주 제

고성능 국산 임베디드 마이크로프로세서(EISC)

에이디칩스 이 희

차례

I. EISC 소개

명령어는 동작을 나타내는 Op-Code와 그 동작을 받는 Operand로 구성되며, Operand는 단순한 bit string으로 이에 의미를 부여하는 것은 Op-Code이다. 따라서 Operand를 Op-Code와 독립적인 bit string으로 해석하면 Op-Code가 필요로 하는 길이만큼 Operand를 확장하는 명령어 구조 체계를 만들 수 있다. 확장 레지스타(ER : Extension register)와 확장 프래그(E : Extension flag)라는 새로운 개념을 도입하여 고정길이명령어로 이러한 명령어 구조를 구현한 것이 EISC (Extendable Instruction Set Computer)이다.

한편 microprocessor는 시장의 요구에 따라서 16/32/64 bit의 다양한 사양이 필요하다. 그런데 16/32/64 bit microprocessor의 명령어에서 Op-Code는 동일하며 Operand 길이만이 달라진다. EISC 명령어 구조는 Operand 길이를 필요한 길이만큼 임의 길이로 확장이 가능하므로 16/32/64 bit microprocessor를 모두 동일 명령어 구조로 설계되어 있다.

II. EISC 특징

EISC microprocessor의 중요한 특징을 5S로 정

의 하여 설명 한다

5S 란?
Small code size
Scalable Architecture
Simple instruction set and hardware
Speed – high Speed
Saving – power saving

1. Small code size (High code density)

Embedded application에서는 microprocessor, memory 및 I/O circuit가 하나의 ASIC에 integration되는 SoC(System on Chip) 기술이 특히 중요하다. 이러한 ASIC에서 가장 넓은 면적을 차지하는 것이 program ROM이다. 그러므로 ASIC의 가격을 낮추기 위해서는 program size를 줄여야 하며, 이를 위해서는 microprocessor의 code density가 높아야 한다.

한편 반도체 기술의 급격한 발전으로 microprocessor operating frequency가 GHz 단위가 되고 있다. 이렇게 core frequency는 급격하게 증가하고 있지만 memory access time과 bus bandwidth는 이러한 증가 속도를 따라가지 못 하고 있다. PCI bus는 아직 33MHz에 머물고 있으며 최근에 PCI-X는 133MHz를 목표로 하고 있다.

이러한 현상은 향후 더욱 심화될 것으로 전망되어, system performance 향상에 가장 큰 bottle neck으로 대두되고 있다.

이러한 bottle neck을 해결하기 위해서 bus architecture에 대한 연구와 더불어서 data bus traffic을 줄이는 기술이 필요하다. Data bus traffic은 80%가 program code이며 20%가 data이다. 따라서, code density를 높이는 것이 data bus traffic을 줄이는 유용한 기술이다.

EISC는 출현 빈도가 높은 short length Operand을 가지는 16 bit fixed length instruction으로 구성되어 있으며, 필요한 Operand length 만큼 LERI instruction을 사용하여 Operand를 확장하는 방식을 사용하므로 program size가 작아 진다. 즉, code density가 높은 장점을 가진다.

2. Scalable Architecture Scalable 16/32/64 bit microprocessor architecture

RISC architecture는 32 bit fixed length instruction을 사용하므로 16 bit microprocessor에 사용하기에는 instruction length가 너무 길어서 적합하지 않다. 또한 64 bit microprocessor에서도 maximum length Operand가 16 bit로 제한되므로 적합하지 않다. 따라서 RISC는 32 bit microprocessor에 한정적으로 사용할 수 있는 architecture이다,

CISC architecture는 Operand length에 따른 다수개의 Op-code를 가지는 복잡한 instruction set를 가지므로 64 bit microprocessor에는 적합하지 않다.

또한 16 bit fixed length instruction이나 limited variable length instruction을 가지는 대부분의 microprocessor에서도 long length Operand를 표현하는 방식이 적합하지 않아서 64 bit microprocessor에는 적합하지 않다.

예를 들면 Tensilica는 16/24 bit length instruction set을 사용한다. 따라서 16 bit microprocessor에 24 bit length instruction은 너무 길기 때문에 효율적이지 않다. 또한 24 bit length

instruction에서도 16 bit 이상 길이 Operand는 모두 PC relative immediate data load instruction을 사용한다. 따라서 64 bit microprocessor를 Tensilica architecture로 실현하려면 16 bit 길이 이상 되는 모든 Operand를 64 bit로 표현하여야 하므로 효율적이지 않다.

16/32/64 bit microprocessor에서 Op-Code는 모두 동일하다. 즉, MOV/Load/Store/ADD/SUB/Compare/ADD/OR/XOR/Branch/Call or Jal/Shift 등등 Op-code의 종류는 16/32/64 bit microprocessor에 따라서 차이가 없다.

그러나 Operand length에는 현격한 차이가 있다. 즉, Operand length는 word length와 일치하거나 word length보다 길다. 이러한 차이점에도 불구하고 short length Operand를 가지는 instruction의 출현 빈도가 높은 것은 공통적인 사항이다.

EISC는 이러한 16/32/64 microprocessor의 특징을 반영하여서, Operand를 필요한 길이만큼 확장하는 구조이므로 16/32/64 bit microprocessor 모두에서 효율적이다.

16 bit EISC microprocessor에서는 'LERI #immd12'로 12 bit를 ER에서 확장하며, 32 bit EISC microprocessor에서는 'LERI #immd14'로 14 bit 단위로 ER을 확장한다. 즉, immediate data computation instruction에서 Operand length는 4/18/32 bit length 단위로 가장 적합한 Operand length를 선정할 수 있다.

또한 64 bit EISC microprocessor에서는 'LERI #immd12'로 12 bit 단위로 ER을 확장한다. 따라서 4/16/28/40/52/64 bit length 단위로 최적의 Operand length를 선정할 수 있다. 이에 반하여 기존의 64 bit microprocessor는 16 bit, 64 bit 2가지 length Operand만을 선택해야 하므로 불필요하게 program size를 증가시키고, 이에 따라서 performance도 저하되고, 전력소모도 증가하는 문제점이 있었다.

나아가서 EISC architecture에서는 16/32/64 bit microprocessor에 따른 instruction set의 변화가 크지 않으므로 하나의 microprocessor만 개발하면 다른 word length microprocessor도 손쉽게 개발할 수 있는 장점을 가진다.

3. Simple instruction set, simple hardware

EISC는 16 bit fixed length instruction set를 가지며, 한가지 Op-Code에 한 개의 instruction만을 가지고 있으므로 instruction의 수가 적다. 즉, Operand length에 따른 여러 개의 instruction을 만들 필요가 없으므로 instruction 수가 기존 microprocessor 보다 적다. 따라서 hardware가 간단하고, operating frequency가 높아지므로 performance 또한 높아진다.

8개의 register를 가지는 32 bit simple EISC인 SE3208과 이와 유사한 ARM-7TDMI를 비교하면 다음과 같다.

	SE3208	ARM-7TDMI
No. of register	8	8
Instruction	16 bit fixed	16 bit fixed
Multiplier	32 X 32	32 X 8
No of Gate	25K	40K
Operating Freq.	50MHz	40MHz
Design	VHDL	Hard macro

위 예의 비교는 삼성 0.35 micron CMOS 공정을 사용하였으며, SE3208은 VHDL로 설계하여 합성한 것이며, ARM-7TDMI는 ARM사에서 제공한

hard macro를 사용하였다. 특히 multiplier는 많은 gate count가 필요한 회로이다. SE3208은 performance를 위하여 32 X 32 구성을 사용하고 있으면서도 32 X 8 multiplier를 채용한 ARM-7TDMI보다 total gate count가 작다.

일반적으로 VHDL 설계는 optimization이 되어 있지 않으므로 hard macro에 비하여 gate count도 증가하고 operating frequency가 크게 낮다.

이러한 조건에도 불구하고 SE3208이 ARM-7TDMI에 비하여 gate count와 operating frequency에서 대단히 우수하게 비교 결과가 나왔다. 한편 동일한 frequency를 가지고 있다고 하여도 SE3208에서는 PC relative immediate data load instruction을 사용하지 않으므로 이에 따른 data dependence가 없으므로 ARM-7TDMI 보다 performance가 높다. SE3208과 ARM-7TDMI의 code density는 비슷하다.

4. Speed (high Speed)

EISC는 16 bit fixed length instruction set를 가지며, 한가지 Op-Code에 한 개의 instruction만을 가지고 있으므로 instruction의 수가 적다. 즉, Operand length에 따른 여러 개의 instruction을 만들 필요가 없으므로 instruction 수가 기존 micro-processor 보다 적다. 따라서 hardware가 간단하고, operating frequency가 높아지므로 per-formance 또한 높아진다.

5. Saving (power Saving - low power)

Microprocessor의 power consumption은 implementation과 realization에 주로 의존한다. 예를 들면 pentium chip이 desk top 용이 있고 note book 용이 있는데 이들은 동일한 architecture이다. 이와 같이 microprocessor의 전력 소모는 implementation과 realization에 주로 영향을 받는다. 한편 architecture level에서도 low power consumption에 대한 고려가 필요하다.

CMOS에서 전력 소모는 logic status가 변화하여서 발생한다. 즉, CMOS에서 전력 소모 Pd는 Pd = C X V2 X F로 주어진다. C는 CMOS 제작 공정 중에서 발생하는 capacitance이고, V는 공급 전압, 그리고 F는 동작 주파수이다.

여기서 C와 V를 낮추는 것은 implementation과 realization에 직접적으로 관련되어 있으며 architecture와는 무관하다. F는 동작 주파수인데 CMOS 출력의 변화 주파수이다.

따라서 architecture에서 전력 소모를 낮추기 위해서는 첫째로 전력을 소모하는 logic gate count를 줄이도록 하여야 하는데 이를 위해서는 hardware가 간단하게 되는 구조가 되어야 한다. 둘째로는 logic gate 입출력의 상태가 자주 변화하지 않도록 하여야 하는데, 이를 위해서는 data bus traffic을 낮추어야 한다. 따라서 program size가 작은 architecture 즉 high code density architecture가 요구된다.

EISC는 기존 microprocessor에 비하여 hardware가 간단하고 code density가 높은 architecture이므로 전력 소모가 적은 architecture이다.

III. EISC 아키텍처의 기본 개념

EISC architecture는 가변길이 Operand를 표현하기 위해서 확장 레지스타(ER)과 확장 프레그(E)를 사용한다. 32 bit EISC microprocessor에서는

32 bit 확장레지스타를 사용한다. 확장 레지스타는 확장할 Operand를 기억하며 확장 레지스타에 확장할 Operand가 기억되어 있는 것을 나타내는 것이 확장 프래그이다. 또한 확장 레지스타에 확장할 Operand를 load하는 명령어가 'LERI' 명령어로 다음과 같이 정의한다.

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

확장 프래그는 'LERI' 명령어를 수행하면 '1'이 되고, Operand를 확장하는 모든 명령어에서는 '0'이 된다. Low end 32 bit EISC microprocessor인 SE3208에서 load/store 명령어는 다음과 같이 정의한다.

Instruction Function : Load/Store
Instruction Representation :
bit 15-14 = 00
bit 13-11 = Operation
000 : Sign extend 8 bit load. LDB (Index, offset), dst
001 : Sign extend 16 bit load. LDS (Index, offset), dst
010 : 32 bit load. LD (Index, offset), dst
011 : Zero extend 8 bit load. LDBU (Index, offset), dst
100 : 8 bit store. STB src, (Index, offset)
101 : 16 bit store. STS src, (Index, offset)
110 : 32 bit store. ST src, (Index, offset)
111 : Zero extend 16 bit load. LDSU (Index, offset), dst
bit 10-8 = Source/Destination register. %R0 thru %R7.
bit 7-3 = Offset bit 5-0 if 8 bit load/store
Offset bit 6-1 if 16 bit load/store
Offset bit 7-2 if 32 bit load/store
bit 2-0 = Index register. %R0 thru %R7.
Effective operand address : EA
If (E flag is 0) -- EA = Zero extend offset + Index register
If (E flag is 1) -- EA = %ER << 4 + Offset + Index register

이와 같이 확장 레지스타와 확장 프래그에 의하여 가변길이 Operand를 구현하면 모든 명령어를 16 비트 고정 길이로 표현하는 것이 가능하다.

IV. EISC 명령어 셋트 특징

EISC 명령어의 중요 특징을 정리하면 다음과 같다.

1. General purpose register

RISC는 대부분 32개의 범용 레지스타를 가지고

있다. 이것은 RISC 연구가 진행되던 1980년대 초에 주로 사용하였던 고급언어가 C/PASCAL/APL 등이었는데, PASCAL과 APL은 nested procedure를 허용하는 언어이다. 따라서 RISC는 nested procedure를 지원하기에 적합한 구조를 가지게 되었다. 그러나 현재는 C/C++/Java가 주로 사용되며 이들 언어는 nested procedure를 허용하지 않는다. 따라서 RISC 연구 당시와는 다른 환경을 가지게 되었다.

한편 MIPS-R3000은 32개의 32 비트 레지스타를 가지고 있다. 그런데 5개는 스택 포인터, 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

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 정규화 수치로 나타내고 있다. 범용 레지스타 수가 작아지면 load/store 빈도와 프로그램 크기가 증가한다. Load/store는 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 load/store 빈도에서 큰 차이가 없다. 따라서 AE32000과 AE64000 EISC micro-processor 군에서는 16개의 범용 레지스타를 설정하였다.

한편 8개의 범용 레지스타는 load/store 빈도가 높아서 비효율적이지만 이에 비례하여 hardware가 간단해 지므로 low end 시장에 적합하다. 따라서 SE1608/SE16108/SE3208/SE6408 등 simple EISC microprocessor 군에서는 8개의 범용 레지스타를 설정하였다.

설명의 편의상 이후 본 보고서의 프로그램 분석은 16개 레지스타를 사용하도록 변형한 MIPS-R3000 컴파일러를 사용한다.

2. Load/store machine

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

EISC는 모든 연산 명령은 레지스타 오퍼랜드를 가지며, 메모리 입출력은 load/store 명령어로 제한하는 load/store 구조를 가진다. Load/store 구조를 가지므로 하드웨어가 단순해지고 따라서 동작 속도를 빠르게 할 수 있다.

3. 16 bit fixed length instruction

표 2에서 'move' 명령어가 20.27%의 사용 빈도를 보이고 있다. AE32000/AE64000에서 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, Load/store 명령어와 상수 오퍼랜드 명령어는 오프셋 및 상수 오퍼랜드 길이를 감안하면 16 비트 고정 길이 명령어로 표현하기가 용이하지 않다.

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

<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 3 비트 오프셋으로 77%의 명령

〈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%

어를 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 3>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 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 짧은 길이 오프셋과 상수 오퍼랜드를 가지는 명령어의 빈도가 특히 높으므로, 짧은 길이 오프셋과 상수 오퍼랜드를 가지는 명령어를 정의하고 확장 레지스타와 확장 프래그를 사용하면 긴 길이의 오퍼랜드를 표현할 수 있다.

따라서 EISC는 출현 빈도가 높은 짧은 오퍼랜드를 가지는 명령어만을 정의하여 code density가 특

히 높은 16 비트 고정 길이 명령어를 구현할 수 있다.

4. Stack related features

<표 3>에서 'lw, sw' 명령어의 오프셋 길이가 스택 포인터와 인덱스 레지스타를 사용하는 경우에 현격한 차이를 보이고 있다. 이것은 C/C++/Java 등 고급 언어의 특성상 지역 변수와 전달 변수가 스택에 만들어지기 때문이다.

한편 대부분의 RISC에서는 스택 포인터를 별도로 정의하지 않고 범용 레지스타의 하나를 스택 포인터로 사용하고 있다. 이러한 RISC 구조는 16 bit 오프셋을 사용하므로 <표 3>에 보인 바와 같이 일반 영역과 스택 영역을 구분하지 않아도 문제가 되지 않았다.

또한 SH-3, V800, ARM 등 RISC와 CISC의 중간 구조를 가지는 microprocessor에서도 스택 포인터를 별도로 정의하지는 않았다.

EISC는 짧은 길이 오퍼랜드를 가지는 명령어만을 정의하고 'LERI' 명령어로 임의 길이 오퍼랜드를 표현하는 구조이므로 명령어 수가 기존 RISC 보다도 작다. 따라서 instruction code space에 여유가 충분하다. 이러한 점과 스택 영역의 동작 특성이 일반 영역과 다른 점을 고려하여 EISC에서는 스택 포인터를 별도로 분리하였으며, 스택 영역 접근에 적합하도록 별도의 스택 명령어들을 정의하였다.

그러므로 EISC는 범용 레지스타의 일부를 특수 레지스타로 사용하지 않으므로 컴파일러에서 사용 가능한 연산용 레지스타가 경쟁제품에 비하여 많다.

예를 들면 ARM-7/8은 16개의 레지스타를 가지지만 R15(PC), R14(SP), R13(LR)로 3개의 레지스타를 특수 목적으로 사용하고 있다. 또한 제한된 길이의 오퍼랜드만을 표현할 수 있고 긴 길이 오퍼랜드는 PC relative immediate data load 명령어를 사용하여야 하므로 별도의 레지스타를 하나 더 점유하게 된다. 따라서 ARM-7/8에서는 실제적으로 12개의 범용 레지스타만을 컴파일러가 사용할 수 있다.

따라서 EISC는 고급언어 컴파일러가 사용하는 범용 레지스타 수가 많으며, 스택 영역을 효율적으로 사용할 수 있도록 지원하여 주므로 고급언어 컴파일러에 효율적이 된다.

5. Other instructions

<표 2>에서 조건 분기 명령어의 빈도가 6.69%이다. 이를 효율적으로 지원하기 위해서 EISC에서는 캐리, 사인, 제로, 오버플로우의 4개 플래그를 사용하며 이들을 조합하여 14가지 조건 분기 명령어를 만든다.

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 있다. AE32000/AE64000은 16개의 범용 레지스타를 가지므로 논리 산술 연산 명령어는 2 오퍼랜드 형식을 가진다. 반면에 SE 계열 EISC는 8개의 범용 레지스타만을 가지므로 3 오퍼랜드 형식을 가진다.

곱셈 명령어는 출현 빈도는 낮으나 멀티미디어 등 특정 응용 분야에서 사용 빈도가 특히 높으며, 구현 방식에 따라 동작 속도에 차이가 크다. 이러한 성질을 감안하여 EISC에서는 SE16108을 제외하고 곱셈 명령어를 가진다.

32/64 bit EISC에서 코프로세서는 최대 4개를 설정할 수 있으며, 각각의 코프로세서는 범용 레지스터 수와 동일한 레지스타를 가진다. 코프로세서 '0'은 시스템 코프로세서로서 캐시 제어, 파이프라인 제어, 메모리 관리 등의 시스템 관리를 수행한다. 부동수소점수 연산기는 코프로세서 '1'에 의하여 구현한다. 코프로세서 명령은 확장 레지스타를 이용하여 20 비

트 또는 34 비트 길이를 가진다.

V. EISC 소프트웨어

대부분의 16 bit 이상 microprocessor에서는 C/C++/Java 등 high level language를 사용하여 program을 개발한다. 따라서 high level language에 적합한 architecture를 가져야 한다.

C/C++ compiler는 C/C++ source file을 compile하여 assembler source file을 생성한다. 생성된 assembler source file을 assembler가 assemble하여서 object file을 생성한다. 이렇게 생성된 하나 또는 그 이상의 object file은 link에 의하여 library와 link되어서 execution file이 생성된다.

EISC의 instruction set는 C/C++ program을 compile하여 생성된 instruction의 출현 빈도를 정밀하게 분석하여 선정하였다. 예를 들면, load/store의 출현 빈도가 25-30%에 이르는 MOVE와 함께 가장 많이 사용되는 instruction이다. Load/store의 특징을 보다 세밀하게 하면 heap area와 stack area에 load/store하는 특성이 다르다.

이러한 점을 고려하여 EISC에서는 heap area와 stack area에 load/store하는 instruction이 분리되어 있다.

한편 compiler에서 assembler source file을 생성할 때 instruction의 Operand length가 결정되지 않는 경우가 있다. 대부분 high level language는 library에 utility 및 interface program 등을 작성하여 놓고, 사용자 program에서 이를 호출하는 방식을 사용하고 있다. 따라서 link 이전에는 instruction의 Operand 값을 확정지을 수 없다.

그런데 많은 microprocessor에서는 Operand length에 따라서 다른 instruction을 사용하고 있으므로 문제가 발생한다.

Intel 8086 계열의 C compiler에서는 Operand length에 따라서 small model, medium model, large model, huge model 등으로 model을 분리하고, model에 따라서 서로 다른 instruction set을 사용하기도 하였었다.

대부분의 microprocessor에서는 assembler시에 확정되지 않은 Operand는 가장 긴 길이의 operand를 가지는 것으로 가정하고 object code를 생성한다. 이러한 정책은 비효율적인 code를 생성하는 단점이 있지만 link 시에 오류를 생성하지 않기 때문에 불가피한 측면이 있다.

이러한 문제는 C++/Java 등 object oriented language가 발달하면서 더욱 심화되고 있다.

EISC에서는 모든 길이의 Operand를 사용할 수 있으므로 이러한 문제는 발생하지 않는다. 즉, compiler에서 assembler source file을 생성할 때는 Operand length에 상관없이 assembler source를 생성한다. 이 단계에서는 LERI instruction을 사용하지 않는다. Assembler가 object file을 생성할 때 Operand length에 따라서 필요로 하는 LERI instruction을 생성한다.

그런데 assembler 단계에서 Object length를 알지 못 하면 최대 길이 Operand를 가상하여 LERI instruction을 생성한다. 그리고 link에서 Operand length가 확정되어 LERI instruction의 수가 결정되면 이때 여분으로 들어 간 LERI instruction은 relaxation시킨다. 이와 같이 link의 relaxation 기능을 사용하면 항상 최적의 LERI instruction만을 사용할 수 있다.

EISC는 high level language compiler에 적합한 instruction set을 갖추고 있으며, 모든 길이의 Operand를 표현할 수 있으며, link의 relaxation 기능을 이용하여 최적의 code를 생성할 수 있으므로

high level language compiler에 적합하다.

ADC에서는 free software를 이용하여 EISC compiler를 개발하였다. 개발된 EISC software 목록은 다음과 같다.

Host Platform	IBM-PC-Linux, IBM-PC-Window-95/98/Me/NT, SUN
Object File format	ELF
Cross assembler	Binutils-2.9.1
Cross loader	Binutils-2.9.1
Binary Utilities	Binutils-2.9.1
Cross C complier	GCC-3.4.5
Cross C++ complier	GCC-3.4.5
ANSI C library	Redhat(Cygnus) Newlib-1.8.2
C Math library	Redhat(Cygnus) Newlib-1.8.2
C++ library	GCC-2.95.2 (LIBSTDC++-2.10.0)
Cross debugger	GDB-5.0
Real time OS	uC/OS-2.0 Redhat(Cygnus) eCOS -- Under development
IDE(Work bench)	V-IDE for Window and Linux K-development for Linux

VI. EISC CPU에 적용된 기술

에이디칩스의 EISC microprocessor에 적용된 기술들은 다음과 같다.

1. On Silicon ICE (OSI)

Microprocessor core가 ASIC에 integration 되는 SoC 분야에서는 ICE(In Circuit Emulator)를 지원하기 위한 수단이 필요하다. 이러한 수단으로는 Motorola사의 BDM, ARM의 IceBreaker 등이 있다.

한편 이러한 ICE module은 개발시에만 필요한 것이며 microprocessor 양산시에는 불필요한 부가적인 circuit이다. 따라서, ICE module은 간단한 hardware에 의하여 기본적인 기능만을 가지도록 설계되는 것이 요구된다.

ICE가 동작하는 개념을 살펴보면 host system이 target system의 동작을 감시, 관리하는 구조이다. 여기서 host system은 PC 또는 WS으로 사용자가 debug program을 수행시키는 system이며, target system은 debug 대상이 되는 microprocessor를 포함하는 system이다. Host system은 target microprocessor의 동작을 감시, 관리하기 위해서 필요한 hardware를 사용하게 되는 데, 이 hardware module이 BDM, IceBreaker 등이다.

EISC에서는 이러한 ICE을 지원하기 위하여 virtual processor를 설정하고, virtual processor가 host system과 통신을 하면서 target processor를 감시, 관리하는 architecture를 사용하였다.

Process는 독립적인 resource를 가지고 program을 수행하는 object로 정의할 수 있다. 독립적인 resource를 가지기 위해서는 독립적인 process mode와 stack pointer 및 stack area가 필요하다. 이를 위하여 EISC에서는 OSI mode, OSI stack pointer를 가지고 있다.

이러한 EISC의 ICE module인 OSI(On Silicon ICE)는 최소한의 hardware로 remote debug 기능을 제공하므로 program 개발을 용이하게 한다.

2. Multiplier, DSP and SIMD-DSP

Multiplier는 많은 gate를 필요로 하는 복잡한 circuit이다. 따라서 EISC microprocessor는 종류에 따라서 다른 multiplier 및 DSP 기능을 가지고 있다.

SE1608 -- 16 bit low end simple EISC microprocessor

16 X 16 = 32 signed/unsigned integer multiplier

SE3208 -- 32 bit low end simple EISC microprocessor

32 X 32 = 32 signed integer multiplier

AE32000 -- 32 bit EISC microprocessor

basic model ⇒ 32 X 32 = 64 signed/unsigned integer multi-plier

32 X 32 + 64 = 64 signed multiplier and accumulator(MAC)

DSP model ⇒ 32 X 32 = 64 signed/unsigned integer multiplier

32 X 32 + 64 = 64 signed multiplier and accumulator(MAC)

Next address generator, Reverse address generator

DSP XY model ⇒ 32 X 32 = 64 signed/unsigned integer multiplier

32 X 32 + 64 = 64 signed multiplier and accumulator(MAC)

Next address generator, Reverse address generator

Dedicated DSP X/Y memory -- Maximum 256K byte

MAC, address generate, reverse address in a single clock

Vector instruction

AE64000 -- 64 bit EISC microprocessor

basic model ⇒ 64 X 64 = 64 signed integer multiplier

64 X 64 + 64 = 64 signed multiplier and accumulator(MAC)

SIMD-DSP model ⇒ Quad 16 X 16 + 64 = 64 MAC in a clock

Dual 32 X 32 + 64 = 64 MAC in a clock

Next address generator, Reverse address generator

Dedicated DSP X/Y memory -- Maximum 8M byte

MAC, address generate, reverse address in a single clock

Vector instruction

SIMD instruction = Pack/Unpack, Add/Subtract,

Saturated add/subtract, Clamp, Compare,

Extract/Substitute/Replace, Shift/Rotate,

Repeat/Convert

3. Power management

Embedded system에서 power management는 대단히 중요한 기술이다. Power management unit(PMU)는 microprocessor의 상태를 알고 있어야 한다.

이를 위하여 EISC에서는 exception이 발생하는 등 microprocessor 상태가 변화하면 special bus cycle을 발생하여 상태 변화를 PMU에게 알려 준다.

4. Java accelerator

Java는 internet application system에서 중요한 language이다. 따라서 Java의 J-code를 효율적으로 지원하는 수단이 필요하다.

이를 위하여 J-code를 hardware로 수행하는 방식과 JIT(Just-In-Time) compiler를 이용하는 방

식이 있다. 지금까지의 연구 결과 JIT compiler가 J-code hardware emulator보다 성능이 우수한 것으로 보고되고 있다. 그런데 JIT compiler는 많은 RAM을 필요로 한다. 따라서 RAM 용량이 적은 휴대용 제품에서는 J-code hardware emulator가 필요하다.

EISC에서는 J-code hardware emulator가 없는 model과 있는 model 사이에서 software compatibility가 유지되는 Java accelerator instruction을 가지고 있다.

즉, EISC Java accelerator instruction은 J-code hardware emulator가 없으면 J-code software interpreter를 효율적으로 지원하며, J-code hardware emulator가 있으면 software를 수정하지 않고 이를 이용하는 구조이다.

J-code hardware emulator는 현재 개발중이다.

5. Cache

EISC cache는 application system에 따라서 다양한 구성으로 되어 있다.

Cache는 core architecture와 무관하게 선정할 수 있는 부가적인 회로이다. 또한 상당히 큰 silicon area를 차지하므로 여러 가지 조건을 종합적으로 판단하여 구조를 결정하여야 한다.

이를 위하여 EISC microprocessor 종류에 따라서 다음과 같은 cache 구조를 개발하였다. Cache size는 silicon size에 따라서 결정되는 사항이므로 명시하지 않는다.

SE3208 -- 32 bit low end simple EISC microprocessor

Unified, 2 way set associative, write through/back, without snooping cache

AE32000/AE64000 -- 32/64 bit EISC microprocessor

Instruction/Data separated, virtual address indexed physical address tagged,

2 or 4 way set associative cache.

Write policy는 'write through/back without snooping', 'write through with snooping and write back without snooping' 또는 'MSI'를 선택한다.

6. Memory management unit (MMU)

EISC MMU는 memory bank control logic과 TLB의 2 level로 구성되어 있으며, memory bank control logic은 32 bit EISC AE32000에서는 512Mbyte 8 bank로 구성되어 있어서, TLB가 없는 model 또는 non-TLB region에서 memory bank attribute를 정의한다.

TLB는 SAS(Single Address Space) model과 MAS(Multiple Address Space) model의 2 가지가 있다. SAS model에서는 4K/256K byte/page, 4 way set associative map, software replacement 구조이다. MAS model에서는 block TLB와 page TLB의 dual TLB로 구성되어 있으며, memory bank 단위로 MAS 구조를 가지며, hardware/software replacement가 가능하며 최근의 OS 구조에서 많이 사용하는 shared library 및 DLL을 효율적으로 지원하는 구조로 되어 있다.

이러한 EISC MMU는 가장 간단한 system에서 embedded-linux 또는 Window/CE를 지원할 수 있는 수준에 이르기까지 확장할 수 있는 유연성을 가지고 있다.

7. Floating point co-processor

Floating point 연산은 high quality audio가 널리 보급되면서 embedded system에서도 많이 요구되고 있다.

EISC는 최대 4개까지의 co-processor를 지원할 수 있는데, co-processor#0은 system control co-processor로 cache/MMU/OSI 등의 기능을 지원하며, co-processor#1은 floating point co-processor이다.

EISC는 IEEE-754 single/double precision floating point format을 사용한다.

8. EBUS (EISC BUS)

EISC core와 peripheral device를 interface는 internal bus이다. Peripheral device는 EBUS-to-PCI bus bridge, EBUS-to-AMBA bus bridge, DRAM/SDRAM controller, EBUS-to-AGP bus bridge, Fast-IO controller 등이 있다.

VII. MCU Trand

ARM의 기술 방향은 ARM-tdmi기술을 바탕으로 code density를 높이고 micro-architecture 연구에 주력 하고 있다. 그래서 ARM은 ARM7, ARM8, ARM9,ARM10 등의 고성능화에 주력 하는 한편 외부에 라이선스을 주어 인텔에서 Strong ARM, XScale 개발했다.

다른 한방향은 ARM 코어에 DSP, SIMD, jazelle을 접목한 MCU를 가지고 있다.

Mips의 기술 방향은 ARM과 달리 ASE (Application Specific Extension) 프로세서를 개발 하고 있다. R-3000을 기반으로 암호화 명령어 탑재한 Smart MIPS 개발과 bule-tooth 명령어을 탑재한 프로세서 개발 그리고 R-4000기반에 SIMD DSP 명령어를 갖고 있는 MDMX 개발과 3D 그래픽 명령어를 갖고 있는 ISA-3D 프로세서를 들수 있다. 그리고 MIPS,의 주시장은 64bit-4000계열의 43xx,45xx,46xx ASE 이다.

VIII. 한국에서 마이크로 프로세서를 해야 하는 이유

1) 21세기 산업을 주도하는 핵심 기술로써 "전자 산업의 쌀" 이라고 할 수 있는 기술임 SoC에서 하나의 MCU는 꼭 사용됨
2) 전 세계적으로 미국, 일본, 영국, 독일, 프랑스 등 소수 선진국만이 독자적인 마이크로 프로세서 아키텍쳐를 소유하고 있음
3) 선진 업체에서는 지속적으로 많은 투자와 연구 개발이 이루어 지고 있음
   국내 반도체 관련 대기업, 학교 및 연구소에서는 지금까지 오랫동안 마이크로 프로세서를 이끌어온 선진회사 (Intel, Motorola, ARM, MIPS 등) 보다 더 나은 것을 개발할 수 없다는 생각으로 대부분의
4) 관련 기술을 선진 회사로 부터 License 계약을 맺어 사용함으로써, 해마다 상당액의 로열티를 지불 하여 오고 있는 실정임.
5) 외국 업체의 의존에서 탈피하고, 외화 절감 및 수출 경쟁력 확보
   국내 마이크로 프로세서의 아키텍쳐 및 관련 기술의 기초 부실, 기술 인력 부족 등으로 인한 전자 산업 국제 경쟁력 약화 과거 정부의 메모리 반도체 개발 국책 사업이 상당한 성공을 거둔

값진 경험을 되살려 시스템 2010 사업이 성공적으로 수행 되어야 함

IX. 에이디칩스 EISC microprocessors

에이디칩스는 EISC architecture를 적용한 다양한 embedded microprocessor를 개발하고 있다.

1. SE1608 (16bit EISC CPU)

8 bit microprocessor를 전체 embedded microprocessor 시장에서 volume market share가 70%에 이르는 대표적 제품이다. Zilog의 Z80과 Intel의 I8051 이외에도 여러 회사가 여러 종류의 8 bit microprocessor core를 공급하고 있다.

또한 16 bit microprocessor도 Hitachi의 H-8, Panasonic의 MN10200 등 많은 제품들이 있어서 8 bit microprocessor와 함께 low end market을 형성하고 있다.

그런데 이들 8/16 bit microprocessor의 instruction set이 C 등 high level language에 적합하지 않아서 대부분 assembler를 사용하여 program을 개발하고 있다. 따라서 gate count가 작고, 속도가 빠르면서 high level language compiler에 적합한 8 bit 또는 16 bit microprocessor가 필요하다. 이러한 시장의 요구에 부응하기 위하여 16 bit Simple EISC microprocessor인 SE1608을 개발하게 되었다.

SE1608은 7개의 general purpose register, 6개의 special purpose register를 가지고 있으며 16 bit fixed length instruction set으로 되어 있다. 또한 16 X 16 = 32 signed/unsigned multiplier와 16 bit barrel shift를 option으로 갖추고 있다.

Gate count는 multiplier와 barrel shift를 포함하여 7K gate이며, VHDL로 설계한 model이 0.35 micron CMOS 공정에서 50MHz clock으로 동작한다.

C/C++ library를 GNU GCC로 compile하여 16 bit microprocessor들의 program size를 비교하면 다음과 같다. Host system은 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. Newlib는 ANSI C standard library인 libc와 floating point library인 libm으로 구분된다. 비교표의 수치는 program size를 byte로 표시한 것이며 괄호안의 수는 SE1608의 program size를 100으로 하였을 때의 상대적인 program size이다.

	Newlib-1.8.0 libc	Newlib-1.8.0 libm	Libstdc++ -2.10.0	Total
SE1608	60,899 (100)	66,630 (100)	135,649 (100)	263,178 (100)
H-8/300	68,840 (126)	93,088 (140)	187,300 (138)	357,249 (136)
MN10200	70,243 (115)	64,100 (96)	163,587 (121)	297,930 (113)

H-8/300은 Hitachi사에서 개발한 16 bit microprocessor이고, MN10200은 Panasonic에서 개발한 16 bit microprocessor이다. 이들은 CISC 구조이며 variable length instruction set를 가지고 있다. 따라서 hardware가 복잡하여서 ASIC library로 제공되지 않는다.

8 bit microprocessor와 비교는 이들의 GCC compiler가 없는 관계로 수행할 수 없었다. 한편 Z80와 I8051을 VHDL로 설계하여 합성하면 각각 10K,

8K gate가 소요된다. 이것은 SE1608에서 multiplier와 barrel shift를 포함하였을 때의 7K gate보다 많은 회로가 필요함을 나타낸다. 즉, SE1608은 기존 8 bit micro-processor보다 훨씬 hardware가 간단하다.

SE1608은 16 bit EISC microprocessor로 기존 8 bit microprocessor보다도 hardware가 간단하며, 16 X 16 = 32 multiplier와 16 bit barrel shift를 가지고 있으므로 low end DSP 대용으로도 사용이 가능하며, 0.35 micron CMOS에서 50MHz로 동작하는 high performance이며, C 등 high level language compiler에 적합한 microprocessor이다.

따라서 기존의 8/16 bit microprocessor를 대치하여 광범위한 시장을 형성할 것으로 전망된다.

2. SE3208 (32bit EISC CPU)

Embedded microprocessor 시장에서 30MIPS 이하 performance의 microprocessor의 점유율이 80%를 넘는 것으로 조사되고 있다.

SE3208은 이러한 low end 32 bit micro-processor 시장을 목표로 개발되었다.

SE3208은 8개의 general purpose register, 6개의 special purpose register를 가지고 있으며, 16 bit fixed length instruction set을 사용한다.

또한 32 X 32 = 32 signed multiplier와 32 bit barrel shift를 갖추고 있다.

Gate count는 multiplier와 barrel shift를 포함하여 25K gate이며, VHDL로 설계한 model이 삼성 0.35 micron CMOS 공정에서 50MHz clock으로 동작한다.

SE3208과 시장 경쟁을 하는 제품으로는 Panasonic의 MN10300, ARM의 ARM-Thumb, LSI logic/NEC의 MIPS-16 등이 있다. MN10300은 variable length instruction set이므로 hardware가 복잡한 단점이 있으며, ARM-Thumb과 MIPS-16은 long length Operand를 표현하기 위하여 PC relative immediate data load instruction을 사용한다. 따라서 SE3208에 비하여 performance가 낮다. Code density는 MN10300은 SE3208보다 약간 낮으며, ARM-Thumb와 MIPS-16은 비슷하다.

SE3208은 32 bit simple EISC microprocessor로 30-50 MIPS 이하의 performance를 필요로 하는 대부분의 embedded market에 적합하도록 개발된 제품이다. 경쟁사에 비하여 hardware가 간단하고, code density가 높거나 같으며, performance가 높은 장점을 가지고 있다.

3. AE32000 (고성능 32bit EISC CPU)

High code density, high performance 32 bit embedded microprocessor이다. 이 시장은 volume은 작지만 high profit market이므로 대부분의 microprocessor 업체들이 주력하는 분야이다.

AE32000은 16개의 general purpose register, 7개의 special purpose register를 가지고 있으며, 16 bit fixed length instruction set을 사용한다.

또한 32 X 32 = 64 signed/unsigned multiplier와 32 X 32 + 64 = 64 multiply and accumulate (MAC) 및 32 bit barrel shift를 갖추고 있으며, DSP instruction과 DSP X/Y memory를 option으로 갖추고 있다.

AE32000은 기존 RISC/CISC microprocessor와 비교하여 code density가 높고, hardware가 간단하며, performance가 높으면서, 전력 소모가 적다.

4. AE64000 (고성능 64bit EISC CPU)

High code density, high performance 64 bit embedded microprocessor이다. 이 시장은 향후 embedded market에서 중요한 위치를 점유할 것으로 전망되므로 몇몇 microprocessor 업체들이 연구하는 분야이다.

그런데 종래 RISC/CISC architecture는 효율적인 64 bit embedded microprocessor를 구현하기 어려운 문제점이 있었다.

이에 반하여 EISC는 16/32 bit 뿐만 아니라 64 bit microprocessor에도 효율적이다.

AE64000은 16개의 general purpose register, 7개의 special purpose register를 가지고 있으며, 16 bit fixed length instruction set을 사용한다.

또한 64 X 64 = 64 signed multiplier와 64 X 64 + 64 = 64 multiply and accumulate(MAC) 및 64 bit barrel shift를 갖추고 있으며, SIMD-DSP instruction과 DSP X/Y memory를 option으로 갖추고 있다.

AE64000은 code density가 높고, hardware가 간단하며, performance가 높으면서, 전력 소모가 적은 64 bit embedded microprocessor이다.

X. 결 론

Architecture는 설계 사상을 종합적으로 의미하는 것으로 instruction set에 의하여 그 사상이 잘 표현되어 진다. Instruction은 operation을 나타내는 Op-Code field와 object를 나타내는 Operand field로 구성된다.

Op-code는 모든 microprocessor에서 공통적이다. 즉, 16/32/64 bit microprocessor에서, 또는 RISC와 CISC 및 이들을 혼용한 중간적 architecture를 가지는 모든 microprocessor에서 Op-code는 공통적인 것들이 많다.

예를 들면 ADD/SUB/Compare/Shift/ Branch/ Shift/AND/OR/XOR 등의 operation은 모든 microprocessor가 가지고 있다.

Operand는 register, offset 및 immediate data를 표현한다. 이러한 Operand의 길이는 필요에 따라서, microprocessor에서 따라서 변화하는데, instruction set의 출현 빈도를 조사하면 short length Operand의 출현 빈도가 높게 나타난다.

EISC는 instruction의 이러한 특성을 효과적으로 나타내는 구조를 가진다. 즉, EISC의 instruction은 16 bit fixed length로 되어 있으며, 출현 빈도가 높은 short length Operand를 가지고 있다. 또한 extension register(ER)와 extension(E) flag를 이용하여 필요에 따라서 instruction의 Operand를 확장한다.

즉, EISC는 16 bit fixed length instruction set을 가지고 필요에 따라서 필요한 길이만큼 Operand를 확장하므로 instruction set의 수가 작으며, hardware가 간단하여 진다.

또한 불필요한 Operand가 instruction에 포함되지 않으면서도 효율적으로 모든 길이의 Operand를 표현할 수 있으므로 code density가 대단히 높은 장점을 가진다.

나아가서 LERI folding circuit을 도입하여 performance를 크게 향상시킬 수 있다.

EISC는 Op-code와 Operand가 다른 성질을 가지는 점에 착안하여 이들을 효율적으로 수용하는 구조를 가지고 있으므로 16/32/64 bit microprocessor 모두에 효율적인 architecture이다.

에이디칩스에서는 EISC architecture를 적용한

16/32/64 bit EISC microprocessor를 개발을 완료했으며, 이들은 hardware가 간단하고, code density가 높으며, 전력 소모가 적고, high performance를 요구하는 embedded 분야에 특히 적합하다.

이 희

1977년 ~ 1981년 인하대학교 전자공학과 졸업
1982년 ~ 1984년 인하대학교 대학원 전자 전공
1983년 삼성전자 반도체 연구소 입사
1986년 삼성전자 전임연구원 (설계팀장)
1990년 삼성전자 선임연구원
1993년 삼성전자 수석연구원
1998년 ㈜에이디칩스 연구소장
2001년 ~ 현재 ㈜에이디칩스 SOC 사업부 부사장 & CTO