	ADD	1, A	ADD	1, A	JUMP	105
102	JUMP	105	JUMP	106	ADD	1, A
103	ADD	A, B	NO-OP		ADD	A, B
104	SUB	C, B	ADD	A, B	SUB	C, B
105	STORE	A, Z	SUB	C, B	STORE	A, Z
106			STORE	A, Z		

CISC의 분기 처리 방법에 따르면, 주소 102의 JUMP 명령을 처리한 후 새로운 명령을 가져오기 위해 파이

표2 RISC 프로세서에 대한 비교

	RISC I	RISC II	Stanford MIPS	MIPS	Sun SPARC
Single Cycle	Yes	Yes	Yes	Yes	Mostly
Load / Store	Yes	Yes	Partly	Partly	Yes
Hardwired	Yes	Yes	Yes	Yes	?
Small Inst. Set	31	39	65	79	64
Fixed Format	Yes	Yes	2	2	3
레지스터 수	138	138	16	32	120
컴파일러의 복잡도	중간	중간	복잡	복잡	?

프라인 수행 과정에 지연이 발생한다. 반면 RISC가 최적화된 프로그램에 대해 지연 분기 명령어를 처리하는 경우는, 주소 101의 JUMP 명령이 수행되는 것과 상관없이 주소 102의 명령을 수행한 후에 105로 분기하게 되므로 더 효율적인 파이프라인 처리 방식이라고 할 수 있다.

위의 표에서 보듯이 MIPS 및 Stanford MIPS와 Sun SPARC는 RISC I, II보다 많은 명령어들을 가지고 있으며, 명령어 형식도 2개 혹은 3가지 종류로 구성되므로, Berkeley대학의 RISC I, II에 비해 하드웨어의 단순성이 떨어진다. 또한 MIPS나 Stanford MIPS의 경우는 하나의 레지스터 집합만을 사용하기 때문에 procedure의 호출 및 복귀에 따르는 성능저하가 야기될 수 있다.

3. 앞으로의 연구 과제

Berkeley대학에서 RISC I, II를 개발한 이래, 많은 종류의 RISC 프로세서가 개발되어 사용되고 있다. 그러나 이러한 RISC 구조들은 더욱 개선될 여지가 있으며, 이를 위해 앞으로 수행해야 할 연구 과제중 일부분은 다음과 같다.

첫째, 레지스터 화일에 관한 오버플로우(overflow) 처리 방법에 관한 연구이다. 대부분의 RISC 구조는 여러 개의 레지스터 집합을 가지고 있지만, procedure의 연속적인 호출로 인하여, 레지스터 화일로부터 레지스터들이 더 이상 할당될 수 없는 경우가 발생하는데, 이것을 레지스터 화일의 오버플로우라고 한다. 이런 경우는 레지스터들을 메모리에 저장하거나, 메모리로부터 레지스터로 복원시키기 위하여 별도의 명령어들을 수행시켜야 하므로, 실질적인 성능 저하가 발생된다. 그러므로 레지스터화일의 오버플로우에 따른 성능 저하를 최소화시키기 위한 방법에 대한 연구가 필요하다.

둘째, 파이프라인의 각 단계가 명령어와 데이타를 동시에 요구하여 메모리 접근에 대한 충돌(conflict)이 발생하는 경우, 이를 동시에 처리하지 못하므로 파이프라인의 성능이 저하된다. 따라서 이러한 성능 저하를 방지하는 방법에 관한 연구가 필요하다.

세째, 현재의 RISC 구조는 하나의 워드가 한 개의 명령어만을 저장하기 때문에 명령어들의 병렬 수행에 제한을 받고 있다. 따라서 하나의 워드에 두개 이상의 명령을 저장하여 명령어들을 병렬적으로 수행할 수 있는 RISC-구조에 대해 연구할 가치가 있다.

참 고 문 헌

1) D. A. Patterson, "Reduced Instruction Set Computers", Communications of the ACM, January 1985, pp. 8-21.

2) W. Stallings, Computer Organization and Architecture, 1986.

3) Manolis G. H. Katevenis, "Reduced Instruction Set Computer Architectures for VLSI", Doctorial Dissertation, Berkeley University.

4) D. A. Patterson, R. S. Piepho, "Assessing RISCs in High-Level language Support", IEEE Micro, November 1982 00. 9-18.

5) J. L. Hennessy, "VLSI Processor Architecture", IEEE Transactions on Computers, December 1984, pp. 1221-1246.

6) C. Bell, "RISC : Back to the Future?", Datamation, June 1986, pp. 96-108.

7) J. L. Heath, "Re-evaluation of the RISC I", Computer Architecture News, March 1984, pp. 3-10.