16/32비트 길이 명령어를 갖는 32비트 마이크로 프로세서에 관한 연구

조 경 연†

요 약

마이크로 프로세서의 동작 속도가 빨라지면서 메모리의 데이터 전송 폭이 시스템 성능을 제한하는 중요 인자로 대두되면서 코드 밀도가 높은 컴퓨터 구조에 대한 연구의 필요성이 증대되고 있다. 본 논문에서는 코드 밀도가 높은 32비트 마이크로 프로세서 구조로 16비트와 32비트 2종류 길이의 명령어를 가지는 가청 2가지 길이 명령어 세트 컴퓨터(Bi-length Instruction Set Computer : BISC)를 제안한다. 32비트 BISC는 16개의 범용 레지스타를 가지며, 오프셋과 상수 오퍼랜드의 길이에 따라서 2종류의 명령어를 가진다. 제안한 32비트 BISC는 FPGA로 구현하여 1.8432MHz에서 모든 기능이 정상적으로 동작하는 것을 확인하였고, 크로스 어셈블러와 크로스 C/C++ 컴파일러 및 명령어 시뮬레이터를 설계하고 동작을 검증하였다. BISC의 코드 밀도는 기존 RISC의 130~220%, 기존 CISC의 130~140%로 높은 장점을 가진다. 따라서 데이터 전송 폭을 적게 요구하므로 차세대 컴퓨터 구조로 적합하고, 프로그램 메모리 크기가 작아지므로 실장 제어용 마이크로 프로세서에 적합하기 때문에 폭 넓은 활용이 기대된다.

A Study on 16/32 bit Bi-length Instruction Set Computer 32 bit Micro Processor

Gyoung-Youn Cho†

ABSTRACT

The speed of microprocessor getting faster, the data transfer width between the microprocessor and the memory becomes a critical part to limit the system performance. So the study of the computer architecture with the high code density is emerged. In this paper, a tentative Bi-Length Instruction Set Computer(BISC) that consists of 16 bit and 32 bit length instructions is proposed as the high code density 32 bit microprocessor architecture. The 32 bit BISC has 16 general purpose registers and two kinds of instructions due to the length of offset and the size of immediate operand. The proposed 32 bit BISC is implemented by FPGA, and all of its functions are tested and verified at 1.8432MHz. And the cross assembler, the cross C/C++ compiler and the instruction simulator of the 32 bit BISC are designed and verified. This paper also proves that the code density of 32 bit BISC is much higher than the one of traditional architecture, it accounts for 130~220% of RISC and 130~140% of CISC. As a consequence, the BISC is suitable for the next generation computer architecture because it needs less data transfer width. And its small memory requirement offers that it could be useful for the embedded microprocessor.

† 정 회 원 : 부경대학교 컴퓨터멀티미디어공학부 교수
논문접수 : 1998년 10월 23일, 심사완료 : 2000년 1월 9일

1. 서 론

1970년대에 개발된 마이크로 프로세서는 1980년대에 이르러 RISC(Reduced Instruction Set Computer)[1] 구조의 도입으로 실장 제어 기기 분야에서 중대형 및 소형 컴퓨터에 이르기까지 광범위하게 사용되고 있다. 또한 반도체 기술의 급격한 발전으로 슈퍼스칼라 구조[2]가 마이크로 프로세서에도 적용되고 있으며 동작 속도도 수백 MHz에 이르고 있다[3-6].

마이크로 프로세서는 프로그램을 수행하기 위해서 프로그램과 데이터를 메모리로부터 읽어 와야 한다. 그런데 메모리 용량은 빠른 속도로 증가하고 있지만 동작 속도는 마이크로 프로세서의 동작 속도에 크게 미치지 못하고 있다. 1980년에 DRAM의 접근 속도는 250 nsec이었으나 1998년에 RDRAM의 동작속도는 300MHz로 70여 배 빨라지는 데 그치고 있다. 그러나 마이크로 프로세서는 1980년에 8086의 동작 속도는 8MHz이었으나 1998년에는 팬티엄-2가 500MHz에 이르고 있다. 더욱이 팬티엄-2는 슈퍼스칼라 구조이므로 이를 감안하면 1GHz 이상에 이르러 120여 배 빨라진 것을 알 수 있다. 이와 같은 메모리 속도와 마이크로 프로세서 속도 차이에 더하여, 메모리와 마이크로 프로세서를 인쇄 회로 기판에서 연결하는 데 따른 물리적 특성은 변화하지 않으므로 데이터 전송 폭을 넓히는 것에는 한계가 있다.

따라서 향후 컴퓨터 성능 발달을 제한하는 주요 요소 중 하나는 마이크로 프로세서와 메모리사이의 데이터 전송 폭이다[7]. 프로그램과 데이터가 메모리에 저장되는 본 노이먼(Von Neumann) 방식의 컴퓨터에서 데이터 전송 폭을 줄이기 위해서는 코드 밀도(Code Density)가 높은 컴퓨터 구조를 연구하는 것이 필요하다.

한편 마이크로 프로세서는 실장 제어용으로 거의 모든 전자 제품 및 자동화 기기에서 채용하고 있다. 특히 냉장고, 에어콘, 전축, TV, 세탁기 등 가전기기와 Fax, 복사기, 프린터 등 사무용기기와 자동차, 선박, 자동화기계 등 사무 및 산업용 기기와 PDA(Personal Digital Assistant), NC(Network Computer) 등 정보 기기 그리고 각종 오락기, 노래반주기 등 정보 기기 등에서 사용하는 실장 제어용 마이크로 프로세서 시장은 매년 10% 이상씩 성장하고 있으며, 21 세기 산업을 주도하는 핵심 기술로 자리 매김하고 있다[8].

이러한 실장 제어용 기기는 마이크로 프로세서와 메모리 및 입출력 장치가 하나의 반도체에 집적되는 경우가 많다. 그런데 반도체 가격은 반도체 크기에 따라 결정되며, 가장 넓은 면적을 차지하는 것은 메모리이다. 따라서 반도체 가격을 낮추기 위해서는 메모리 크기를 줄여야 하며, 이를 위해서 또한 코드 밀도가 높은 컴퓨터 구조에 대한 연구가 필요하다[8].

최근에는 32비트 RISC 명령어를 16비트 명령어로 축약한 구조가 연구되었다[9]. ARM-7TDMI는 ARM-7의 16비트 축약 명령어 구조이며, TR4101은 MIPS-R3000의 16비트 축약 명령어 구조이다[8, 9]. 이들 16비트 축약 명령어 RISC는 종래 RISC와의 호환성을 위하여 2가지 모드로 동작하므로 구조가 복잡하고, 16비트 명령어에서는 8개의 레지스타만을 접근할 수 있으므로 성능이 크게 떨어지는 단점을 가진다.

본 논문에서는 코드 밀도가 높은 32비트 마이크로 프로세서 구조로 16비트와 32비트 2가지 길이 명령어를 가지는 가칭 2가지 길이 명령어 세트 컴퓨터(Bi-length Instruction Set Computer : BISC)를 제안한다.

이를 위하여 본 논문에서는 MIPS-R3000으로 작성된 프로그램을 분석하여 명령어 사용 특성을 분석한다. C/C++ 컴파일러는 EGCS-1.1[10]을 사용하였으며 C 라이브러리 NEWLIB- 1.8.1[11]과 C++ 라이브러리 LIBSTDC++-2.8.1[12] 및 벤치마크 프로그램을 대상으로 하여 분석하였다. 이들은 C/C++로 작성된 대표적인 프로그램이고, 컴파일된 기계어 크기는 약 500 키로 바이트로 대상 프로그램으로 적합하다.

분석 결과 범용 레지스타는 16개를 사용하는 것이 적합함을 보인다. 그리고 로드, 스토어 명령어가 차지하는 비중이 높으며, 짧은 길이의 오프셋을 사용하는 경우가 대부분임을 보인다. 또한 상수의 사용에서도 작은 크기의 상수 빈도가 높음을 보인다. 이러한 특성을 효율적으로 지원하기 위해서 오프셋 및 상수 오퍼랜드의 크기에 따라서 16비트와 32비트의 2가지 길이 명령어를 가지는 32비트 BISC의 명령어 세트를 설계한다. 32비트 BISC는 코드 밀도를 더욱 높이기 위하여 레지스타 리스트 푸시, 팝 명령을 가지며, 파이프라인 제어를 하드웨어로 수행하여 불필요한 NOP 명령어를 제거하였다.

제안한 32비트 BISC는 FPGA로 구현하여 1.8432 MHz에서 모든 기능이 정상적으로 동작하는 것을 확인하였다. 또한 크로스 어셈블러와 크로스 C/C++ 컴파일러 및 명령어 시뮬레이터를 설계하였다. 자료 구조 및

수학 함수 프로그램을 C/C++ 언어로 작성하고, 이들을 크로스 컴파일러와 어셈블러를 사용하여 기계어로 번역하고, 이것을 시뮬레이터에서 동작시킨 결과와 IBM-PC에서 전용 컴파일러를 사용하여 얻은 결과를 비교하여 동작을 검증하였다.

그리고 제안한 BISC와 기존의 RISC 및 CISC의 코드 밀도를 비교하였다. 제안한 BISC의 코드 밀도는 기존 RISC의 130~220%, 기존 CISC의 130~150%로 기존 구조에 비하여 데이터 전송 폭을 크게 낮출 수 있으며, 이에 비례하여 프로그램 크기도 작아 진다. 또한 16비트 축약 명령 RISC보다 코드 밀도가 3~10% 높으며, 로드 스토어 비중이 15% 낮다.

제2장에서는 MIPS-R3000 프로그램을 분석하고 이에 따른 32비트 BISC 구조를 제안하며, 제3장에서는 제안한 32비트 BISC의 하드웨어 및 소프트웨어를 구현하고 제4장에서는 기존의 RISC 및 CISC 구조와 비교하여 성능을 검증하며, 제5장에서는 결론을 맺는다. 그리고 부록에 제안한 32비트 BISC의 명령어 세트를 보인다.

2. 32비트 BISC 구조

2.1 16개 범용 레지스타

MIPS-R3000은 34개의 32비트 레지스타를 가지고 있다. 2개는 곱셈 및 나눗셈 계산을 위한 전용 레지스타이며, 5개는 스택, 프레임 포인터, 조건 등을 저장하는 특수 레지스타로 사용하고 있어서 범용 레지스타로는 27개를 사용하고 있다. <표 1>에는 EGCS C/C++ 컴파일러를 수정하여 MIPS-R3000의 범용 레지스타 수에 따른 컴파일러를 작성하고, 이를 이용하여 C/C++ 라이브러리 및 벤치마크 프로그램을 컴파일한 결과를 보인다.

〈표 1〉 MIPS-R3000에서 레지스타 수에 따른 프로그램 특성

No. of Register	Program size	Load/Store	Move
27	100.00	27.90%	22.58%
24	100.35	28.21%	22.31%
22	100.51	28.34%	22.27%
20	100.56	28.38%	22.24%
18	100.97	28.85%	21.93%
16	101.62	30.22%	20.47%
14	103.49	31.84%	19.28%
12	104.45	34.31%	16.39%
10	109.41	41.02%	10.96%
8	114.76	44.45%	8.46%

<표 1>에서 프로그램 크기는 범용 레지스타가 27개인 경우를 100으로 정규화한 수치로 나타내고 있다. 범용 레지스타 수가 작아지면 로드, 스토어의 빈도와 프로그램 크기가 증가한다. 로드, 스토어는 메모리 입출력을 동반하므로 데이터 전송 폭에 직접적인 영향을 미친다. 반면에 래지스타 수가 많아지면 명령어 길이가 길어져서 프로그램 크기가 커진다. 이러한 점을 감안하면 범용 레지스타가 16개인 경우가 27개인 경우와 비교하여 프로그램 크기나 로드, 스토어 빈도에서 큰 차이가 없다. 반면에 8개의 범용 레지스타는 로드, 스토어 빈도가 너무 높아서 비효율적임을 알 수 있다.

따라서 본 논문에서 제안하는 32비트 BISC에서는 16개의 범용 레지스타를 사용한다. 이후의 프로그램 분석은 16개 레지스타를 사용하도록 변형한 MIPS-R3000 컴파일러를 사용한다.

2.2 로드, 스토어 구조

<표 2>에 16개 범용 레지스타를 가지는 MIPS-R3000 프로그램에서 명령어 사용 빈도를 보인다.

〈표 2〉 16개 범용 레지스타 MIPS-R3000에서 명령어 사용 빈도

Instruction	Frequency
move	20.27%
lw, sw	28.27%
nop	7.26%
addiu	7.53%
li	2.93%
lui	3.76%
sh, sb, lh, lb, lhu, lbu	1.98%
bnez, bne, beqz, beq, bltz, ...	6.69%
j, jal	10.36%
jr	1.79%
addu, subu, and, or, xor, nor, negu	3.33%
andi, ori, xori	2.17%
jalr	0.17%
slt, sltu, slti, sltiu	1.70%
sll, srl, sra, sllv, srlv, srav	1.40%
mult, multu, div, divu	0.09%
break, mfhi, mflo	0.12%

<표 2>에서 변수 산술연산 명령어(addu, subu, and 등)는 3.33%로 빈도가 높지 않다. 이들 명령어에서 메모리 변수를 사용하는 빈도는 출현 빈도보다 높지 않

다. 즉 메모리 연산 명령어의 출현 빈도가 작으므로 이를 위한 명령어는 정의하지 않는 것이 효율적이다.

32비트 BISC는 모든 연산 명령은 레지스타 오퍼랜드를 가지며, 메모리 입출력은 로드 스토어 명령어로 제한하는 로드, 스토어 구조를 가진다. 이러한 로드, 스토어 구조를 가지므로 하드웨어 구조가 단순해지고 따라서 동작 속도를 빠르게 할 수 있다.

2.3 16/32비트 두 가지 길이 명령어

<표 2>에서 'move' 명령어가 20.27%의 사용 빈도를 보이고 있다. 32비트 BISC에서 레지스타 수는 16개이므로 원시 레지스타 및 목적 레지스타 표현에 각각 4비트가 필요하다. 따라서 'move' 명령어는 16비트 길이 명령어로 표현할 수 있다.

16비트 단일 길이 명령어를 사용하면 하드웨어가 단순하여 진다. 대부분의 명령어는 'move' 명령어와 같이 16비트 길이 명령어로 표현할 수 있으나, 로드, 스토어 명령과 상수 오퍼랜드 명령어는 오프셋 및 상수 오퍼랜드 길이를 감안하면 16비트 길이 명령어로 표현하기가 용이하지 않다.

<표 2>에서 32비트 로드, 스토어가 전체 로드, 스토어의 93.5%를 점유하고 있다. 따라서 로드, 스토어 명령의 특성을 분석하기 위해서 <표 3>에 'lw(load word), sw(store word)'의 사용 특성을 보인다.

〈표 3〉 'lw, sw' 명령어 특성

Offset length	Stack pointer (60.9%)	Index register (39.1%)
3 bit	43.2%	77.0%
4 bit	72.6%	81.6%
5 bit	88.1%	89.6%
6 bit	90.5%	91.5%
7 bit	95.7%	93.5%

<표 3>에서 'lw, sw' 명령어의 60.9%가 스택 포인터를 사용하고 있다. 이것은 지역 변수 및 전달 변수가 스택에 설정되기 때문이다. 인덱스 레지스타를 사용하는 경우에는 4비트 오프셋으로 81.6%의 명령어를 표현할 수 있다. 이것은 구조체의 크기가 크지 않은 것을 나타내며, 또한 자주 사용하는 변수를 구조체 앞단에 선언하는 것으로 그 빈도를 더욱 증가시킬 수 있다.

그리고 상수 오퍼랜드 명령어인 'li(load immediate)' 명령어는 <표 2>에서 2.9%를 차지하며 상수의 출현 빈도는 <표 4>와 같다.

〈표 4〉 'li' 명령어에서 상수의 사용 빈도

Constant range	Frequency
-32 -- +31	72.4%
-64 -- +63	86.9%
-128 -- +127	93.6%
-256 -- +255	95.2%
others	100%

<표 4>로부터 'li' 명령어의 93.6%는 8비트 상수로 표현할 수 있다.

이상으로부터 'lw, sw' 명령어에서 짧은 길이 오프셋과 'li' 명령어에서 짧은 길이 상수 오퍼랜드 명령어의 출현 빈도가 높다는 것을 확인할 수 있다. 이러한 현상은 'addiu, slti, sltiu' 명령어 등에서도 공통적으로 나타난다.

이와 같이 빈도가 높은 짧은 길이 오프셋과 상수 오퍼랜드를 지원하기 위해서 32비트 BISC는 오프셋 길이와 상수 오퍼랜드 길이에 따라서 두 가지 명령어를 가진다. 짧은 길이 오프셋과 상수 오퍼랜드 명령어는 16비트 길이가 되며, 16비트 오프셋과 상수 오퍼랜드 명령어는 32비트 길이가 된다.

짧은 길이 오프셋을 가지는 로드, 스토어 명령어는 다음과 같다.

Instruction Function : Short offset load/store
Instruction Representation :
bit 15 = 0
bit 14-12 = Operation
000 : Sign extend 8 bit load. LDB src, dst
001 : Sign extend 16 bit load. LDS src, dst
010 : 32 bit load. LD src, dst
011 : Zero extend 8 bit load. LDBU src, dst
100 : 8 bit store. STB src, dst
101 : 16 bit store. STS src, dst
110 : 32 bit store. ST src, dst
111 : Zero extend 16 bit load. LDSU src, dst
bit 11-8 = Source/Destination register. %R0 thru %R15.
bit 7-4 = Index register. %R0 thru %R15.
bit 3-0 = Offset bit 3-0 if 8 bit load/store
Offset bit 4-1 if 16 bit load/store
Offset bit 5-2 if 32 bit load/store

짧은 길이 오프셋 로드, 스토어 명령어는 전체 로드, 스토어 명령문의 81.6%를 점유한다. 16비트 오프셋을 가지는 로드, 스토어 명령어는 다음과 같이 별도로 정의한다.

Instruction Function : 16 bit offset load/store
Instruction Representation :
bit 15-11 = 1010 0
bit 10-8 = Operation
bit 7-4 = Source/Destination register. %R0 thru %R15.
bit 3-0 = Index register. %R0 thru %R15.
*** Second word = Sign extend offset bit 15-0

이와 같이 오프셋과 상수 오퍼랜드의 길이에 따라 2종류의 명령어를 정의하면 대부분의 경우에 있어서 16비트 명령어를 사용하므로 코드 밀도를 높일 수 있다. 따라서 32비트 BISC에서는 로드, 스토어 명령어, 상수 로드 명령어, 제어 분기 명령어를 16비트와 32비트 2가지 길이 명령어로 정의한다.

2.4 스택 명령어

<표 3>에서 'lw, sw' 명령어의 오프셋 길이가 스택 포인터와 인덱스 레지스타를 사용하는 경우에 현격한 차이를 보이고 있다. <표 3>으로부터 스택 포인터를 사용하는 경우에 'lw, sw' 오프셋은 5비트 이상이 필요하다. 32비트 BISC에서는 이러한 특성을 감안하여 7비트 오프셋을 가지는 스택 포인터 'lw, sw' 명령어를 별도로 정의한다.

또한 'lw, sw' 명령어에서 스택에 푸시 팝하는 빈도가 15.8%로 조사되었으며 8개 레지스타를 하나의 리스트로 선언하면 평균 푸시 팝 레지스타 수는 4.3개로 조사되었다. 그러므로 32비트 BISC에서는 8개 레지스타를 하나의 리스트로 묶어서 표현하는 푸시 팝 명령어를 정의한다. 이러한 푸시 팝 레지스타 리스트는 여러 개의 메모리 동작을 발생시키는 명령어로 RISC 구조에서는 사용되지 않으나 CISC 구조에서는 많이 사용하고 있다.

<표 2>에서 'addiu(add immediate)' 명령어의 출현 빈도는 7.53%이다. 이중에서 스택 포인터를 사용하는 빈도는 35%이며, 이중 7비트 상수의 빈도가 99.1%이다. 그러므로 32비트 BISC에서는 7비트 상수를 스택 포인터에 더하고, 빼는 명령어를 정의한다.

2.5 기타 명령어

<표 2>에서 조건 분기 명령어의 빈도가 6.69%이다. 이를 효율적으로 지원하기 위해서 32비트 BISC에서는 캐리, 사인, 제로, 오버플로우의 4개 플래그를 사용하며 이들을 조합하여 14가지 조건 분기 명령어를 만든다. 분기 명령어의 오프셋은 8비트와 24비트 2가지 길이를 가진다. 8비트 오프셋 분기 명령어는 16비트, 24비트 오프셋 분기 명령어는 32비트 길이가 된다.

논리 산술 연산 명령어는 48.5%가 2오퍼랜드 명령어이고 51.4%가 3오퍼랜드 명령어로 조사되었다. 3오퍼랜드 명령어는 'move' 명령과 2오퍼랜드 명령어의 조합으로 표현할 수 있으므로 32비트 BISC의 논리 산술 연산 명령어는 2오퍼랜드 형식을 가진다. 따라서 레지스타 오퍼랜드 논리 산술 연산 명령어는 16비트 길이가 되고, 16비트 상수 오퍼랜드 논리 산술 연산 명령어는 32비트 길이가 된다.

곱셈, 나눗셈 명령어는 출현 빈도는 낮으나 멀티미디어 등 특정 응용 분야에서 사용 빈도가 특히 높으며, 구현 방식에 따라 동작 속도에 차이가 크다. 이러한 성질을 감안하여 곱셈, 나눗셈 결과를 저장하는 2개의 32비트 레지스타(%ML, %MH)를 설정한다. 그러므로 곱셈, 나눗셈 명령어는 16비트 길이이다.

코프로세서는 최대 8개를 설정할 수 있으며, 각각의 코프로세서는 16개의 범용 레지스터를 가지도록 한다. 코프로세서 '0'은 시스템 코프로세서로서 캐시 제어, 파이프라인 제어, 메모리 관리 등의 시스템 관리를 수행한다. 부동수소점수 연산기 및 멀티미디어 가속기 등은 별도 코프로세서에 의하여 구현한다. 코프로세서 명령은 32비트 길이를 가진다.

3. 하드웨어 및 소프트웨어 구현

<표 5>에 32비트 BISC의 레지스타 구성을 보인다.

제안한 32비트 BISC는 FPGA로 구현하여 1.8432MHz에서 정상적으로 동작하는 것을 확인하였다. 곱셈기는 modified Booth 알고리즘을 사용하였으며, 시프터는 1, 2, 3, 4비트 시프터를 만들고 이들을 조합하였다. 따라서 31비트 시프트 명령어는 최대 8클럭이 소요되었다. 코프로세서는 구현하지 않았고, 파이프라인은 명령어 패치, 연산, 메모리 저장의 3단계로 설계하여서, 약 3만 게이트가 소요되었다.

설계한 FPGA 32비트 BISC는 LED 제어기, RS-232C

〈표 5〉 32비트 BISC의 레지스타 구성

Register name	Description
R0 - R15	32 bit general purpose register
PC	32 bit program counter
USP	32 bit user stack pointer Co-processor#0 R13 at supervisor
SSP	32 bit supervisor stack pointer Not accessable at user mode
LR	32 bit link register
ML	32 bit multiply result low register 32 bit divide quotient register
MH	32 bit multiply result high register 32 bit divide remainder register
SR	32 bit status register
bit 31 = User/Supervisor# mode bit 18 = Enable NMI bit 17 = Auto-vector interrupt if 0 bit 16 = Enable maskable interrupt bit 7 = Carry flag (CY) bit 6 = Zero flag (Z) bit 5 = Sign flag (S) bit 4 = Overflow flag (V) *** If (user_mode && (%SR == source_register)) b31-b16 are always 0s. *** If (user_mode && (%SR == dest_register)) b31-b16 are unchanged.	

제어기, 타이머, 인터럽트 제어기, 메모리 제어기를 갖춘 FPGA와 함께 인쇄 회로 기판에 장착하였고, RS-232C로 IBM-PC와 연결하여 프로그램을 다운 로드 받아서 수행하고, 수행 결과를 LED에 표시하거나 RS-232C를 통하여 IBM-PC에 출력하여 동작을 검증하였다.

<표 6>에는 크로스 소프트웨어 구현 현황을 보인다.

〈표 6〉 32비트 BISC 크로스 소프트웨어

Host platform	Window-95, Window-NT, Linux, SUN
Object file format	ELF
Cross assembler	FSF Binutils-2.9.1
Cross loader	FSF Binutils-2.9.1
Binary utilities	FSF Binutils-2.9.1
Cross C compiler	Cygnus EGCS-1.1
Cross C++ compiler	Cygnus EGCS-1.1
C library	Cygnus NEWLIB-1.8.1
C++ library	FSF LIBSTDC++-2.8.1
Cross debugger	FSF GDB-4.17
Simulator	FSF GDB-4.17
Real Time OS	uC/OS-1.5

크로스 소프트웨어는 프리웨어로 제공되는 프로그램을 포팅하여 개발하였으며, 개발의 편의성을 위하여 IBM-PC의 Linux OS 상에서 개발하였고, 개발된 소프트웨어를 Window와 SUN에 재포팅하였다. 수학 함수 및 자료 구조 등에 관한 예제 프로그램을 C와 C++로 작성하고 크로스 소프트웨어로 컴파일하여 기계어를 생성하고, 생성된 기계어를 시뮬레이터에서 수행하여 얻은 결과를 IBM-PC의 전용 컴파일러에 의한 결과와 비교하여 동작을 검증하였다.

4. 성능 평가

코드 밀도는 프로그램 크기에 반비례한다. 상대 코드 밀도(Relative Code Density : RCD)를 대상 마이크로 프로세서에 대한 32비트 BISC의 상대적인 코드 밀도로 정의하면 식 (1)과 같이 표현된다.

$$RCD = \frac{Code-Density-of-32bit-BISC}{Code-Density-of-Object-Microprocessor} = \frac{Program-Size-of-Object-Microprocessor}{Program-Size-of-32bit-BISC} \quad (1)$$

성능 평가를 위하여 <표 6>의 환경으로 32비트 BISC 및 기존의 마이크로 프로세서들에 대한 크로스 C/C++ 컴파일러를 작성하고, C/C++ 라이브러리 및 벤치마크 테스트 프로그램을 컴파일하여 구한 상대 코드 밀도를 <표 7>에 보인다.

〈표 7〉 32비트 BISC에 대한 상대 코드 밀도

Processor	Relative Code Density	Processor	Relative Code Density
32 bit BISC	1.00	MC88000	1.53
MIPS-R3000	1.61	MC5200	1.39
TR4101	1.03	MC68000	1.30
MIPS-R4000	1.41	MC68332	1.28
MIPSTX-39	1.53	MC68020	1.28
ARM-7	1.59	MN10300	1.15
ARM-7TDMI	1.10	Pentium	1.34
PowerPC 601	1.85	I80960	1.62
SPARC V8	1.34	ARC	2.12
SPARCLITE	1.72	SH-3	1.33
PA-RISC	2.25	V850	1.17
Alpha-RISC	2.15	M32R	1.40

<표 7>에서 대표적인 32비트 RISC인 MIPS-R3000의 상대 코드 밀도가 1.61이다. 즉, 32비트 BISC의 코드 밀도가 MIPS-R3000보다 61% 높으며, 따라서 MIPS-3000의 프로그램 크기가 32비트 BISC보다 61% 크다는

것을 나타내고 있다. 실장 제어 기기 분야에서 많이 쓰이는 ARM-7의 상대 코드 밀도는 1.59이다. 따라서 ARM-7의 프로그램 크기는 32비트 BISC보다 59% 커진다. <표 7>로부터 RISC의 상대 코드 밀도는 1.3~2.2이며 따라서 기존 RISC의 프로그램 크기는 32비트 BISC보다 30~120% 커진다.

또한 MC68000 및 I80386 등 기존 CISC의 상대 코드 밀도는 1.3~1.4로 나타나고 있다. 즉 기존 CISC의 프로그램 크기는 32비트 BISC보다 30~40% 커진다.

마쓰시타의 MN10300은 4개의 데이터 레지스타와 4개의 인덱스 레지스타를 가지므로 로드, 스토어 빈도가 높아서 데이터 전송 폭이 크며, 명령어는 8비트부터 56비트까지 다중 길이를 가지므로 구조가 복잡하다. 32비트 BISC는 16개의 범용 레지스타를 가지므로 로드, 스토어 빈도가 MN10300보다 작으며, 16비트와 32비트 2가지 길이 명령어로 구조가 간단하며, 프로그램 크기도 15% 정도 작다.

16비트 축약 명령어 RISC인 ARM-7TDMI 및 TR4101의 상대 코드 밀도는 각각 1.10과 1.03으로 32비트 BISC보다 프로그램 크기가 조금 크고, 또한 범용 레지스타가 8개이므로 로드, 스토어가 증가한다. 32비트 BISC와 16비트 축약 명령어 RISC의 비교표를 <표 8>에 보인다.

〈표 8〉 32비트 BISC와 16비트 축약 명령어 RISC의 비교

	32 bit BISC	TR4101	ARM-7TDMI
Relative Code Density	1.00	1.03	1.10
Load/Store	30.2%	48.3%	46.5%

로드 스토어 명령은 메모리 읽기 쓰기를 동반하므로 데이터 전송 폭을 증가시킨다. <표 8>로부터 TR4101은 32비트 BISC와 비교하여 프로그램 크기가 3% 크고, 로드 스토어가 18% 크다. 따라서 32비트 BISC보다 20% 높은 데이터 전송 폭이 요구된다. ARM-7TDMI는 32비트 BISC에 비하여 25% 높은 데이터 전송 폭이 필요하다. 따라서 32비트 BISC는 16비트 축약 명령어 RISC 계열 마이크로 프로세서보다 20~25% 낮은 데이터 전송 폭을 가지면서 프로그램 크기도 3~10% 작다.

5. 결 론

마이크로 프로세서는 실장 제어 기기 분야에서 중대형 및 소형 컴퓨터에 이르기까지 광범위하게 사용되고 있으며, 반도체 기술의 급격한 발전으로 동작 속도도 1GHz에 조만간 이르게 될 전망이다. 이들 마이크로 프로세서에서 프로그램을 수행하기 위해서는 프로그램과 데이터를 메모리로부터 읽어 와야 한다. 그런데 메모리 동작 속도는 마이크로 프로세서 동작 속도에 크게 미치지 못하기 때문에 마이크로 프로세서와 메모리사이의 데이터 전송 폭이 시스템 성능을 제한하는 중요한 요인으로 작용하고 있다.

또한 실장 제어용 기기는 마이크로 프로세서와 메모리 및 입출력 장치가 하나의 반도체에 집적되는 경우가 많다. 반도체 가격을 낮추기 위해서는 메모리 크기를 줄여야 하며, 이를 위해서 또한 프로그램 크기가 작은 컴퓨터 구조에 대한 연구가 필요하다.

본 논문에서는 프로그램 크기가 작은 32비트 마이크로 프로세서 구조로 16비트와 32비트 2가지 길이 명령어를 가지는 가칭 2가지 길이 명령어 세트 컴퓨터(Bi-length Instruction Set Computer : BISC)를 제안하였다.

제안한 32비트 BISC는 16개의 32비트 범용 레지스타, 로드, 스토어 명령어 형식을 가지며, 오프셋과 상수 오퍼랜드 길이에 따라서 16비트와 32비트 2가지 길이 명령어를 가진다.

제안한 32비트 BISC는 FPGA 회로로 구현하여 1.8432 MHz에서 모든 기능이 정상적으로 동작하는 것을 확인하였고, 크로스 어셈블러와 크로스 C/C++ 컴파일러 및 명령어 시뮬레이터를 설계하였다. 자료 구조 및 수학 함수 프로그램을 C/C++ 언어로 작성하고, 이들을 크로스 컴파일러와 어셈블러를 사용하여 기계어로 번역하고, 이것을 시뮬레이터에서 동작시킨 결과와 IBM-PC에서 전용 컴파일러를 사용하여 얻은 결과를 비교하여 동작을 검증하였다.

제안한 BISC의 코드 밀도는 기존 RISC의 130~220%, 기존 CISC의 130~140%로 현격하게 높은 장점을 가진다. 또한 16비트 축약 명령어 RISC보다 프로그램 크기가 3~10% 작고, 로드 스토어 빈도가 15% 이상 낮다. 따라서 데이터 전송 폭을 적게 요구하므로 차세대 컴퓨터 구조로 적합하고, 프로그램 메모리 크기가 작아지므로 실장 제어용 마이크로 프로세서에 특히 적합하므로 폭 넓은 활용이 기대된다.

부 록 : 32비트 BISC 명령어 세트

Type 0 : Short ofzbitfset load/store

bit 15 = 0
bit 14-12 = Operation
bit 11-8 = Source/Destination register
bit 7-4 = Index register
bit 3-0 = Offset

Type 8 : Short index stack area 32 bit load/store
bit 15-12 = 1000
bit 11-8 = Source/Destination register
bit 7 = Operating code
bit 6-0 = Offset

Type 9 : Load 8 bit immediate data
bit 15-12 = 1001
bit 11-8 = Destination register
bit 7-0 = Sign extend 8 bit immediate data bit 7-0

Type 10-0 : 16 bit offset load/store
bit 15-12 = 1010
bit 11 = 0
bit 10-8 = Operation
bit 7-4 = Source/Destination register
bit 3-0 = Index register
*** Second word = Sign extend offset bit 15-0

Type 10-8 : Push/Pop register list
bit 15-12 = 1010
bit 11 = 1
bit 10-9 = Register bank
bit 8 = Operating code
bit 7-0 = Register list

Type 10-14 : Immediate data add/subtract stack pointer
bit 15-12 = 1010
bit 11-9 = 111
bit 8 = Immediate data length
bit 7 = Operating code
bit 6-0 = Immediate data bit 8-2
*** Second word = Immediate bit 24-9

Type 11-0 : Arithmetic/Logic immediate operation
bit 15-12 = 1011
bit 11 = 0
bit 10-8 = Operating code
bit 7-4 = Source/Destination register
bit 3-0 = 0000
*** Second word = Immediate data

Type 11-8 : Arithmetic/Logic register operation
bit 15-12 = 1011
bit 11 = 1
bit 10-8 = Operating code
bit 7-4 = Source/Destination register
bit 3-0 = Source register

Type 12 : Short immediate ADD/SUB/CMP
bit 15-12 = 1100
bit 11-8 = Source/Destination register
bit 7-6 = Operation
bit 5-0 = Immediate data

Type 13 : Conditional branch, JUMP and JAL
bit 15-12 = 1101
bit 11-8 = Conditional code
bit 7 = Offset length
bit 6-0 = Offset bit 7-1
*** Second word if bit 7 is 1 = Sign extend offset bit 23-8

Type 14-0 : Misc
bit 15-12 = 1110
bit 11-8 = 0000
bit 7-4 = Register or immediate data if needed
bit 3-0 = 0000 == 8 bit sign extend
bit 3-0 = 0001 == 16 bit sign extend
bit 3-0 = 0010 == Software Interrupt
bit 3-0 = 0011 == Test and set
bit 3-0 = 0100 == Register indirect JMP
bit 3-0 = 0101 == Register indirect JAL
bit 3-0 = 0110 == Jump indirect to LR
bit 3-0 = 1000 == Move to %ML
bit 3-0 = 1001 == Move to %MH
bit 3-0 = 1010 == Move from %ML
bit 3-0 = 1011 == Move from %MH
bit 3-0 = 1100 == Set status flag 15 to 0
bit 3-0 = 1101 == Clear status flag 15 to 0

Type 14-2 : Privileged instruction
bit 15-12 = 1110
bit 11-8 = 0010
bit 7-4 = Register or immediate data if needed
bit 3-0 = 0000 == Set status flag 31 to 16 [SET pos]
bit 3-0 = 0001 == Clear status flag 31 to 16 [CLR pos]
bit 3-0 = 0010 == Halt [HALT]

Type 14-3 : Move/LEA from/to stack pointer
bit 15-12 = 1110
bit 11-8 = 0011
bit 7-4 = Source/destination register
bit 3-0 = 0000 == Move to SP
bit 3-0 = 0001 == Move from SP
bit 3-0 = 0010 == LEA to SP
*** Second word = Sign extend immediate data bit 15-0
bit 3-0 = 0011 == LEA from SP
*** Second word = Sign extend immediate data bit 15-0

Type 14-4 : Move
bit 15-12 = 1110
bit 11-8 = 0100
bit 7-4 = Destination register
bit 3-0 = Source register

Type 14-5 : Load effective address
bit 15-12 = 1110
bit 11-8 = 0101

bit 7-4 = Destination register
bit 3-0 = Source register
*** Second word = Sign extend immediate data bit 15-0

Type 14-8 : Shift operation
bit 15-12 = 1110
bit 11 = 1
bit 10-9 = Operating code
bit 7-4 = Register
bit 8, 3-0 = Shift amount

Type 15-0 : Multiply/Divide
bit 15-12 = 1111
bit 11-10 = 00
bit 9-8 = Operation
bit 7-4 = Source 1 register
bit 3-0 = Source 2 register

Type 15-4 : 16 bit offset stack area load/store
bit 15-12 = 1111
bit 11-8 = 0100
bit 7 = 0
bit 6-4 = Operation
bit 3-0 = Source/Destination register
*** Second word = Sign extend offset bit 15-0

Type 15-8-0 : Command co-processor
bit 15-12 = 1111
bit 11 = 1
bit 10-8 = Co-processor unit number
bit 7-6 = 00
bit 5-0 = Command bit 5-0
*** Second word = Command bit 21 - 6

Type 15-8-8 : Check co-processor status
bit 15-12 = 1111
bit 11 = 1
bit 10-8 = Co-processor unit number
bit 7-5 = 100
bit 4-0 = Status bit number

Type 15-8-12 : Move to/from co-processor register
bit 15-12 = 1111
bit 11 = 1
bit 10-8 = Co-processor unit number
bit 7-6 = 11
bit 5 = Operating code
bit 4 = 0
bit 3-0 = Co-processor register number

참 고 문 헌

[1] D. Patterson, "Reduced Instruction Set Computer," Comm. ACM, Vol.28, No.1, pp.8-21, Jan. 1985.

[2] Dezso Sima *et al.*, "Superscalar Instruction Issue," IEEE Micro, pp.28-39, Oct. 1987.

[3] B. Gieseke *et al.*, "A 600MHz Superscalar RISC Microprocessor with out-of-order execution," ISSCC Digest Tech. Papers, pp.176-177, Feb. 1997.

[4] C. A. Maier *et al.*, "A 533MHz BiCMOS Superscalar RISC Microprocessor," IEEE Journal of Solid-State Circuits, Vol.32, No.11, pp.1625-1634, Nov. 1997.

[5] Charles F. Webb *et al.*, "A 400MHz S/390 Microprocessor," IEEE Journal of Solid-State Circuits, Vol.32, No.11, pp.1665-1675, Nov, 1997.

[6] Paul E. Gronowski *et al.*, "High-Performance Microprocessor Design," IEEE Journal of Solid-State Circuits, Vol.33, No.5, pp.676-686, May 1998.

[7] Doug Burger, "Limited Bandwidth to Affect Processor Design," IEEE Micro, pp.55-62, Dec. 1997.

[8] Manfred Schlett, "Trends in Embedded-Microprocessor Design," IEEE Computer, pp.44-50, Aug. 1998.

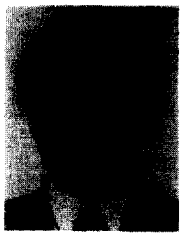
[9] S. Segars *et al.*, "Embedded Control Problems, Thumb, and the ARM7TDMI," IEEE Micro, pp.22-30, Oct. 1995.

[10] ftp://cair-archive.kaist.ac.kr/pub/gnu/egcs/releases/egcs-1.1b/egcs-1.1b.tar.gz.

[11] ftp://ftp.cygnus.com/pub/newlib/newlib-1.8.1.tar.gz.

[12] ftp://cair-archive.kaist.ac.kr/pub/gnu/released/libstdc++-2.8.1.tar.gz.

[13] ftp://cair-archive.kaist.ac.kr/pub/gnu/released/gdb-4.17.tar.gz.



조 경 연

e-mail : gycho@dolphin.pnu.ac.kr

1990년 인하대학교 공과대학 전자공학과 정보공학전공(공학박사)

1983년~1991년 삼보컴퓨터 기술연구소 책임연구원

1991년~현재 부경대학교 공과대학 컴퓨터멀티미디어공학부 부교수

1991년~현재 삼보컴퓨터 기술연구소 비상임 기술고문

1993년~현재 아시아디자인(주) 비상임 기술고문

1995년~현재 대흥전자(주) 비상임 기술고문

관심분야 : 전자계산기구조, ASIC 회로 설계, ASIC memory