論文99-36D-5-2

확장 명령어 32 비트 마이크로 프로세서에 관한 연구

(A Study on Extendable Instruction Set Computer 32 bit Microprocessor)

趙 璟 衍 *

(Gyoung-Yun Cho)

요 약

마이크로 프로세서의 동작 속도가 빨라지면서 메모리의 데이터 전송 폭이 시스템 성능을 제한하는 중요 인자가 되고 있다. 또한 CPU와 메모리 및 입출력회로가 하나의 반도체에 집적되는 실장 제어용 마이크로 프로세서의 가격을 낮추기 위해서 메모리 크기를 줄이는 것이 중요하다. 본 논문에서는 코드 밀도가 높은 32 비트 마이크로 프로세서 구조로 가칭 확장 명령어 세트 컴퓨터(Extendable Instruction Set Computer: EISC)를 제안한다. 32 비트 EISC는 16개의 범용 레지스타를 가지며, 16 비트 고정 길이 명령어, 짧은 오프셋 인덱스 어드래싱과 짧은 상수 오퍼랜드 명령어를 가지며, 확장 레지스타와 확장 프래그를 사용하여 오프셋 및 상수 오퍼랜드를 확장할 수 있다. 32 비트 EISC는 FPGA로 구현하여 1.8432MHz에서 모든 기능이 정상적으로 동작하는 것을 확인하였고, 크로스 어셈블러와 크로스 C/C++ 컴파일러 및 명령어 시뮬레이터를 설계하고 동작을 검증하였다. 제안한 EISC의 코드 밀도는 기존 RISC의 140-220%, 기존 CISC의 120-140%로 현격하게 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.

Abstract

The data transfer width between the microprocessor and the memory comes to a critical part that limits system performance since the data transfer width has been as it was while the performance of a microprocessor is getting higher due to its continuous development in speed. And it is important that the memory should be in small size for the reduction of embedded microprocessor's price which is integrated on a single chip with the memory and IO circuit. In this paper, a microprocessor tentatively named as Extendable Instruction Set Computer(EISC) is proposed as the high code density 32 bit microprocessor architecture. The 32 bit EISC has 16 general purpose registers and 16 bit fixed length instruction which has the short length offset and small immediate operand. By using an extend register and extend flag, the offset and immediate operand could be extended. The proposed 32 bit EISC is implemented with an FPGA and all of its functions have been tested and verified at 1.8432MHz. And the cross assembler, the cross C/C++ compiler and the instruction simulator of the 32 bit EISC have been designed and verified. This paper also proves that the code density of 32 bit EISC shows 140-220% and 120-140% higher code density than RISC and CISC respectively, which is much higher than any other traditional architectures. As a consequence, the EISC is suitable for the next generation computer architecture since it requires less data transfer width compared to any other ones. And its lower memory requirement will make embedded microprocessor more useful.

* 正會員, 釜慶大學校 컴퓨터멀티미디어工學部
(Division of Computer and multimedia engineering, Pukyong National University)

接受日字:1998年11月30日, 수정완료일:1999年4月22日

Ⅰ. 서 론

1970년대에 개발된 마이크로 프로세서는 제어 기기 분야 및 소형 컴퓨터에서 주로 사용되어 오다가 1980년대에 이르러 RISC(Reduced Instruction Set Computer)[1] 구조의 도입으로 중대형 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. 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 뉴먼 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(휴대용 정보 기기), 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 32 비트 RISC 명령어를 16 비트 명령어로 축약한 구조가 연구되었다[9]. ARM-7TDMI는 ARM-7의 16 비트 축약 명령어 구조이며, TR4101은 MIPS-R3000의 16 비트 축약 명령어 구조이다[8, 9]. 이들 16 비트 축약 명령어 RISC는 종래 RISC와의 호환성을 위하여 2가지 모드로 동작하므로 구조가 복잡하고, 16 비트 명령어에서는 8개의 레지스타만을 접근할 수 있으므로 성능이 크게 떨어지는 단점을 가진다.

본 논문에서는 코드 밀도가 높은 32 비트 마이크로 프로세서 구조로 가칭 확장 명령어 세트 컴퓨터(Extendable Instruction Set Computer: EISC)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]를 벤치마크 프로그램으로 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 16 비트 고정 길이 명령어를 가지며, 오프셋 및 상수 필드를 확장하는 32 비트 EISC의 명령어 세트를 설계한다. 32 비트 EISC는 코드 밀도를 더욱 높이기 위하여 레지스타 리스트 푸시, 팝 명령을 가지며, 파이프라인 제어를 하드웨어로 수행하여 불필요한 NOP 명령어를 제거하였다.

제안한 32 비트 EISC는 FPGA로 구현하여 1.8432MHz에서 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 EISC와 기존의 RISC 및 CISC의

코드 밀도를 비교하였다. 제안한 EISC의 코드 밀도는 기존 RISC의 140-220%, 기존 CISC의 120-150%로 기존 구조에 비하여 데이터 전송 폭을 크게 낮출 수 있으며, 프로그램 크기도 이에 비례하여 작아 진다.

제 2장에서는 MIPS-R3000 프로그램을 분석하고 이에 따른 32 비트 EISC 구조를 제안하며, 제 3장에서는 제안한 32 비트 EISC의 하드웨어 및 소프트웨어를 구현하고 제 4장에서는 기존의 RISC 및 CISC 구조와 비교하여 성능을 검증하며, 제 5장에서는 결론을 맺는다. 그리고 부록에 제안한 32 비트 EISC의 명령어 세트를 보인다.

Ⅱ. 32 비트 EISC 구조

1. 16개 범용 레지스타

MIPS-R3000은 34개의 32 비트 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 1에는 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

Table 1. Program characteristics according to various number of registers of MIPS-R3000.

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 1에서 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 레지스타 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 32 비트 EISC에서는 16개의 범용 레지스타를 사용한다. 이후의 프로그램 분석은 16개 레지스타를 사용하도록 변형한 MIPS-R3000 컴파일러를 사용한다.

2. 로드, 스토어 구조

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

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

Table 2. Instruction frequency of MIPS-R3000 with 16 general purpose registers.

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 2에서 변수 산술연산 명령어(addu, subu, and 등)는 3.33%로 빈도가 높지 않다. 이들 명령어에서 메모리 변수를 사용하는 빈도는 출현 빈도보다 높지 않다. 즉 메모리 연산 명령어의 출현 빈도가 작으므로 이를 위한 명령어는 정의하지 않는 것이 효율적이다.

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

3. 16 비트 고정 길이 확장 명령어

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

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

표 2에서 32 비트 로드, 스토어가 전체 로드, 스토어의 93,5%를 점유하고 있다. 따라서 로드, 스토어 명령의 특성을 분석하기 위해서 표 3에 'lw(load word), sw(store word)'의 사용 특성을 보인다.

표 3. 'lw, sw' 명령어 특성
Table 3. Characteristics of 'lw, sw' instruction.

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 3에서 'lw, sw' 명령어의 60.9%가 스택 포인터를 사용하고 있다. 이것은 지역 변수 및 전달 변수가 스택에 설정되기 때문이다. 인덱스 레지스타를 사용하는 경우에는 3 비트 오프셋으로 77%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.

또한 상수 오퍼랜드 명령어인 'li(load immediate)' 명령어는 표 3에서 2.9%를 차지하며 상수의 출현 빈도는 표 4와 같다.

표 4. 'li' 명령어에서 상수의 사용 빈도
Table 4. Frequency by constant size of 'li' instruction.

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

표 4로부터 'li' 명령어의 93.6%는 8 비트 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 가지는 명령어를 16 비트 길이로 정의하면서, 긴 길이의 오프셋과 상수 오퍼랜드를 가지는 명령어도 16 비트 길이 명령어의 조합으로 표현하기 위해서 32 비트 EISC에서는 32 비트 확장 레지스타(%ER)와 %ER 레지스타에 새로운 상수가 입력되었음을 나타내는 확장 프래그(E)를 설정한다. 또한 'LERI constant' 명령어를 다음과 같이 설정한다.

Instruction Mnemonics : LERI
Instruction Format : LERI constant
Instruction Representation :
bit 15-14 = 01
bit 13-0 = Constant data bit 13-0
Operation :
If (E flag is 0) Load %ER with sign extend constant
ELSE %ER = %ER << 14 + Constant
Set E flag

확장 프래그는 'LERI' 명령어를 수행하면 '1'이 되고, 다른 모든 명령어에서는 '0'이 된다. 로드, 스토어 명령어는 다음과 같이 정의한다.

```
Instruction Function : Load/Store
Instruction Representation :
        bit 15-14 = 00
        bit 13-12, 7 = Operation
            000 : Sign extend  8 bit load.
            001 : Sign extend 16 bit load.
            010 : 32 bit load.
            011 : Zero extend 8 bit load.
            100 :  8 bit store.
            101 : 16 bit store.
            110 : 32 bit store.
            111 : Zero extend 16 bit load.
    bit 11-8 = Source/Destination register.
                %R0 thru %R15.
    bit 6-4 = Offset bit 2-0 if  8 bit load/store
              Offset bit 3-1 if 16 bit load/store
              Offset bit 4-2 if 32 bit load/store
    bit 3-0 = Index register. %R0 thru %R15.

Effective operand address : EA
    If ( E flag is 0 )
      EA = Zero extend offset + Index Register
    If ( E flag is 1 )
      if ( 32/16 bit load/store)
        EA = %ER <<4 + Offset + Index Register
      if (  8 bit load/store)
        EA = %ER << 3 + Offset + Index Register
```

이와 같이 확장 프래그에 의하여 긴 오프셋과 긴 상수 오퍼랜드로 확장하는 구조를 채용하면 모든 명령어를 16 비트 고정 길이로 표현하는 것이 가능하다.

4. 스택 명령어

표 3에서 'lw, sw' 명령어의 오프셋 길이가 스택 포인터와 인덱스 레지스타를 사용하는 경우에 현격한 차이를 보이고 있다. 표 3으로부터 스택 포인터를 사용하는 경우에 'lw, sw' 오프셋은 5 비트 이상이 필요하다. 32 비트 EISC에서는 이러한 특성을 감안하여 7 비트 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 32 비트 EISC에서는 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 2에서 'addiu(add immediate)' 명령어의 출현 빈도는 7.53%이다. 이중에서 스택 포인터를 사용하는 빈도는 35%이며, 이중 7 비트 상수의 빈도가 99.1%이다. 그러므로 32 비트 EISC에서는 7 비트 상수를 스택 포인터에 더하고, 빼는 명령어를 정의한다.

5. 기타 명령어

표 2에서 조건 분기 명령어의 빈도가 6.69%이다. 이를 효율적으로 지원하기 위해서 32 비트 EISC에서는 캐리, 사인, 제로, 오버플로우의 4개 플래그를 사용하며 이들을 조합하여 14가지 조건 분기 명령어를 만든다. 조건 분기 명령어의 오프셋은 9 비트로 설정하며, 확장 레지스타를 이용하여 32 비트까지 확장한다.

논리 산술 연산 명령어는 48.5%가 2 오퍼랜드 명령어이고 51.4%가 3 오퍼랜드 명령어로 조사되었다. 3 오퍼랜드 명령어는 'move' 명령과 2 오퍼랜드 명령어의 조합으로 표현할 수 있으므로 32 비트 EISC의 논리 산술 연산 명령어는 2 오퍼랜드 형식을 가진다.

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 32 비트 레지스타(%ML, %MH)를 설정한다.

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 확장 레지스타를 이용하여 20 비트 또는 34 비트 길이를 가진다.

Ⅲ. 하드웨어 및 소프트웨어 구현

표 5에 32 비트 EISC의 레지스타 구성을 보인다.

표 5. 32 비트 EISC의 레지스타 구성
Table 5. 32 bit EISC register.

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
ER	32 bit extension 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 19 = Extension flag (E)
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.

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

설계한 FPGA 32 비트 EISC는 LED 제어기, RS-232C 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 6에는 크로스 소프트웨어 구현 현황을 보인다.

표 6. 32 비트 EISC 크로스 소프트웨어
Table 6. Cross software of 32 bit EISC.

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.

Ⅳ. 성능 평가

코드 밀도는 프로그램 크기에 반비례한다. 상대 코드 밀도(Relative Code Density : RCD)를 대상 마이크로 프로세서에 대한 32 비트 EISC의 상대적인

코드 밀도로 정의하면 식 (1)과 같이 표현된다.

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

성능 평가를 위하여 표 6의 환경으로 32 비트 EISC 및 기존의 마이크로 프로세서들에 대한 크로스 C/C++ 컴파일러를 작성하였다. 객관적인 성능 평가를 위해서 C 라이브러리 NEWLIB-1.8.1[11]과 C++ 라이브러리 LIBSTDC++-2.8.1[12]를 벤치마크 프로그램으로 선정하였다. NEWLIB-1.8.1은 Cygnus 사에서 실장 제어용으로 개발한 C 라이브러리로 ANSI C 라이브러리와 SUN 사에서 SunPro 워크스테이션에서 사용하고 있는 단정도 및 배정도 실수 함수 라이브러리로 구성되어 있다. LIBSTDC++는 SGI 사의 STL(C++ Standard Template Library)로 C++ 입출력 함수와 수학 함수 라이브러리이다. 이들은 C/C++로 작성된 대표적인 프로그램이고, 컴파일된 기계어 크기는 MIPS-R3000에서 381,560 바이트이며, 기계어 구성은 표-2와 같다.

표-7에 벤치마크 프로그램을 사용하여 구한 32 비트 EISC에 대한 기존 마이크로 프로세서들의 상대 코드 밀도를 보인다.

표 7에서 대표적인 32 비트 RISC인 MIPS-R3000의 상대 코드 밀도가 1.66이다. 즉, 32 비트 EISC의 코드 밀도가 MIPS-R3000보다 66% 높으며, 따라서 MIPS-3000의 프로그램 크기가 32 비트 EISC보다 66%크다는 것을 나타내고 있다. 실장 제어 기기 분야에서 많이 쓰이는 ARM-7의 상대 코드 밀도는 1.64이다. 따라서 ARM-7의 프로그램 크기는 32 비트 EISC보다 64% 커진다. 표 7로부터 RISC의 상대 코드 밀도는 1.4-2.2이며 따라서 기존 RISC의 프로그램 크기는 32 비트 EISC보다 40-120% 크다.

또한 MC68000 및 I80386 등 기존 CISC의 상대 코드 밀도는 1.2-1.4로 나타나고 있다. 즉 기존 CISC의 프로그램 크기는 32 비트 EISC보다 20-40% 크다.

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

표 7. 32 비트 EISC에 대한 상대 코드 밀도
Table 7. Relative code density with 32 bit EISC.

Processor	Relative Code Density	Processor	Relative Code Density
32 bit EISC	1.00	MC88000	1.59
MIPS-R3000	1.66	MC5200	1.43
TR4101	1.07	MC68000	1.35
MIPS-R4000	1.46	MC68332	1.32
MIPSTX-39	1.58	MC68020	1.32
ARM-7	1.64	MN10300	1.17
ARM-7TDM	1.13	Pentium	1.39
PowerPC 601	1.92	I80960	1.68
SPARC V8	1.38	ARC	2.19
SPARCLITE	1.78	SH-3	1.38
PA-RISC	2.22	V850	1.21
Alpha-RISC	2.23	M32R	1.44

표 8. 32 비트 EISC와 16 비트 축약 명령어 RISC의 비교
Table 8. Comparition table between 32 bit EISC and 16 bit compressed RISC.

	32 bit EISC	TR4101	ARM-7TDMI
Relative Code Density	1.00	1.07	1.13
Load/Store	30.2%	48.3%	46.5%

16 비트 축약 명령어 RISC인 ARM-7TDMI 및 TR4101의 상대 코드 밀도는 각각 1.13과 1.07로 32 비트 EISC보다 프로그램 크기가 조금 크고, 범용 레지스타가 8개이므로 로드, 스토어가 증가한다. 32 비트 EISC와 16 비트 축약 명령어 RISC의 비교표를 표 8에 보인다.

로드 스토어 명령은 메모리 읽기 쓰기를 동반하므로 데이터 전송 폭을 증가시킨다. 표 8로부터 TR4101

은 32 비트 EISC와 비교하여 프로그램 크기가 7% 크고, 로드 스토어가 18% 크다. 따라서 32 비트 EISC보다 25% 높은 데이터 전송 폭이 요구된다. ARM-7TDMI는 32 비트 EISC에 비하여 30% 높은 데이터 전송 폭이 필요하다. 따라서 32 비트 EISC는 16 비트 축약 명령어 RISC 계열 마이크로 프로세서보다 20-30% 낮은 데이터 전송 폭을 가지면서 프로그램 크기도 5-15% 작다.

V. 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. 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.

이러한 필요성에 따라 본 논문에서는 프로그램 크기가 작은 32 비트 마이크로 프로세서 구조로 가칭 확장 명령어 세트 컴퓨터(Extendable Instruction Set Computer: EISC)를 제안하였다.

제안한 32 비트 EISC는 16개의 32 비트 범용 레지스타, 로드, 스토어 명령어 형식, 16 비트 고정 길이 명령어, 짧은 길이 오프셋 어드래싱, 짧은 길이 상수 오퍼랜드 명령어를 가지며 확장 레지스타와 확장 프래그를 채용하여 오프셋 및 상수 오퍼랜드를 확장할 수 있다.

제안한 32 비트 EISC는 FPGA 회로로 구현하여 1.8432MHz에서 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 EISC의 코드 밀도는 기존 RISC의 140-220%, 기존 CISC의 120-140%로 현격하게 높은 장점을 가진다. 또한 16 비트 축약 명령어 RISC보다 프로그램 크기가 5-15% 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.

부록 : 32 비트 EISC 명령어 세트

Type 0 : Load/store
- bit 15-14 = 00
- bit 13-12, 7 = Operation
- bit 11-8 = Source/Destination register
- bit 6-4 = Offset
- bit 3-0 = Index register

Type 4 : Load extension register and set E
- bit 15-14 = 01
- bit 13-0 = Immediate data bit 13-0

Type 8 : Stack area 32 bit load/store
- bit 15-12 = 1000
- bit 11-8 = Source/Destination register
- bit 7 = Operating code
- bit 6-0 = Offset bit 8-2

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

Type 10 : Load effective address
- bit 15-12 = 1010
- bit 11-8 = Destination register
- bit 7-4 = Immediate data bit 3-0
- bit 3-0 = Source register

Type 11-0 : Push/Pop register list
- bit 15-12 = 1011

bit 11 = 0
bit 10-9 = Register bank
bit 8 = Operating code
bit 7-0 = Register list

Type 11-6 : Add/subtract stack pointer
bit 15-12 = 1011
bit 11-8 = 0110
bit 7 = Operating code
bit 6-0 = Immediate data bit 8-2

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

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 7-0 = PC relative offset

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
bit 3-0 = 0001 == Clear status flag 31 to 16
bit 3-0 = 0010 == Halt

Type 14-4 : Load effective address from/to SP
bit 15-12 = 1110
bit 11-9 = 010
bit 8 = Operation
bit 7-4 = Source/Destination register
bit 3-0 = Immediate data bit 3-0

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 : Stack area load/store
bit 15-12 = 1111
bit 11-10 = 00
bit 9-7 = Operation
bit 6-4 = Offset
bit 3-0 = Source/Destination register

Type 15-4 : Multiply/Divide
bit 15-12 = 1111
bit 11-10 = 01
bit 9-8 = Operation
bit 7-4 = Source 1 register
bit 3-0 = Source 2 register/immediate data

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

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. 1997.

[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.

저 자 소 개

趙 璟 衍(正會員) 第 33卷 B編 第 4號 參照