EISC : 실장 제어 마이크로프로세서

조 경 연*

1. Introduction

마이크로프로세서는 1960년대에 최초로 4 비트 계열이 발표되면서 모든 가전제품, 산업기계 및 자동화기기에 필수적인 부품으로 사용되고 있다. 특히 마이크로프로세서 개발은 고도의 논리회로 설계 기술과 반도체 공정 기술 그리고 소프트웨어 기술이 종합적으로 결합되어야 하므로 기술의 파급효과가 크다. 이러한 이유로 미국, 일본 및 유럽 등 선진국의 대부분 전자회사 및 반도체회사는 마이크로프로세서 개발에 많은 투자를 계속하였고 지금도 많은 연구비를 투입하고 있다[1-8].

한편 2000년대 들어서 시스템을 하나의 반도체에 집적하는 SoC(System on Chip) 기술이 개발되었다. SoC는 가전기구, 산업기계, 자동화기계 및 IT 제품 등의 필수적인 핵심 기술로 자리잡아 가고 있다. 모든 SoC는 두뇌 역할을 수행하는 마이크로프로세서를 포함하고 있다. SoC는 마이크로 프로세서와 메모리 및 입출력 장치가 하나의 반도체에 집적된다. 그런데 반도체 가격은 반도체 크기에 따라 결정되며, SoC에서 가장 넓은 실리콘 면적을 차지하는 것은 프로그램 메모리이다. 따라서 반도체 가격을 낮추기 위해서는 메모리 크기를 줄여야 하며, 이를 위해서 코드 밀도가 높은 마이크로프로세서가 필수적이다.

이와 더불어 차세대 실장 제어 마이크로프로세서에 대한 많은 연구 및 개발이 진행되고 있다. 현재 연구되고 있는 3대 기술은 scalable, configurable, application specific extendable로 요약할 수 있다[9,10].

실장 제어 기기는 성능대비 가격의 폭이 넓다. 즉, 8/16 비트 마이크로프로세서부터 64 비트 마이크로프로세서에 이르기까지 다양한 성능 및 가격대의 마이크로프로세서가 응용제품에 따라서 요구된다. 이들 광범위한 성능대비 가격의 마이크로프로세서를 하나의 architecture로 구현한 것을 scalable 마이크로프로세서라고 한다. Scalable 마이크로프로세서를 구현하기 위해서는 16/32/64 비트 마이크로프로세서가 하나의 architecture를 가지는 것이 필요하다.

SoC는 VHDL 등 하드웨어 설계 툴을 사용하여 IP(Intellectual Property)를 설계하고, 필요한 IP를 연결하여서 하나의 시스템을 구성한다. SoC에 필요한 다양한 기능 세트를 가진 IP를 설계하고, SoC 사양에 맞는 정확한 IP를 생성할 수 있도록 한 것을 configurable IP라고 한다. Configurable IP의 개념을 마이크로프로세서에 적용한 것이 configurable 마이크로프로세서이다.

Configurable 마이크로프로세서는 가격대비 성능에 영향을 미치는 기능들을 사용자가 선택하

* 부경대학교 전자컴퓨터정보통신공학부 교수

여 구성할 수 있도록 제공하므로 최적의 시스템 구현을 가능하게 한다. Configurable 마이크로프로세서는 VHDL 등으로 설계되어야 하므로 명령어 세트가 단순하고, 하드웨어가 간단해야 한다.

실장 제어 마이크로프로세서를 특정한 응용분야에 효율적으로 적용하려면 특수한 기능을 가지도록 확장할 수 있어야 한다. 예를 들면 DSP, SIMD, 부동소숫점 연산기, 공개키암호, Java 프로세서 등의 기능은 멀티미디어 기기, 무선 전화기, 네트워크서버, 보안장비, Smart 카드 등의 응용분야에 필요한 기능들이다. 특정한 응용 분야에 적합한 기능을 가지도록 확장할 수 있는 마이크로프로세서를 application specific extendable 마이크로프로세서라고 한다. 이를 위해서는 확정성이 뛰어난 구조를 가지는 마이크로프로세서가 요구된다.

코드 밀도가 높으면서 scalable, configurable, application specific extendable 3 가지 차세대 기능을 가지는 마이크로프로세서를 구현하는 것은 종래의 RISC/CISC architecture로는 적합하지 않다.

RISC(Reduced Instruction Set Computer) architecture는 코드 밀도가 낮은 문제점이 있다. 또한 32 비트 고정 길이 명령어를 사용하므로 16 비트 마이크로프로세서를 만드는 것은 적합하지 않으며, 오퍼랜드 길이가 제한되므로 64 비트 마이크로프로세서로도 적합하지 않다. 한편 CISC (Complex Instruction Set Computer) architecture는 하드웨어가 복잡하므로 SoC에 적합하지 않으며, 오퍼랜드 길이에 따라서 여러 개의 명령어를 만들어야 하므로 명령어 세트가 복잡해진다. 따라서 16 비트 마이크로프로세서는 가능하지만 32/64 비트 마이크로프로세서를 설계하는 것은 비효율적이 된다.

반면에 EISC(Extended Instruction Set Computer) architecture는 16 비트 고정 길이 명령어 세트를 가지면서 필요에 따라서 오퍼랜드 길이를 확장하는 구조이다. 16/32/64 비트 마이크로프로세서는 모두 동일한 오피-코드를 가지며 단지 오퍼랜드 길이만이 다르다. 따라서 EISC architecture는 동일한 구조를 가지는 16/32/64 비트 마이크로프로세서를 설계하는 것이 가능하다. 또한 코드 밀도가 높으며, 명령어 세트가 단순하여 하드웨어가 간단하며, 확장성이 뛰어나다[11-13]

2. Microprocessor Architecture

2.1 Architecture, micro architecture and realization

Architecture는 설계 사상을 종합적으로 의미하는 것으로 명령어 세트에 의하여 그 사상이 잘 표현되어 진다. Micro architecture는 implementation이라고도 부르는데 architecture를 구현하는 실제적인 방법을 의미한다. Realization은 이를 물리적으로 제작하는 기법을 말한다. 따라서 하나의 architecture에 여러 개의 micro architecture가 존재할 수 있으며, 하나의 micro architecture에 여러 개의 realization이 존재할 수 있다.

이들을 간략하게 표현하면 다음과 같다.

Architecture : The architecture of a computer system can be defined as its functional appearance to its immediate users.

Micro architecture : The logic structure that gives shape to the architecture.

Realization : A concrete version of the micro architecture.

예를 들면 8086, 80286, 80386, 80486 등은 모두 동일한 architecture라고 한다. 또는, 8086/80286과 80386/80486/pentium으로 나누어서 후자를 386 architecture라고 구분하기도 한다. 이것은 이들이 모두 동일한 명령어 세트를 가지기 때문이

다. 물론 각각의 기종에 따라서 명령어 세트에 약간의 차이가 있지만, 사상은 동일하다. 8086, 80286, … pentium을 구분하는 것은 데이터 버스 폭, 파이프라인 구조, 캐시, 메모리관리유니트(MMU) 등 micro architecture가 다르기 때문이다. 또한 80386이라는 micro architecture에 대하여 NMOS, CMOS 버전이 있으며 또한 설계 법칙(design rule)에 따라서 동작 주파수가 다르다. 이것은 realization이 다른 예이다.

2.2 RISC and CISC

RISC는 1980년대 중반에 연구, 개발된 architecture를 의미한다.

RISC의 특징으로는

1) 고정 길이 명령어 - 모든 명령어의 길이가 동일하다.

2) 로드/스토어 방식 - 메모리와 레지스터 사이에 데이터를 교환하는 명령어는 load/store 명령어에 국한한다.

3) 간단한 명령어 세트 - 컴파일러에서 자주 사용하는 명령어만을 가진다. 따라서 하드웨어가 단순하다.

등이 있다.

RISC 마이크로프로세서로 대표적인 것이 MIPS-3000, SPARC, PowerPC, Alpha-RISC 등이다.

CISC는 RISC가 출현하면서 이것과 구분하기 위하여 RISC를 제외한 architecture를 통칭한다. 일반적으로 M68000, I80386 등을 CISC로 분류한다.

한편 RISC와 CISC의 중간적인 성격을 가지는 부류가 있는데, RISC로 분류하지만 CISC의 특징을 많이 가지고 있다. 이러한 부류에 속하는 것이 ARM, V800, MN10300, Tensilica, TriCore 등이다.

2.3 Component of instruction

컴퓨터 architecture를 가장 잘 나타내는 부분이 명령어 세트이다. 명령어는 동작의 종류를 나타내는 오피-코드와 동작을 받는 객체를 나타내는 오퍼랜드로 구성되어 있다.

예를 들어 ADD 명령어를 표현하면 'ADD %R0, %R1'이 되는데, 여기서 'ADD'는 더하는 동작을 나타내는 오피-코드이고, 더해지는 객체인 레지스타 %R0와 %R1는 오퍼랜드이다. 이 명령어를 기계어 코드로 표현하면 다음과 같다.

```
1010 0010  0000  0001 ; 16 bit machine code
---------- ----  ----
Op-Code    %R0   %R1
```

모든 마이크로프로세서는 공통적인 오피-코드를 가지고 있다. 즉,

Load/Store - 메모리와 레지스타간에 데이터 전송

Computation - Add, Subtract, Compare, Logical AND/OR/XOR, Shift 등 연산

Branch/Call - 프로그램 흐름을 변경

Move - 레지스타간의 데이터 전송

등이 있다.

반면에 오퍼랜드는 마이크로프로세서 종류에 무관하게 다음과 같은 종류가 있다.

레지스타 - 범용 목적의 일반 레지스타를 지칭한다.

오프셋-메모리 주소를 계산할 때는 인덱스 레지스타에 저장된 주소에 오프셋을 더하여 산출한다. 이러한 주소 계산 방식을 인덱스 주소 지정 방식이라고 한다. 또한 branch/call 명령어에서는 현재 프로그램 위치로부터 오프셋만큼 떨어져 있는 위치로 제어를 변경한다. 이것은 PC 상대 주소 지정 방식이라고 한다.

즉치 - 명령어가 상수를 포함하는 경우이다. 예를 들면 'ADD %R0, 155'는 'R0 레지스타에 상수 155를 더하여 R0 레지스타에 저장하라'라는 명령어이다.

2.4 Instruction set difference between RISC and CISC

명령어 세트를 구성하는데 있어서 오퍼랜드를 표현하는 방식이 RISC와 CISC에서 차이가 있다. 즉치 ADD 명령어에서 ADD 오피-코드 필드 길이를 8 비트라 가정하고,

ADD %R0, 10; 8 bit immediate data
ADD %R0, 1000; 16 bit immediate data
ADD %R0, 100000; 32 bit immediate data

과 같이 즉치 오퍼랜드 필드의 길이가 8/16/32 비트로 구분된다고 하면, CISC에서는 3 종류의 ADD 명령어가 필요하다. 즉,

ADD.B - 8 bit Op-Code, 8 bit Operand - 16 bit length instruction
ADD.S - 8 bit Op-Code, 16 bit Operand - 24 bit length instruction
ADD.W - 8 bit Op-Code, 32 bit Operand - 40 bit length instruction

이와 같이 하나의 즉치 ADD 명령어가 3개의 서로 다른 오피-코드를 가지므로 CISC에서는 명령어의 수가 많아지게 되는 문제점이 있다.

한편 RISC에서는 명령어 수를 줄이기 위해서 오직 한가지 길이 오퍼랜드만을 가진다. 즉,

ADD - 8 bit Op-Code, 16 bit Operand - 24 bit length instruction

이러한 RISC에서는 오퍼랜드 길이와 무관하게 모든 명령어는 항상 24 비트로 표현해야 하므로 프로그램 크기가 증가하는 문제가 있다.

특히 명령어의 출현 빈도를 분석한 결과 즉치 데이터는 8 비트 길이 이하가 90% 이상으로 조사되고 있다. 따라서 RISC에서는 대부분의 경우에서는 프로그램 크기가 불필요하게 증가되고 있다. 또한 32 비트 오퍼랜드가 필요한 경우에는 여러 개의 명령어로 이를 구현해야 하므로 더욱 비효율적이 된다.

이에 더하여 실질적으로 3 바이트(24 비트) 길이 명령어는 컴퓨터 메모리가 32 또는 64 비트 길이 단위로 구성되어 있으므로 대단히 비효율적이다. 이러한 점 때문에 RISC에서는 모든 명령어를 32 비트 길이로 구성한다. 따라서 프로그램 크기의 비효율성은 더욱 증대된다.

2.5 Fixed length and variable length Instruction

CISC 마이크로프로세서는 오퍼랜드 필드 길이에 따라서 다양한 길이의 명령어를 가진다.

80×86은 8개의 레지스타를 가진다. 따라서 레지스타 오퍼랜드를 표현하기 위해서는 3 비트가 필요하다. 그러므로 MOVE 명령어는 오피-코드에 2 비트, 소스 레지스타에 3 비트, 목적 레지스타에 3 비트를 할당하면 8 비트 길이 명령어로 구성할 수 있다. 한편 ADD 명령어는 오퍼랜드 길이에 따라서 다수개의 오피-코드를 가져야 하므로 오피-코드 길이는 8-16 비트가 된다. 여기에 오퍼랜드 필드가 더해져야 한다. 이러한 관계로 80x86 명령어 세트는 8/16/24/32/40/48/56 비트 등의 다양한 길이로 구성된다.

680×0은 16개의 레지스타를 가지므로 MOVE 명령어도 8 비트 길이로 표현할 수 없다. 그러므로 680×0 명령어 세트는 16/32/48/64/80/96 비트 등의 다양한 길이로 구성된다.

이러한 가변 길이 명령어는 오퍼랜드 길이에 따라서 적합한 명령어를 사용하므로 프로그램 크기가 작아지는 장점이 있지만, 명령어의 수가 증가하므로 하드웨어가 복잡해지고 이에 따라서 동작 속도가 낮아지며, 전력 소모는 증가하는 문제점을 가진다.

가변 길이 명령어를 가지는 마이크로프로세서로는 M680x0, I80x86, Panasonic의 MN10300 등이 대표적이다.

반면 RISC 형태의 마이크로프로세서는 대부분 32 비트 고정 길이 명령어 세트를 가진다. 이러한 고정 길이 명령어는 대부분의 경우에 있어서 프로그램 크기가 불필요하게 커지므로 메모리 효율이 저하되는 단점과 긴 길이 오퍼랜드를 필요로 하는 명령은 다수개의 명령어를 사용하여 구현해야 하는 단점이 있다. 반면에 모든 명령어 길이가 동일하므로 하드웨어가 단순하여 지고, 이에 따라서 동작 속도를 높일 수 있다.

32 비트 고정 길이 명령어 세트를 가지는 마이크로프로세서는 MIPS-3000/4000, SPARC, ARM, PowerPC, Alpha-RISC, PA-RISC 등이 대표적이다.

한편 프로그램 실행 시간은 '(프로그램을 구성하는 명령어의 수 : N) * (하나의 명령어를 수행하는 데 필요한 clock 수 : T) / (동작 수파수 : F)'로 결정된다. RISC는 CISC에 비하여 N가 증가하지만 T가 감소하며 F가 증가하므로 결과적으로 프로그램 실행시간이 짧아진다. 즉, 성능이 증가하면서 하드웨어도 단순한 장점이 있다.

2.6 16 bit fixed length Instruction

RISC의 32 비트 고정 길이 명령어는 프로그램 길이를 불필요하게 증가시키므로 한정된 크기의 메모리를 사용하는 실장 제어 시스템 등에는 적합하지 않다.

이러한 관계로 Hitachi사의 SH 시리즈와 Motorola사의 M-Core는 16 비트 고정 길이 명령어 세트를 가진다.

16 비트 고정 길이 명령어 세트에서 가장 문제가 되는 것은 오프셋과 즉치 오퍼랜드의 길이가 긴 명령어를 표현할 수 없다는 점이다. 이를 위하여 PC 상대 즉치 데이터 적재 명령어를 도입하였다.

즉, 'ADD %R0, 10000'이라는 명령어는 16 비트 길이 즉치 데이터 필드를 필요로 하는데, 명령어 길이가 16 비트로 고정되어 있으면 이를 구현할 수가 없다. 따라서

```
        Load    (%PC, offset), %R1
        ADD     %R0, %R1
        ..............
.L10000 .int    10000
```

와 같이 2개의 16 비트 명령어와 32 비트 데이터로 구현하였다.

이러한 방식에서는 'load' 명령어의 수행이 완료되어야 R1 레지스타 값이 결정되고, 이를 받아서 'ADD' 명령어를 수행할 수 있다. 이것은 파이프라인에서 데이터 의존이 항상 발생하므로 성능을 저하시키는 결정적인 단점을 가지고 있다. 또한 캐시는 블럭 단위로 관리되는 데, 위 프로그램 예에서 '.int 10000'이라는 데이터는 캐시의 블럭 단위 관리 특성 때문에 명령어 캐시에 저장된다. 한편 'Load' 명령어는 데이터 캐시를 사용하므로 '.int 10000'은 데이터 캐시에도 저장된다. 즉, 동일한 데이터가 명령어/데이터 캐시 두 곳에 저장되므로 캐시 효율을 저하시키며 이것은 성능 저하를 초래한다.

한편 ARM-Thumb, MIPS-16에서는 레지스터 수를 8개로 제한하고 16 비트 고정 길이 명령어 세트를 사용하고 있다. SH 시리즈와 M-Core는

이러한 가변 길이 명령어는 오퍼랜드 길이에 따라서 적합한 명령어를 사용하므로 프로그램 크기가 작아지는 장점이 있지만, 명령어의 수가 증가하므로 하드웨어가 복잡해지고 이에 따라서 동작 속도가 낮아지며, 전력 소모는 증가하는 문제점을 가진다.

가변 길이 명령어를 가지는 마이크로프로세서로는 M680x0, I80x86, Panasonic의 MN10300 등이 대표적이다.

반면 RISC 형태의 마이크로프로세서는 대부분 32 비트 고정 길이 명령어 세트를 가진다. 이러한 고정 길이 명령어는 대부분의 경우에 있어서 프로그램 크기가 불필요하게 커지므로 메모리 효율이 저하되는 단점과 긴 길이 오퍼랜드를 필요로 하는 명령은 다수개의 명령어를 사용하여 구현해야 하는 단점이 있다. 반면에 모든 명령어 길이가 동일하므로 하드웨어가 단순하여 지고, 이에 따라서 동작 속도를 높일 수 있다.

32 비트 고정 길이 명령어 세트를 가지는 마이크로프로세서는 MIPS-3000/4000, SPARC, ARM, PowerPC, Alpha-RISC, PA-RISC 등이 대표적이다.

한편 프로그램 실행 시간은 '(프로그램을 구성하는 명령어의 수 : N) * (하나의 명령어를 수행하는 데 필요한 clock 수 : T) / (동작 수파수 : F)'로 결정된다. RISC는 CISC에 비하여 N가 증가하지만 T가 감소하며 F가 증가하므로 결과적으로 프로그램 실행시간이 짧아진다. 즉, 성능이 증가하면서 하드웨어도 단순한 장점이 있다.

2.6 16 bit fixed length Instruction

RISC의 32 비트 고정 길이 명령어는 프로그램 길이를 불필요하게 증가시키므로 한정된 크기의 메모리를 사용하는 실장 제어 시스템 등에는 적합하지 않다.

이러한 관계로 Hitachi사의 SH 시리즈와 Motorola사의 M-Core는 16 비트 고정 길이 명령어 세트를 가진다.

16 비트 고정 길이 명령어 세트에서 가장 문제가 되는 것은 오프셋과 즉치 오퍼랜드의 길이가 긴 명령어를 표현할 수 없다는 점이다. 이를 위하여 PC 상대 즉치 데이터 적재 명령어를 도입하였다.

즉, 'ADD %R0, 10000'이라는 명령어는 16 비트 길이 즉치 데이터 필드를 필요로 하는데, 명령어 길이가 16 비트로 고정되어 있으면 이를 구현할 수가 없다. 따라서

```
        Load    (%PC, offset), %R1
        ADD     %R0, %R1
                ..............
.L10000 .int    10000
```

와 같이 2개의 16 비트 명령어와 32 비트 데이터로 구현하였다.

이러한 방식에서는 'load' 명령어의 수행이 완료되어야 R1 레지스타 값이 결정되고, 이를 받아서 'ADD' 명령어를 수행할 수 있다. 이것은 파이프라인에서 데이터 의존이 항상 발생하므로 성능을 저하시키는 결정적인 단점을 가지고 있다. 또한 캐시는 블럭 단위로 관리되는 데, 위 프로그램 예에서 '.int 10000'이라는 데이터는 캐시의 블럭 단위 관리 특성 때문에 명령어 캐시에 저장된다. 한편 'Load' 명령어는 데이터 캐시를 사용하므로 '.int 10000'은 데이터 캐시에도 저장된다. 즉, 동일한 데이터가 명령어/데이터 캐시 두 곳에 저장되므로 캐시 효율을 저하시키며 이것은 성능 저하를 초래한다.

한편 ARM-Thumb, MIPS-16에서는 레지스터 수를 8개로 제한하고 16 비트 고정 길이 명령어 세트를 사용하고 있다. SH 시리즈와 M-Core는

레지스터가 16개이므로 명령어 구성이 복잡하여지는데, 이러한 단점을 개선하는 효과가 있다. 그렇지만 PC 상대 즉치 데이터 적재 명령어를 사용하므로 동일한 문제점을 가지고 있으며, 레지스터 수가 작으므로 로드/스토어 명령어의 사용 빈도가 증가하므로 성능이 저하되고, 전력소모가 증가하는 추가적인 단점이 있다.

2.7 Limited variable length Instruction

32 비트 고정 길이 명령어는 프로그램 크기가 증가하는 단점이 있고, 16 비트 고정 길이 명령어는 성능이 낮은 단점이 있다.

이러한 관계로 NEC V800, Infinion의 TriCore 등에서는 16 비트와 32 비트의 두 가지 길이를 가지는 제한 가변 길이 명령어 세트를 가진다. 즉, MOVE, 레지스타 연산 명령어 등은 16 비트 길이 명령어로, 오프셋 필드를 필요로 하는 load/store/branch 명령어 및 즉치 데이터 필드를 가지는 연산 명령어는 32 비트 길이 명령어로 구성한다.

또한 Tensilica는 오퍼랜드 길이를 보다 제한하여서 16 비트와 24 비트의 두 가지 길이 명령어로 구성하였다.

이러한 제한 가변 길이 명령어는 32 비트 고정 길이 명령어에 비하여 프로그램 크기를 줄이는 장점이 있지만 오퍼랜드 길이가 16 비트보다 길어지면 여러 개의 명령어로 구현해야 하는 단점을 계속 가지고 있다. 또한 가변 길이 명령어에 따라서 하드웨어가 복잡해지는 단점을 추가로 가지고 있다. 특히 Tensilica에서는 명령어 길이가 16 비트와 24 비트로 되어 있는 반면에 데이터 버스는 16 또는 32 비트 폭으로 되어 있으므로 하드웨어 복잡도가 크게 증가하며 이에 따른 성능 저하가 커진다.

한편 ARC와 I80960은 32 비트와 64 비트 길이 명령어로 구성되어 있다. ARC 명령어 세트는 MIPS-3000과 유사하다. MIPS-3000에서는 32 비트 길이 오퍼랜드를 수용할 수 없으므로 이를 해결하기 위하여 ARC에서는 64 비트 길이 명령어를 사용한다.

이렇게 64 비트 길이 명령어를 사용하면 모든 길이의 오퍼랜드를 표현할 수 있지만 두 가지 길이 오퍼랜드를 지원하기 위해서 하드웨어가 복잡해지는 문제점이 있다.

3. EISC architecture

3.1 EISC has 16 bit fixed length instruction set

마이크로프로세서의 architecture는 오퍼랜드를 표현하는 방법에 따라서 구분할 수 있다. 이것은 대단히 중요한 관점인데 지금까지 컴퓨터 이론에서는 오피-코드와 오퍼랜드를 구분하지 않고 명령어라는 단위로 해석하고 있었다. 그런데, 오피-코드는 architecture에 따라서 차이점이 크지 않은 반면에 오퍼랜드를 표현하는 방법은 현격한 차이가 있으므로 이를 분리하여서 연구하는 것이 중요하다.

한편 명령어의 출현 빈도를 조사하면 짧은 길이의 오퍼랜드를 가지는 명령어의 출현 빈도가 특히 높은 것을 알 수 있다.

MIPS-3000에서 표준 C/C++ 라이브러리의 명령어 사용 빈도를 조사한 결과는 다음과 같다.

MOVE 22.8%
Load/Store 27.9%
Branch 19.4%
Immediate Load/Computation 18.3%

또한 Load/Store 명령어의 오프셋 길이에 따른 누적 빈도는 같다.

4 bit offset Load/Store 80%
8 bit offset Load/Store 95%

조건 분기 명령어의 90%는 8 비트 길이 오프셋이었으며, 연산 명령어에서 즉치 데이터 길이의 누적 빈도는 다음과 같다.

4 bit immediate data 58%
8 bit immediate data 94%

이러한 특성을 이용하여 EISC(Extendable Instruction Set Computer)는 출현 빈도가 높은 짧은 길이의 오퍼랜드를 가지는 고정 길이 명령어 세트를 가진다. 한편 오퍼랜드는 단순한 2진수 비트열로 취급할 수 있으므로 긴 길이의 오퍼랜드가 필요한 경우에는 이를 확장하는 수단을 가진다.

3.2 Operand Extension and LERI instruction

EISC 명령어는 짧은 길이 오퍼랜드를 가지고 있으며 긴 길이 오퍼랜드가 필요한 경우에는 확장 레지스타(ER)과 확장 플래그(E flag)를 사용하여 오퍼랜드를 확장한다.

ER은 특수 목적 레지스타이며 확장 플래그는 상태 레지스타 등에 포함되어 있는 1 비트 상태 플래그이다. ER에 즉치 데이터를 적재하는 명령어가 'LERI immd'이다. 32 비트 EISC 마이크로프로세서에서 ER은 32 비트 길이이며, 확장 플래그는 상태 레지스타 내에 있고, LERI 명령어는 다음과 같이 정의된다.

LERI instruction Format : LERI immd14

01 bb bbbb bbbb bbbb

Op-Code 14 bit immediate Operand

LERI operation :

If E flag is '0', then move sign extension of immd14 to ER ;

else ER = ER << 14 + immd14;

Set E flag;

LERI 명령어는 14 비트 즉치 데이터 오퍼랜드를 가지며, 확장 플래그가 '0'이면 14 비트 즉치 데이터 오퍼랜드를 32 비트로 부호 확장시켜 ER에 저장하며, 확장 플래그가 '1'이면, ER을 왼쪽으로 14 비트 회전시킨 후 14 비트 즉치 데이터 오퍼랜드를 연결시켜서 ER에 저장한다. 그리고 확장 플래그를 '1'로 세트시키는 동작을 수행하는 명령어이다,

한편 오프셋 또는 즉치 데이터 오퍼랜드를 가지는 모든 명령어는 확장 플래그가 '0'이면 짧은 길이 오퍼랜드로 동작하며, 확장 플래그가 '1'이면 ER에 저장된 값을 가지고 오퍼랜드를 확장한다. 또한 확장 플래그를 '0'으로 크리어한다.

32 비트 EISC 마이크로프로세서에서 Load/Store 명령어의 주소 계산 동작을 그림-1에 나타내면 다음과 같다.

그림-1에서 확장 플래그 상태를 판별하여 '0'이

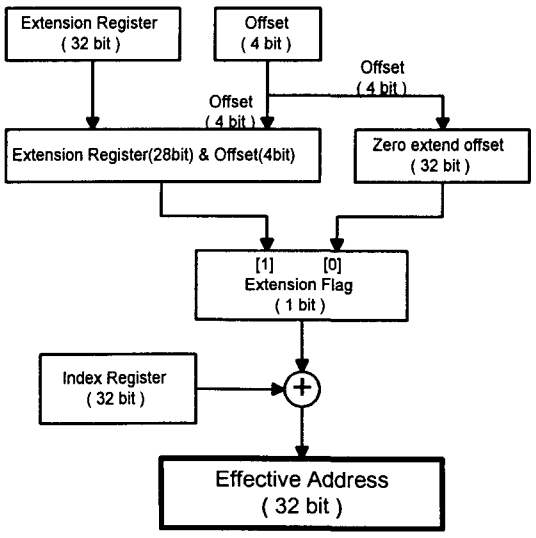

그림-1. 32 비트 EISC 마이크로프로세서에서 Load/Store 명령어의 주소 계산 동작

면 Load/Store 명령에 포함된 4 비트 오프셋을 무부호 확장하여서 32 비트 인덱스 레지스타와 더하여 메모리 주소를 계산한다. 반면에 확장 플래그가 '1'이면 ER 레지스타를 왼쪽으로 4 비트 회전시키고, Load/Store 명령에 포함된 4 비트 오프셋을 연결하여 32 비트 오프셋을 구하고 이를 인덱스 레지스타와 더하여 메모리 주소를 계산한다.

3.3 Pipeline data dependence

최근의 대부분 마이크로프로세서는 성능을 향상시키기 위하여 파이프라인을 기본적으로 채용하고 있다. 파이프라인의 스테이지 수는 종합적인 판단에 의하여 결정되지만 본 논문에서는 통상적인 5 스테이지 파이프라인을 예로 들어 설명한다. 5 스테이지 파이프라인은 다음과 같이 구성된다.

Stage 1 - Instruction fetch (IF)

Stage 2 - Instruction decode and operand fetch (ID/OF, or OF)

Stage 3 - Computational execution or effective address calculation (EX)

State 4 - Memory read or write (MEM)

Stage 5 - Write back to register (WB)

한편 EISC에서는 LERI 명령어는 ER을 변경하고 이 값을 후속하는 명령어에서 사용하므로 데이터 의존이 발생한다. 예를 들면,

```
LERI  #immd14; 18 bit immediate ADD
ADD   %R0, #immd4
SUB   %R1, #immd4
```

위 예에서 LERI 명령어에 의하여 ER 값이 변경되고, ADD 명령에서 변경된 ER 값을 사용한다. 뒤의 SUB 명령은 ADD 명령에서 확장 플래그를 '0'으로 크리어 하였으므로 오퍼랜드를 확장하지 않는다. 따라서 ADD와 SUB 명령 사이에서는 확장 플래그 변경에 따른 데이터 의존이 존재한다.

한편 EISC에서 ER 값을 변경시키는 명령어는 LERI가 유일하며, 또한 확장 플래그를 '1'로 세트시키는 명령어도 LERI가 유일하다. 또한 확장 플래그의 크리어 여부는 명령어 해석 스테이지에서 판단할 수 있다.

그러므로 EISC에서 ER과 확장 플래그 변경을 ID/OF 스테이지에서 수행하면 ER과 확장 플래그 변경에 따른 데이터 의존 문제를 제거할 수 있다.

3.4 LERI folding

EISC는 LERI 명령어를 연속적으로 사용하여 후속하는 명령어의 오퍼랜드를 확장하는 것을 주요 기술로 하고 있다. 오퍼랜드가 길어지면 여러 개의 LERI 명령어를 사용한다.

한편 최근의 32 비트 이상 마이크로프로세서에서는 캐시를 사용하는 것이 일반적이다. 캐시의 데이터 버스 폭은 32-128 비트에 이르는 것을 많이 사용하는데, 이것은 캐시가 블럭 단위로 동작하기 때문이다. 한편 파이프라인은 명령어 길이에 해당하는 폭을 가지고 있다. 이러한 데이터 버스 폭의 차이에 따른 문제점을 해결하고 성능을 높이기 위하여 캐시와 파이프라인 사이에 명령어 버퍼를 사용하고 있다.

즉, 다음에 실행할 명령어들이 명령어 버퍼에 저장되어 있고, 이로부터 순서적으로 명령어를 인출하여 파이프라인에 공급하는 구조이다.

EISC에서는 LERI 명령어는 오직 ER 값을 변경하는 동작만을 수행하므로 명령어 버퍼에 저장되어 있는 명령어를 조사하여 LERI 명령어를 추출하고 이들을 별도로 처리하는 회로를 도입하면 LERI 명령어를 파이프라인에서 처리하는 시간을 줄일 수 있으므로 성능을 크게 향상시킬 수 있다.

이것을 LERI folding이라고 부른다. LERI folding 회로의 블럭도를 그림-2에 보인다.

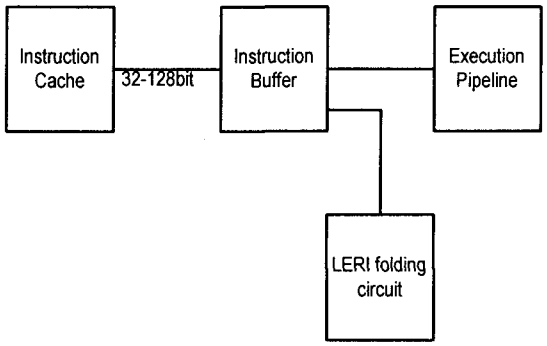

그림-2. LERI folding 블럭도

LERI folding 회로는 LERI 명령어만을 별도로 처리하는 간단한 하드웨어이다.

LERI folding의 효과는 다음 예에서 알 수 있다.

```
        MOV   %R1, %R2
        LERI  #immd14; 32 bit immediate ADD
        LERI  #immd14
        ADD   %R0, #immd4
        SUB   %R2. #immd4
          t1  t2   t3    t4   t5   t6   t7
I1:MOV    IF  OF   EX   MEM  WB
I4:ADD        IF   OF   EX   MEM  WB
I5:SUB             IF   OF   EX   MEM  WB
```

MOV 명령어 다음에 나오는 두 개의 LERI 명령어는 LERI folding 회로에서 처리하였다. 따라서 파이프라인에서는 바로 ADD 명령어를 수행할 수 있다.

32 비트 EISC 마이크로프로세서 AE32000으로 작성한 프로그램에서 LERI 명령어의 출현 빈도는 15~20%로 조사되었다. 따라서 AE32000에서 LERI folding 회로의 히트율을 80%라고 가정하면 12~16%의 성능 증가를 얻을 수 있다.

4. Main characteristics of EISC

4.1 Scalable architecture

EISC는 오피-코드와 오퍼랜드를 분리하여서 명령어를 구성하고 있다. 즉, 명령어는 오피-코드와 사용빈도가 높은 짧은 길이 오퍼랜드만으로 구성되어 있다. 오퍼랜드는 필요에 따라서 ER과 확장 플래그를 사용하여 확장하는 구조이다.

한편 16 비트, 32 비트 및 64 비트 마이크로프로세서에서 오피-코드는 모두 동일하다. 즉, 'ADD', 'SUB', 'MOVE', 'AND', 'OR', 'CALL' 등의 동일한 오피-코드를 가지고 있다. 따라서 모든 마이크로프로세서의 명령어는 소수의 특별한 명령어만을 제외하고는 동일하다.

마이크로프로세서의 설계는 명령어를 수행하는 회로를 설계하는 것이다. 만일 명령어가 동일하면 설계 또한 동일하게 된다. 그러므로 EISC는 16 비트, 32 비트 및 64 비트 마이크로프로세서의 설계가 거의 동일하다.

4.2 Configurable architecture

EISC의 특징을 이해하기 위하여 64 비트 CISC/RISC와 비교하여 본다. 64 비트 CISC는 실제적인 제품이 없으므로 MC68000를 64 비트로 확장한 것을 가정하였다. 64 비트 RISC는 MIPS-4000을 예로 들었다. 64 비트 EISC 마이크로프로세서는 AE64000을 예로 들었다.

즉치 데이터 ADD 명령은 오퍼랜드 길이에 따라서 다음과 같이 구현된다.

64 bit CISC

```
 3 bit add : add.q Rn, immd 3     ; 16 bit
16 bit add : add.s Rn, immd16     ; 32 bit
32 bit add : add.w Rn, immd32     ; 48 bit
64 bit add : add.l Rn, immd64     ; 80 bit
```

```
64 bit RISC
  16 bit add : add Rd, immd16, Rs        ; 32 bit
  32 bit add : ldui Rb, immd16           ; 96 bit
               addiu Rb, immd16, Rb
               add Rd, Rb, Rs
  64 bit add : ldl  Rb, (gp+offset16)    ; 96 bit
               add Rd, Rb, Rs
64 bit EISC
   4 bit add : add Rn, immd4             ; 16 bit
  16 bit add: leri immd12                ; 32 bit
              add Rn, immd4
  28 bit add : leri immd12               ; 48 bit
               leri  immd12
               add  Rn, immd4
  40 bit add : leri immd12               ; 64 bit
               leri  immd12
               leri  immd12
               add  Rn, immd4
  64 bit add : leri immd12               ; 96 bit
                .....
               leri immd12
               add Rn, immd4
```

이 예에서 CISC에서는 오퍼랜드 길이에 따라서 4 가지 종류의 명령어를 필요로 한다. 따라서 명령어 종류가 너무 많아지는 단점이 있다.

RISC에서는 4 비트 즉치 데이터의 출현 빈도가 58%이며, 8 비트 즉치 데이터의 출현 빈도가 93.6%를 차지하지만 이러한 통계와는 무관하게 항상 16 비트 길이 오퍼랜드를 사용하므로 비효율적인 것을 알 수 있다. 또한 32 비트 즉치 데이터가 요구되면 이를 3개의 명령어로 구현해야 하는 단점을 가진다.

이에 반하여 EISC에서는 필요로 하는 오퍼랜드 길이에 따라서 적당한 수의 LERI 명령어를 ADD 명령어 전단에 나열하면 충분하다. 따라서 한가지 ADD 명령어로 모든 길이의 오퍼랜드를 사용할 수 있으므로 프로그램 코드의 효율성이 높으면서도 명령어 수가 적으므로 하드웨어가 간단하여 진다.

따라서 EISC 마이크로프로세서는 하드웨어가 간단하므로 VHDL 등 하드웨어 설계 툴로 설계가 용이하며, 16/32/64 비트 마이크로프로세서를 동일한 구조로 설계할 수 있으므로 configurable 마이크로프로세서 설계를 가능하게 한다.

4.3 High code density

EISC는 출현 빈도가 높은 짧은 길이 오퍼랜드 명령은 16 비트 길이 명령어로 표현되므로 프로그램 크기가 작아진다. 따라서 코드 밀도가 증가한다.

C/C++ 라이브러리를 GNU GCC로 컴파일하여 생성된 기계어 코드 크기를 표-1에 보인다. 호스트 컴퓨터 시스템은 IBM-PC에서 Linux를 사용하였으며, ANSI C standard library는 Newlib-1.8.0을 사용하였고, C++ library는 SGI(Silicon Graphics Inc.)에서 개발하였고, SUN WS에서 사용하는 LIBSTDC++를 사용하였다. 기계어 코드 크기는 MIPS-3000에서 약 380KByte로 밴치 마크 검증으로 사용하기에 충분한 크기이다. 32 비트 EISC 마이크로프로세서인 AE32000의 프로그램 크기를 100으로 하였을 때 상대적인 프로그램 크기를 사용하여 표현하였다.

4.4 Application specific extendable architecture

EISC는 LERI 명령어를 사용하여 오퍼랜드를 확장하는 구조이다. 그런데 동일한 방식으로 명령어를 확장할 수 있다. 32 비트 EISC 마이크로프로세서에서 application specific extend instruction은 다음과 같이 정의된다.

표-1. 32 비트 EISC의 프로그램 크기 비교

Microprocessor	Code size	Reference
AE32000	100	Non optimized GCC compiler
MIPS-3000	171	32 bit RISC
MIPS-4000	164	64 bit RISC
MIPS-tx39	154	Toshiba modifies MIPS-3000 for embedded market
ARM-7/8/9	144	32 bit fixed length instruction
ARM-Thumb	103	16 bit fixed length instruction shrink ARM-7/8/9
SPARC (V8)	141	32 bit RISC
SPARCLITE	168	Fuzits modifies SPARC for embedded market
POWERPC	176	32 bit RISC
PA-RISC	194	32 bit RISC
ALPHA-RISC	178	64 bit RISC
SH-3/4	130	16 bit fixed length instruction
V850	118	16/32 bit length instruction
M32R	135	32 bit fixed length instruction
ARC	225	32/64 bit length instruction
I80960	142	32/64 bit length instruction
MC5200	139	CISC, simplified MC68000 instruction
MC68000	132	CISC
MC68332	130	CISC, modified MC68000 for embedded
MC68020	130	CISC
MN10300	112	8/16/24/32/40/48/56 variable length instruction
I80386	145	CISC

Application specific extend instruction :

1110 1110 00 bb bbbb

Op-Code 6 bit instruction

Effective extend instruction → ER << 6 + 6 bit instruction

32 비트 EISC 마이크로프로세서에서 하나의 LERI 명령어로 14 비트를 확장할 수 있다. 따라서 application specific extend instruction의 길이는 20 비트 또는 34 비트 길이이다.

5. Conclusion

실장 제어 마이크로프로세서는 코드 밀도가 높으면서 scalable, configurable, application specific extendable의 3 가지 기능을 갖추어야 한다.

코드 밀도가 높고 scalable, configurable, application specific extendable 기능을 가지는 마이크로프로세서를 구현하기 위해서는 종래 RISC/CISC architecture는 적합하지 않다.

RISC는 32 비트 고정 길이 명령어를 사용하므로 16 비트 마이크로프로세서를 만드는 것은 적합하지 않다. 또한 오퍼랜드 길이가 제한되므로 64 비트 마이크로프로세서 또한 적합하지 않다. 그리고 CISC는 하드웨어가 복잡하므로 적합하지 않으며, 오퍼랜드 길이에 따라서 여러 개의 명령어를 만들어야 하므로 명령어 세트가 복잡해진다.

따라서 16 비트 마이크로프로세서는 가능하지만 32/64 비트 마이크로프로세서를 설계하는 것은 적합하지 않다.

EISC는 16 비트 고정 길이 명령어 세트를 가지면서 필요에 따라서 오퍼랜드 길이를 확장하는 구조이다. 16/32/64 비트 마이크로프로세서는 모두 동일한 오피-코드를 가지며 단지 오퍼랜드 길이만이 다르다. 따라서 EISC는 동일한 구조를 가지는 16/32/64 비트 마이크로프로세서를 설계하는 것이 가능하다. 또한 EISC 기반 마이크로프로세서는 명령어 세트가 간단하므로 하드웨어가 간단하면서 코드 밀도가 높아서 프로그램 메모리 크기가 작아지므로 SoC 등 실장 제어 마이크로프로세서로 특히 적합하다.

참 고 문 헌

[1] Dezso Sima *et al.*, "Superscalar Instruction Issue," IEEE Micro, pp. 28-39, Oct. 1997

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

[3] 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

[4] 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

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

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

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

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

[9] Kazuaki Murakami, et al, "Trends in High-Performance, Low-Power Processor Architecture," IEICE Trans. Electron., Vol E84-C, No. 2. Feb. 2001

[10] Steve Leibson, "Trend in Low-Power and High-Integration Embedded Processors," Embedded Processor Forum, 2000

[11] 조 경연, "확장 명령어 32 비트 마이크로프로세서에 관한 연구," 전자공학회 논문지, 6권 D편 5호, pp. 391-400, May 1999

[12] 조 경연, "16 비트 EISC 마이크로프로세서에 관한 연구," 한국멀티미디어학회 논문지, 3권 2, pp. 192-200, Apr. 2000

[13] 차 영호, 조 경연, 최 혁환, "실장 제어 16 비트 FPGA 마이크로프로세서," 한국해양정보통신학회 논문지, 5권 호7, pp. 1332-1338, Dec. 2001

조 경 연

- 1990.2~인하대학교 공학박사
- 1983.3~1991.2 : 삼보컴퓨터 기술연구소 부장
- 1991.3~현재 : 부경대학교 전자컴퓨터정보통신공학부 교수
- 1991.3~2001.9 : 삼보컴퓨터 기술연구소 기술고문
- 1993.3~1995.2 : 부경대학교 전자계산소 소장
- 1993.6~1998.4 : 아남에스엔티 기술고문
- 1998.4~현재 : 에이디칩스 사외이사 겸 기술고문
- 주관심분야 : 마이크로프로세서 설계, 반도체 설계, 시스템 프로그래밍, 컴퓨터 아키텍쳐