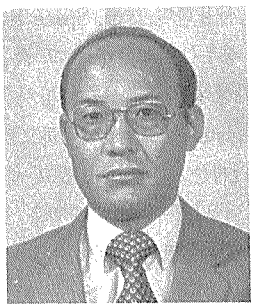


吳 吉 祿
韓國電子通信研究所
컴퓨터기술연구단장/工博

HARP(High-performance Architecture for Rise-type Processor)의 構造設計

Ⅰ. 서 론

반도체 기술의 급격한 발전으로 마이크로 프로세서를 이용하여 수퍼미니급의 컴퓨터를 개발하는 것이 가능하게 되었다. 따라서 프로세서 칩 개발노력이 증대되었으며 컴퓨터 구조 또는 프로세서 구조에 관한 연구도 여러 곳에서 진행되고 있다.

최근에 개발된 새로운 프로세서 구조로서, 미국의 경우 Motorola 사의 MC88000[1], Hewlett Packard 사의 Precision Architecture[2], Sun Microsystems 사의 SPARC[3], Stanford 대학의 MIPS-X[4] 등이 이에 속한다. 이들 프로세서의 특징은 RISC(Reduced Instruction Set Computer)형태를 사용하고 있다는 것이다.[5,6]

일본의 경우 새로운 컴퓨터의 구현은 TRON(The Realtime Operating System Nucleus) 으로 대표되며[7], 이의 특징은 RISC의 반대 개념인 CISC(Complex Instruction Set Computer) 라는 것이다.

우리나라의 경우 독자적인 명령어를 가지는 컴퓨터를 개발하겠다는 노력은 미미하였으며 논문으로 발표된 것은 전무한 상태이다.[8,9] 우리나라가 독자적인 프로세서 구조를 갖는다는 것은 국가적인 차원으로 보아 매우 중요한 문제이므로 여러가지 조건을 감안하여야 한다. 특히 외국 업체와 법적인 문제가 발생하지 않아야 될 것과 개발 기간이 짧아야 될 것 등을 우선적인 조건으로 들 수 있다. CISC 형태의 프로세서는 구조 뿐만 아니라 반도체로서의 구현과정이 복잡하여 개발 기간이 길어지므로, 비교적 짧은 RISC 형태의 프로세서를 개발하는 것이 유리할 것이다. 그리고 한글의 기초 단위인 16비트 데이타를 지원할 수 있어야 하며, 2000년대까지 프로세서 구조의 근본적인 변화가 없어야 할 것이므로 가상 어드레스 확장의 여지를 남겨 두어야 할 것이다.

본 논문은 독자적인 구조를 갖는 32비트 프로세서 설계에 관한 것이다. Ⅱ장에서는 프로세서의 기본구조 설계를 위하여 1980년대 이후에 등장한 RICS 형태의 프

로세서들에 대한 사례 검토의 결과를 기술하였다. Ⅲ장에서는 현재까지 연구가 진행된 RISC 형태의 32비트 프로세서인 HARP(High-performance Architecture for Rise-type Processor) 의 기본 구조, 즉 명령어 및 데이타 형식, 레지스터 구성, 가상 어드레스 방식, 어드레싱 모드, 명령어 설계 등에 관하여 기술한다. 마지막으로 Ⅳ장에서는 현재까지의 연구업무에서 얻은 결론과 앞으로의 진행 방향에 대하여 기술하도록 한다.

Ⅱ. 사례연구 및 기본 구조 설계

컴퓨터 또는 프로세서 구조라고 하는 용어는 이를 사용하는 사람에 따라 다르게 해석된다. Flynn이 사용한 정의에 프로세서 구조는 명령어 세트(Instruction Set)만을 포함하였고, 그 외의 사항은 구현(Implementation) 사항으로 정의하였다.[10] 이와 같은 정의는 RISC 형태의 프로세서 구조를 표현하기에는 부족한 점들이 많다. 예를 들어 지연분기일 경우 "NOP(No Operation)" 명령어를 사용하여야 한다는 사실은 명령어 세트만을 포함하는 프로세서의 구조 정의로는 설명할 수가 없기 때문이다. 따라서 본 논문에서는 명령어 세트 및 프로세서 동작에 필요한 구현에 관한 정보의 합으로서 프로세서의 구조를 정의하기로 한다. 즉, 프로세서 구조는 명령어 세트 뿐만 아니라 Memory Management, 파이프라인 처리 등의 정보를 포함한다.

본 사례연구에서는 여러 칩들을 모두 검토하기 보다는 프로세서 구조의 기본설계에 관계되는 부분에 대한 사례를 검토하여 HARP의 기본구조를 갖추도록 하였다.

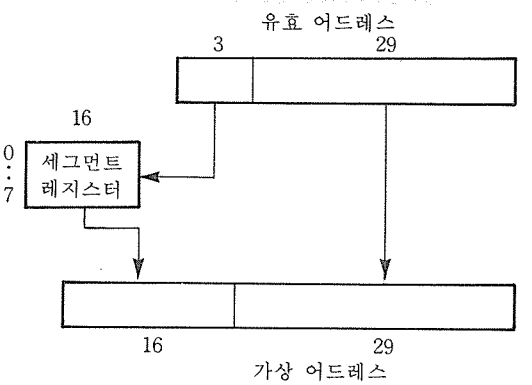
1. 가상 어드레스 변환 방식

1980년대 초에 개발된 32비트 프로세서의 경우 가상 어드레스의 크기는 대략 32비트(2^{32}= 4 Gigabyte) 로 제한되었다. 그러나 인공지능과 영상처리 등의 매우 많은 데이타 영역을 필요로 하는 응용분야에서는 4 Gigabyte로 한계를 느끼고 있으며, 최근에 개발된 프로세서들은 이를 해결하기 위하여 32비트 이상의 가상 어드레스 영역을 제공하고 있다.

Berkeley 대학에서 AI Workstation 용도로 개발한 SPUR의 경우 상위 2비트가 세그먼트를 가르키도록 하여 가상 어드레스를 38비트로 확장하였으며,[11] ROMP의 경우 SPUR와 같은 방식을 이용하나 4비트를 세그먼트 지정에 사용하여 세그먼트의 수를 16개로 확장하였다.[12] Precision Architecture의 경우 다음과 같은 두가지 다른 방식을 사용한다. Opcode 필드 중 2비트를 Space Register 필드로 사용하여 첫째, 이 필드의 값이 0인 경우에는 32비트의 상위 2비트에 4를 더하여 Space Register의 Index로 사용한다. 둘째, Space Register 필드의 값이 0이 아닐 경우(1, 2, 3,)에는 그 값을 직접 Index로 사용한다. 이 가상 어드레스 변환 방식은 명령어 필드 중에 Space Register 필드를 사용하므로 가상 어드레스 계산에서 한 단계를 절약할 수 있고, 세그먼트의 크기를 32비트 전부 이용할 수 있는 장점을 갖고 있으나 명령어 필드 중 2～4비트를 할애하여야 한다는 문제점을 갖고 있다.

가상 어드레스 변환방식에서는 세그먼트 레지스터의 수와 크기를 정의하여야 한다. 세그먼트의 수는 User Code, System Code, 스택, 그리고 데이타 등 4가지 영역을 각각 하나씩 4개의 세그먼트로 정의할 수 있으나 데이타 영역에서 Array의 처리에 하나 이상의 세그먼트를 사용하고자 할 경우에는 5개 이상의 세그먼트가 필요하게 되어 문제가 발생한다.

따라서 8개의 세그먼트 사용이 적절하다고 판단되며 그 크기는 16비트로 하되 추후 32비트로 늘릴 수 있는 여지를 남겨 두는 것이 적당하다. 이와 같은 가상 어드레스 변환방식을 도시하면 〈그림 1〉과 같다.



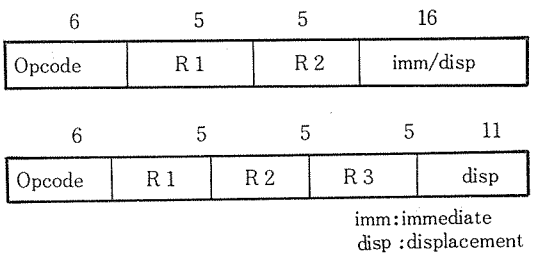
〈그림 1〉 가상 어드레스 변환 방식

2. 레지스터 수

범용 레지스터의 수는 16개(MIPS), 32개(HP), 또는 256개(AM29000[3]) 등이 사용된다. RISC나 SPUR칩의 경우에는 8개의 Over Lapped Window를 사용하여 138

개의 레지스터에 매핑하여 사용한다. 이와 같은 방식은 명령어 형식을 정하는데 있어서는 16개의 레지스터를 사용하는 것과 유사한 결과를 갖는다. MIPS칩을 더욱 발전시킨 MIPS-X에서는 16개의 레지스터를 32개로 늘려서 사용하였다. 16개의 레지스터로는 별도의 부동소숫점 연산을 위한 프로세서가 존재하지 않는 한 부동소숫점 처리면에서 2개의 레지스터로 64비트의 부동소숫점 데이타를 저장하여야 하므로 부족함을 느낀다. 다만 레지스터의 수를 32개로 정할 경우 Opcode의 길이를 6비트로 하여야만 16비트의 데이타 필드를 사용할 수 있다는 문제를 가지고 있다.

MIPS의 경험을 살린다면 범용 레지스터의 갯수는 32개로 하고 Opcode의 길이는 6비트를 사용하도록 하는 것이 바람직하다. 다만 Opcode 필드가 모자라는 것을 고려하여 필요시에는 Opcode 필드를 확장할 수 있도록 한다 이와 같은 방식을 도시하면 〈그림 2〉와 같다.



〈그림 2〉 명령어의 일반 형식

3. 어드레싱 모드 지원

어드레싱 모드는 명령어 세트의 설계에 근간이 되는 주요사항 중의 하나이다. CISC 프로세서는 갈수록 복잡한 어드레싱 모드를 지원하는 방향으로 설계되고 있으며, RISC 프로세서인 경우에는 아주 적은 수의 어드레싱 모드만을 지원한다. 극단적인 예로 AM 29000의 경우 Register Direct, Register Indirect와 Immediate Addressing Mode의 3가지만을 지원한다.

모든 RISC 형태의 프로세서는 Load/Store 형태의 구조를 갖고 있으며, 일단 데이타를 레지스터에 Load시킨 후에 ALU 명령어를 레지스터간에 사용한다. 다만 상수의 경우에는 별도로 레지스터에 Load시키지 않고도 사용할 수 있다.

MIPS

Load/Store

R=M [R1+disp_16]

R=M [R1+R2]

R=M [R1<<R2]

R=M [adr16]

32 bit data only

Branch/Call

Pc=Pc+offset_ 26

Pc=Pc+offset_ 5 (Conditional)

Pc=adr_26

Pc=M [R1+offset_16]

RISC/SPUR

Load/Store

R=M [R1+R2]

R=M[R1+disp_13]

R=M [Pc+disp_19]

(byte, short) (signed, unsigned) support.

Branch/Call

Pc=R1+R2

Pc=R1+disp_14

Pc=adr_28

ROMP

Load/Store

R=M [R1+disp_16]

byte, short support

Branch/Call

Pc=adr_24

Pc=R1+disp_20

Pc=R1

AM29000

Load/Store

R=M [R1]

Branch/Call

Pc=Pc+disp_16

Pc+R

〈그림 3〉 RISC프로세서의 어드레싱 모드

어드레싱 모드는 Load/Store와 분기(Branch) 명령에 있어 다르게 사용되며 몇 가지 프로세서의 예를 들면 〈그림 3〉과 같다.

ALU 명령에 상수를 사용하는 것은 비록 16비트 이내의 데이타에 제한된다 하더라도 대개의 상수가 작은 수

임을 고려할 때 필요한 모드라고 판단된다.

Load/Store의 어드레싱 모드에는 최소한 M[R1+R2] 또는 M [R1+disp]를 처리하도록 해야 Base Register를 사용하는 Argument Passing, Array Handling 등에 있어 처리속도를 높일 수 있다. 32비트 명령어 형식의 Load/Store 명령에서는 32비트의 상수 값을 엑세스 하는 적절한 방법이 없으므로 이는 레지스터를 사용하여 두 개의 명령어를 이용하여 처리하도록 한다. 이와 같은 방식은 RISC의 대표적인 예로 꼽히는 VAX의 어드레싱 모드 사용 예를 보아 충분한 타당성이 있는 것으로 보인다.

Branch/Call 명령은 Absolute, Relative, Indirect Addressing Mode 등이 모두 필요하다. 이 경우 변위의 크기는 Opcode 형식을 확정할 시에 결정된다.

4. 분기 처리 구조

일반적인 RISC 프로세서들은 파이프라인 방식을 사용하여 분기 명령 외에는 가능한 한 1사이클에 하나의 명령을 수행하도록 하고 있다. 분기 명령의 경우에는 다음 어드레스를 미리 알 수 없으므로 한 사이클에 수행하기가 어려우며 이에 따른 성능상의 손실이 크기 때문에 여러가지 기술을 사용하여 이 문제를 해결하고 있다. 또한 분기 명령은 Condition Code의 유무에 따라 그 처리방식이 달라진다. 기존의 범용 CISC 프로세서는 예외없이 Condition Code를 사용하고 있으며, RISC와 같은 계열의 SPARC, 그리고 CRISP에서 사용된다.[14]

MIPS와 HP Spectrum에서는 Condition Code를 사용하지 않고 분기 명령과 Condition Code를 방생시키는 명령을 하나로 합성하여 사용한다. 예로 "Compare and Branch"명령을 들 수 있다. 또한 명령어의 사이클을 낭비되는 것을 막기 위해 지연분기 또는 Instruction Cancelling 방식을 사용한다. 이보다 더욱 발전된 방법으로는 하드웨어를 사용하여 Branch Address의 내용을 캐쉬에 저장하여 이를 이용하도록 하는 것으로 CRISP과 AM29000에서 사용하고 있다. 또한 분기 명령에 Prediction Indicator를 사용하는 방식도 고려되고 있다.

Branch Target Cache 방식은 하드웨어로 해결하는 방식이므로 이는 구현의 일부라 할 수도 있으므로 구조에서 꼭 정의할 필요는 없는 것으로 판단된다. 지연분기나 Instruction Cancelling 방식은 근본적으로 같은 개념이며 이를 위해서는 컴파일러에서 분기 명령 전후에 유효한 명령 코드를 생성하여야 한다.

HARP의 경우에는 가능한 한 하드웨어의 부담을 줄이기 위하여 Condition Code의 사용을 억제하고 Delayed Branch 개념을 사용토록 한다.

5. 부동소숫점(Floating Point) 연산

부동소숫점 연산은 단순한 작업이 아니므로 여러 사이클이 소요되며 사이클 수도 데이타에 따라 달라지기 때문에 RISC의 "1 사이클 1 명령어 수행"이라는 근본 개념에 위배된다. 이를 해결하기 위해서 여러가지 방식이 사용되는데 Coprocessor를 사용하는 방식이 가장 보편적이며, 이는 RISC 프로세서의 기본구조에 영향을 주지 않는 방식이다. Coprocessor를 사용하는 방식을 취하더라도 스스로 부동 소숫점 연산을 처리할 수 있는 여지를 두는 것이 보통이며, 이를 위하여 Multiply Step Divide Step과 같은 명령어를 두는 것이 상례이다. Clipper와 Motorola 88000에서는 부동소숫점 데이타를 CPU에서 직접 처리한다.[5] 따라서 일반 명령문에도 Multi-Cycle 명령을 허용하고 있으며 파이프라인의 효과가 많이 감소된다.

6. LISP 언어 처리

인공지능 분야에서 가장 많이 쓰이는 LISP 언어는 기존의 FORTRAN, C 등의 언어와 아주 다른 방식으로 처리되며 이를 지원하기 위해서는 프로세서 구조의 변경이 필요하다. 첫째 Procedure Call, 특히 Recursive Call이 빈번하게 일어난다. 따라서 Window 사용하는 개념을 구조에서는 Window Register를 메모리에 자주 옮겨야 하므로 Window 개념의 장점을 살릴 수 없게 된다. 둘째로 Type Checking이 자주 발생하며 이를 신속히 처리하기 위해서는 Tagged 구조에 대한 하드웨어의 지원이 필요하다. Tagged 구조는 일반 프로세서 구조와 레지스터의 크기, Datapath의 크기 등 모든 사항이 다르므로 새로운 프로세서 구조에서 이를 수용할 경우 범용 목적으로는 많은 손실을 감수하여야 할 것이다. 따라서 HARP에서는 Tagged 구조는 지원하지 않으며, 이를 필요로 할 경우 별도로 새로운 프로세서 구조를 정의하기로 한다. 다만 명령어에 Tag를 사용하기에 편리하도록 하는 명령의 포함을 검토하기로 한다.

7. 데이타 형식

프로세서 구조에서 데이타 형식에 대해 정할 사항은 Little Endian 방식이냐 Big Endian 방식이냐 하는 것이다. 현재 우리나라에서 중요한 위치를 차지하고 있는 행정전산망 주전산기에서는 Big Endian 계열인 Motorola 68030을 사용하고 있으며 이의 입출력 버스 역시 Big Endian 계열의 VME 버스를 사용하고 있다. 그러나 IBM PC에서 사용되는 Intel의 80X86 계열과 기타 상당수의 프로세서가 Little Endian이다.

Ⅲ. HARP의 구조 설계

1. 개요

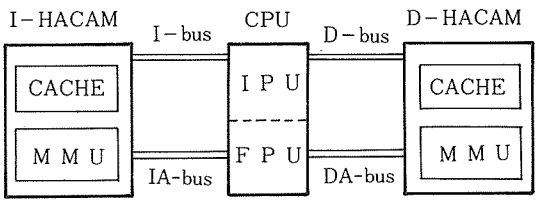
HARP는 HLL(High Level Language), 한글 처리 등에 사용될 수 있는 범용 프로세서로서 CPU의 개발 기간을 단축하고, 명령어 처리 속도를 향상시킬 수 있는 RISC 형태의 Load/Store Machine으로 워드(Word) 를 기본 단위로 취한다. 모든 명령어는 고정된 32비트 명령어 형식을 사용한다. 명령어 형식은 8가지 형식으로 이루어져 있으며, Compare and Branch 명령어를 사용하여 Condition Code의 기능을 대신 한다. 현재 〈表1〉과 같이 39개의 명령어를 정의하였다.

HARP에서는 32비트의 범용 레지스터와 특수 레지스터가 사용되며, 48비트의 가상 어드레스 영역을 제공하기 위해 세그먼트 레지스터가 사용된다. 그리고 가상어드레스의 필드가 크기 때문에 어드레스 변환시 페이지 테이블 방식 대신에 HT(Hashing Table)과 IPT(Inverted Page Table)를 이용한 방식을 사용하였다.[15]

HARP의 기본 Module은 〈그림 4〉와 같이 CPU, 명령어 캐쉬 및 데이타 캐쉬로 구성된다.

CPU는 IPU(Integer Processing Unit)와 FPU(Floating Point Unit)로[17] 이루어지며, 캐쉬(Cache)는 MMU(Memory Management Unit)와 함께 하나의 칩으로 설계되는데, 이 칩을 HACAM(HArp CAche and Mmu)이라 한다. HACAM은 명령어와 데이타 캐쉬에 같은 칩이 사용될 수 있도록 설계되었으며, 명령어 캐쉬에 사용되는 HACAM을 I-HACAM, 데이타 캐쉬에 사용되는 HACAM을 D-HACAM이라 한다.[16] HACAM에는 어드레스 변환시간을 줄일 수 있도록 Virtual Cache가 사용되며, 캐쉬 크기는 8K Byte이지만 필요에 따라 HACAM칩의 수를 늘려 캐쉬의 크기를 증가시킬 수 있다.

CPU와 HACAM 사이에는 4개의 버스(Instruction



〈그림 4〉 HARP의 Module

Address Bus, Instruction Data Bus, Data Address Bus, Data Bus)가 사용된다. 그리고 대부분의 명령어들이 수행되는 5클럭 사이클이 소요되며, 파이프라인을 이용하므로 한 사이클 마다 하나의 명령어가 수행된다. 파이프라인은 IF(Instruction Fetch), ID(Instruction Decode) EX(EXecute), MA(Memory Access), 및 WB(Write Back)의 5단계로 구성되며 각 단계는 1클럭 사이클이 소요된다.[18]

2. 명령어 및 데이타 형식

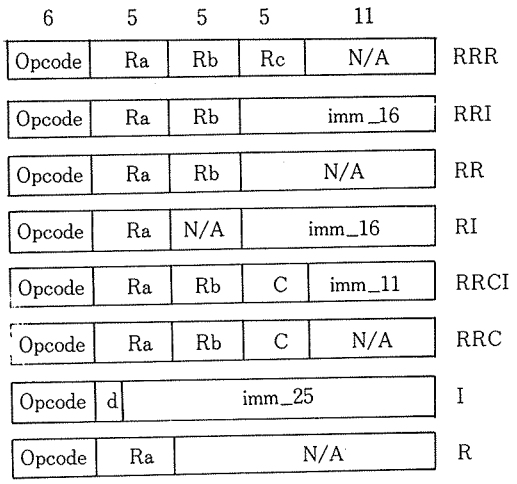
RISC 프로세서의 특징 중의 하나인 고정된 명령어 형식(Fixed Length Instruction Format)은 가변 명령어 형식(Variable Length Instruction Format)에 비해 메모리 사용 효율이 떨어 지지만, 명령어 디코딩을 쉽고 빠르게 할 수 있으며, 파이프라인을 용이하게 한다는 장점을 갖고 있다.

HARP의 모든 명령어는 단순성, 규칙성 및 명령어수행의 효율성을 위하여 고정된 32비트 명령어 형식을 사용한다. 명령어 형식은 〈그림 5〉와 같이 8가지 형식으로 이루어져 있다. 이러한 명령어 형식은 6비트의 Opcode를 이용하여 64개의 명령어를 사용할 수 있으며, 레지스터를 지정한 필드를 5비트로 32개의 레지스터를 지정할 수 있다. 범용 레지스터와 특수 레지스터의 지정은 명령어에 의해서 구별된다. 분기 명령어를 제외하고, 상수는 한글 처리에 유용하도록 모두 16비트가 사용된다. 분기 명령어의 d비트는 지연구간(Delayed Slot) 의 유무를 나타내는 데 사용되며, Code-Reorganizer에 의해 d값이 결정된다.[19]

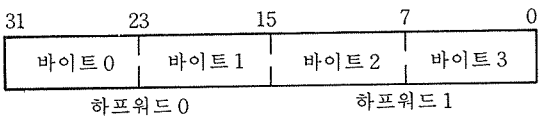
HARP에서 바이트는 8비트, 하프워드는 16비트, 워드는 32비트로 구성된다. 데이타 형식은 MC68000 계열과 호환성을 유지할 수 있도록 〈그림 6〉과 같이 Big Endian 방식을 사용하며 메모리 액세스는 워드를 기본 단위로 한다.

〈表 1〉 HAPP의 어셈블리 명령어

Opcode	Inst	Operands	Action
	LD	Ra, Rb, Rc	Ra<−M[Rb+Rc]
	LDD	Ra, Rb, disp_16	Ra<−M[Rb+disp 16]
	LDH	Ra, imm_16	Ra[31 : 16 <−imm 16
	STD	Ra, Rb, disp_16	Ra−>M[Rb+disp 16]
	RD	Ra, SR, imm_16	Ra<−SR+imm 16
	WR	Ra, SR, imm_16	Ra+imm −>SR
	ADDU	Ra, Rb, Rc	Ra<−Rb+Rc
	ADDIU	Ra, Rb, imm_16	Ra<−Rb+imm 16
	ADDS	Ra, Rb, Rc	Ra<−Rb+Rc
	ADDIS	Ra, Rb, imm_16	Ra<−Rb+imm 16
	SUBU	Ra, Rb, Rc	Ra<−Rb−RC
	SUBIU	Ra, Rb, imm_16	Ra<−Rb+imm 16
	SUBS	Ra, Rb, Rc	Ra<−Rb−Rc
	SUBIS	Ra, Rb, imm_16	Ra<−Rb−imm 16
	AND	Ra, Rb, Rc	Ra<−Rb(and) Rc
	ANDI	Ra, Rb, imm_16	Ra<−Rb(and) imm 16
	NOT	Ra, Rb	Ra<−(not) Rb
	OR	Ra, Rb, Rc	Ra<−Rb(or) Rc
	ORI	Ra, Rb, imm_16	Ra<−Rb(or) imm 16
	XOR	Ra, Rb, Rc	Ra<−Rb(xor) Rc
	XORI	Ra, Rb, imm_16	Ra<−Rb(xor) imm 16
	SHD	Ra, Rb, Rc	shift double
	SRA	Ra, Rb, Rc	Ra<−Rb>>Rc
	EXTB	Ra, Rb, cond	Ra<31 : 08> <−0, Ra<07 : 00> <−byte BP<01 : 00> of Rb or byte cond. of Rb
	EXTH	Ra, Rb, cond	Ra<31 : 16> <−0, Ra<15 : 00> <−halfword BP<00<of Rb or halfword cond of Rb
	INSB	Ra, Rb, cond	byte BP<01 : 00> of Ra <−Rb<07 : 00> or byte cond of Ra
	INSB	Ra, Rb, cond	half word BP<00> of Ra <−Rb <15 : 00> or halfwork cond of Ra
	JMP	imm_25	PC<Pc+imm 25
	JMPR	Ra, Rb	PC<−Ra+Rb
	JMPC	imm_25	if carry PC<−PC+imm 25
	BR	Ra, Rb, cond, disp_11	if (Ra cond Rb) PC<−PC+disp 11
	BRR	Ra, Rb, cond	if (Ra cond Rb) PC<−PC+BR
	TRAP	Ra, Rb, cond, number	if (Ra cond Rb) PC<Constant* number
	TS	Ra, Rb, Rc	Ra<−M[Rb+Rc], M[Rb+Rc]<00><−1
	PUSH	Ra	SP<−SP −4, M[SP] <−Ra
	POP	Ra	Ra<−M[SP] <−SP+4
	PUSHSR	SR	SP<−SP −4, M[SP] <−SR
	POPSR	SR	SR<−M[SP], SP<−SP+4
	RTU	SR	PC<−SR



〈그림 5〉 명령어 형식



〈그림 6〉 데이타 형식

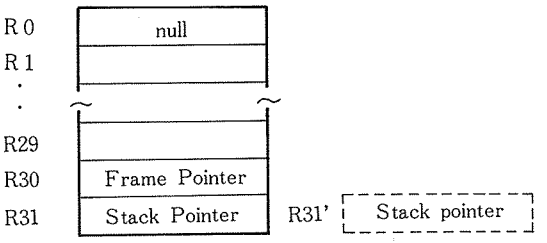
3. 레지스터 구성

프로세서 내에 범용 레지스터가 몇개 있는 가에 따라 프로세서의 구조와 성능이 많이 달라지게 된다. 너무 적은 수의 범용 레지스터를 가지고 있으면 변수를 레지스터에 다 저장하지 못하여 수행시간이 길어지게 되고, 반면에 범용 레지스터가 너무 많으면, 콘텍스트 스위칭시 많은 수의 레지스터를 옮겨야 하므로 역시 성능이 저하될 수 있다. HARP에서는 32개의 32비트 범용 레지스터와 32개의 32비트 특수 레지스터 및 8개의 18비트 세그먼트 레지스터를 사용한다.

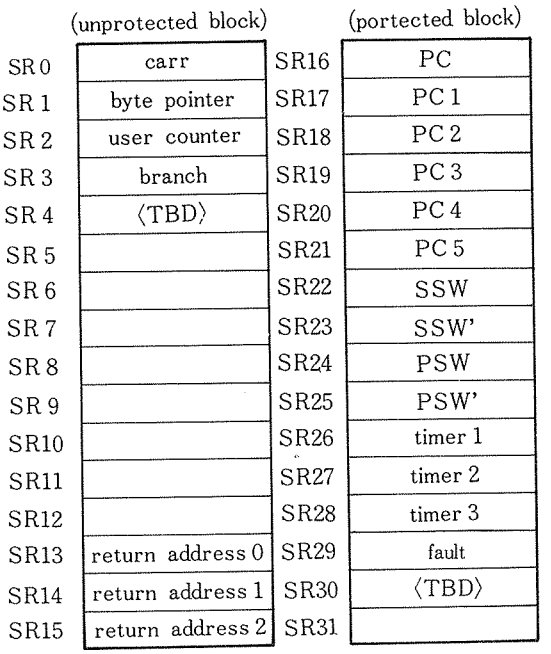
가. 범용레지스터

HARP에서는 32개의 32비트 범용 레지스터가 사용되며, 〈그림 7〉은 범용 레지스터의 구성을 나타낸다. 이들은 어셈블러 신택스에서 Rn으로 표시되며, n은 0에서 31사이의 정수이다.

레지스터 R0는 0 (Unll)레지스터를 의미하며, 0으로 Read되고 Write도 가능하나 R0의 값은 변하지 않는다. 레지스터 사이의 데이타 이동은 R0를 이용하여 ADD/



〈그림 7〉 범용 레지스터



〈그림 8〉 특수 레지스터

SUB 명령어로 수행할 수 있으며, 레지스터의 Clear 등에 아주 효과적으로 사용된다.

레지스터 R1에서 R29는 어큐뮬레이터(Accumulator), 임시 데이타 보관 영역, 어드레싱 모드에서의 베이스 레지스터 등으로 사용된다. R29는 1 Kbyte 이상이 되는 조건부 분기를 수행할 경우 분기 어드레스를 일시 저장하는 장소로도 사용된다. 따라서 분기 명령어의 바로 앞에서는 R29 레지스터를 사용하지 않도록 한다.

레지스터 R30과 R31은 특수 용도로 사용되는 범용 레지스터로 각각 프레임 포인터(Frame Pointer)와 스택 포인터(Stack Pointer)를 나타낸다. Supervisor 모드에서 사용되는 R31과 같은 번호로 시스템 스택 포인터(System Stack Pointer) R31'이 사용된다.

나 특수 레지스터

HARP는 〈그림 8〉과 같이 어셈블러 신택스에서 SR0에서 SR31로 표시되는 32개의 특수 레지스터를 가지고 있으며, 이들은 사용자 모드와 Supervisor 모드에서 사용 될 수 있는 부분(Unprotected Block)과 Supervisor 모드에서만 사용될 수 있는 부분(Protected Block)으로 나누어 진다. 명령어의 레지스터 지정 필드에서 MSB가 0이면 보호되지 않는 블럭을 나타내며, 1이면 보호된 블럭을 가리킨다. 특수 레지스터와 범용 레지스터 사이의 데이타 이동은 RD/WR 명령어에 의해서만 가능하다.

레지스터 SR0에서 SR15는 보호되지 않은 블럭으로 캐리, 바이트 포인터, 사용자 카운터 및 분기 레지스터 등으로 구성되어 있으며, 사용자 모드에서도 RD/WR 명령어를 이용하여 그 내용을 변경할 수 있다.

HARP는 비트 오퍼레이션에 대한 명령어가 없으므로 캐리 비트를 PSW(Program Status Word)에 두지 않고, 특수 레지스터 중의 하나를 캐리 저장용으로 사용한다. SR0는 캐리 레지스터(Carry Register)로 캐리가 발생하면 이 레지스터의 LSB에 자동적으로 저장되며, 이 캐리의 내용은 캐리를 변화시키는 다음 명령어가 수행되기 이전까지 보존된다.

바이트 포인터 레지스터(Byte Pointer Register) SR1은 데이타의 바이트 오퍼레이션을 위하여 사용되며, 계산된 어드레스의 하위 2비트가 SR1의 하위 2비트에 저장된다. 이 값은 추출(Extraction)과 삽입(Insertion) 명령어에서 바이트를 지정하는 값으로 사용될 수 있다.

SR2는 사용자 카운터 레지스터(User Counter Register)로 SHD(SHift Double) 명령어를 사용할 때 이동시키는 값을 저장하는 데 사용되며, 31번째 비트까지 좌우로 이동시키는 것이 가능해야 하므로 하위 5비트가 사용되며 나머지는 0이 된다.

분기 레지스터(Branch Register) SR3는 분기의 범위가 현재의 위치에서 1 Kbyte가 넘어 갈 때 사용된다. BR 명령어는 1 Kbyte 미만의 범위에서 분기할 경우에는 유용하지만 변위에 제약을 받게 되므로 그 이상에서는 BRR 명령어를 사용하여 분기 레지스터에 있는 내용에 따라 분기를 할 수 있다. 그러나 이 경우는 분기할 어드레스를 범용 레지스터에서 분기 레지스터에 이동시켜야 한다.

SR4부터 SR12는 기타 필요한 경우에 사용될 수 있도록 남겨둔다.

SR13부터 SR15는 익셉션 처리 후 수행할 3개의 명령어 어드레스를 스택으로부터 PC에 Write하기 전에 일시 저장하는 데 사용되는 레지스터이다.

레지스터 SR16에서 SR31까지는 Supervisor 모드에서만 내용을 변화시킬 수 있으며, PC, SSW(System Status Word), PSW, Timer 등이 있다.

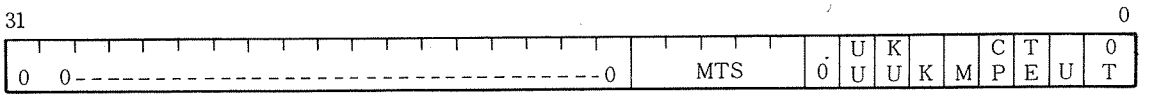
PC와 PC1, PC2, PC3, PC4 및 PC5는 체인(Chain)으로 구성되어 있으며, 익셉션이 발생했을 때 파이프라인이 파괴되지 않고 일을 계속적으로 수행할 수 있도록 파이프라인에서 수행되고 있는 명령어들의 어드레스를 저장한다. PC1은 IF 단계, PC2는 ID 단계, PC3는 EX 단계, PC4는 MA 단계 그리고 PC5는 WB 단계에 있는 어드레스를 저장하고 있다.

SSW는 〈그림 9〉와 같이 구성된다. U(User)비트가 0이면 프로세서가 Supervisor 모드로 변환되어 모든 레지스터의 액세스가 가능하며, 1이면 사용자 모드가 되어 보호된 레지스터의 Read는 가능하나 Write는 할 수 없다. 이때 Write 할 경우 Privileged Violation Exception이 발생한다. CP(Co-Processor) 비트는 Co-Processor의 존재 유무를 표시한다. M(Moce) 비트는 어드레스를 변환할 때의 모드를 나타내며, (K(Protect Key) 비트는 메모리 액세스 프로텍션 코드(Protection Code)를 선택하기 위하여 사용한다. UU(User Data Mode) 비트는 Supervisor 모드에서 사용자 데이타 영역을 액세스할 때 사용하며, KU(User Protect Key) 비트는 Supervisor 모드에서 UU비트가 Set되어 있을 때 K비트 대신에 사용하는 비트이다. MTS(Memory Trap Status) 필드는 메모리에서 발생하는 트랩의 필드로 상위 2비트는 I-HACAM과 D-HACAM를 구별하는 데 사용되며 하위 3 비트는 트랩의 상태를 나타낸다.

PSW는 〈그림 10〉과 같이 구성된다. EL(Exception Level) 비트들은 익셉션 레벨을 표시하며, T 비트는 프로그램에서 한 명령어를 수행한 후 익셉션을 발생시켜 명령어를 추적(Trace)하거나, 프로그램을 수정(Debugging)하는데 사용된다. IL(Interrupt Level) 비트들은 인터럽트 레벨을 나타내는 것으로 인터럽트 입력을 선택적으로 구동시킨다.

PSW'와 SSW'는 익셉션 발생시 PSW와 SSW의 내용을 일시 대피하는데 사용된다.

Timer 레지스터는 시스템 클럭을 위한 것으로 Timer1



- OT (interger overflow trap) bit
- U (user) bit
- TE (trap enable) bit
- CP (co-processor) bit
- M (mode) bit
- K (protect key) bit
- KU (user protect key) bit
- UU (user data mode) bit
- MTS (Memory trap status) field

〈그림 9〉 SSW (System Status Word)

- EL (exceptionlevel) field
- T (trace) bit
- IL (inturrupt level) field

〈그림10〉 PSW (Program Status Word)

이 하위 워드를 저장하는 데 사용되고 차례로 Timer2, Timer3가 사용되며, Fault 레지스터는 익셉션 발생시 Fault가 발생한 명령어의 어드레스를 저장하는 데 사용된다.

여분의 레지스터는 보조 프로세서(Co-Processor) 와 기타 필요한 경우에 사용할 수 있도록 남겨 두었다.

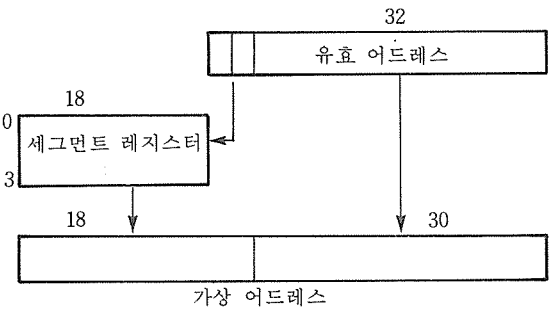
다. 세그먼트 레지스터

세그먼트 레지스터는 HACAM 내에 있으며 48 비트의 가상 어드레스를 만드는데 사용된다. 명령어 및 데이타 어드레스 영역에 대해 각각 4 개의 세그먼트를 사용 하여 큰 가상 어드레스 영역을 제공할 수 있도록 하였다.[16]

4. 가상 어드레스 방식

프로세서의 응용범위가 넓어짐에 따라 기존의 32 비트의 가상 어드레스에서는 한계를 느끼고 있으며, 최근에 개발된 몇 개의 프로세서들은 32 비트 이상의 가상 어드레스 영역을 제공하고 있다.

HARP는 18 비트의 세그먼트 레지스터를 사용하여 48 비트의 가상 어드레스 영역을 제공한다. 메모리를 액세스할 경우에는 CPU 칩에서 32 비트의 유효어드레스가 계산되어 HACAM으로 보내지며, 유효 어드레스는 HACAM 내부에 있는 18 비트의 세그먼트 레지스터와 묶여져 48 비트의 가상 어드레스로 변환된다. 그리고 유효 어드레스 필드를 절약하기 위하여 세그먼트 레지스터를 명령어와 데이타 어드레스 영역에 각각 4 개씩 할당하여 유효 어드레스의 상위 2 비트가 4 개의 세그먼트 레지스터 중에서 1 개를 선택하도록 한다. 〈그림 11〉은 가상 어드레스 사용 방법을 나타낸 것으로, 명령어일 경우에는 I-HACAM에서 수행되며 데이타일 경우에는 D-HACAM에서 수행된다.

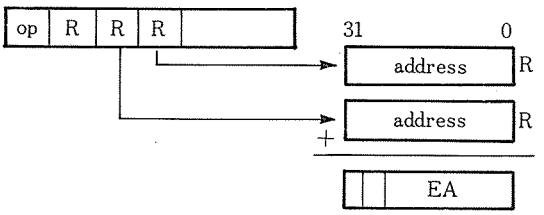


〈그림 11〉 가상 어드레스 사용 방식

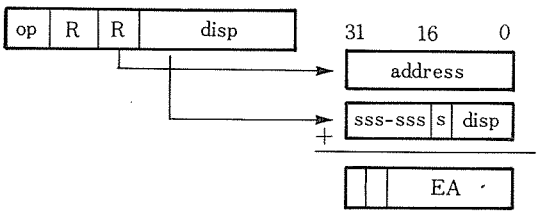
5. 어드레싱 모드

HARP의 어드레스 지정은 언제나 워드 단위로 이루어 진다. 메모리 액세시스 어드레스의 하위 2비트는 사용되지 않으며, 그 2 비트는 바이트 포인터 레지스터에 저장된다. 그리고 Load/Store 형태의 구조를 갖고 있으므로 데이타를 레지스터에 Load시킨 후에 ALU 명령어를 사용할 수 있다. 다만 상수의 경우에는 레지스터에 Load시키지 않고 직접 사용할 수 있다.

Load/Store 명령어에 사용되는 어드레싱 모드는 1사이클에 1명령어 수행 원칙에 벗어나지 않는 범위에서 Argument Passing, 어레이 처리 등에 유용하게 사용되는 Register Indirect, Register Indirect with Displace-



(a) register indirect



(b) register indirect with displacement

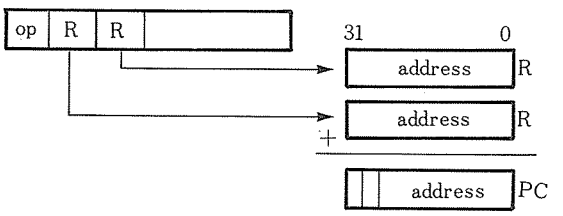
〈그림 12〉 Load/Store 어드레싱 모드

ment와 Immediate Addressing의 세가지 방법을 채택하였다. 〈그림 12〉와 같이 Register Indirect 어드레싱 모드에서는 명령어의 레지스터 필드에 의해 지정된 레지스터 내용의 합이 오퍼랜드의 어드레스가 된다. 그리고 Register Indirect Wigh Displacement에서 오퍼랜드의 어드레스는 레지스터 필드가 지정한 레지스터 내용에 16 비트 변위를 합하여 얻어지며, 16 비트 변위는 어드레스 계산에 사용되기 전에 32 비트로 부호가 확장된다. Immediate Addressing의 경우는 명령어 내에 있는 상수값이 직접 오퍼랜드로 사용된다. 만일 상수의 크기가 16 비트 이상일 때는 2번 Load하여야 한다.

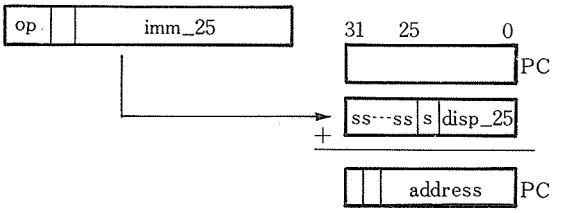
분기 명령에서는 〈그림 13〉과 같이 4가지 어드레싱 모드가 사용된다. PC Direct는 레지스터 필드에 의해 지정된 레지스터 내용의 합이 새로운 PC의 값이 된다. PC Direct with Long Displacement는 25 비트의 변위값이 PC의 내용과 더해져 새로운 PC 값을 만들며, PC Direct with Short Displacement는 PC의 내용에 11 비트의 변위값이 더해져 새로운 PC 값을 만든다. PC Direct with Branch Register는 PC에 분기 레지스터 내용을 더하여 새로운 PC 값을 만든다.

6. 명령어 설계

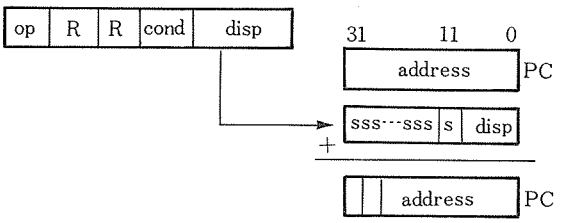
HARP의 IPU에서 설계된 대부분의 명령어는 1 클럭 사이클에 수행이 가능하며 현재 39개의 어셈블러 명령어를 정의하였다.



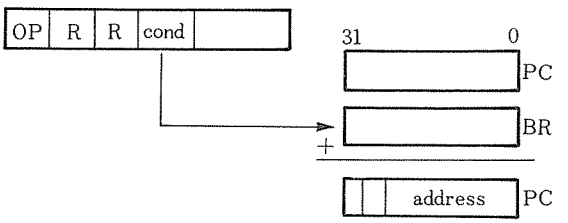
(a) PC direct



(b) PC direct with long displacement



(c) PC direct with short displacement



(d) PC direct with branch register

〈그림 13〉 분기 명령어의 어드레싱 모드

가. Load/Store 명령어

HARP는 외부 데이타 레퍼런스에 대해 Load/Store 구조를 가지고 있으며, 칩 외부와의 데이타 이동은 범용 레지스터에 의해서 이루어 진다. 이 구조는 쉽게 V-LSI로 구현할 수 있으며 파이프라인의 효율을 증가 시킨다.

Load/Store 명령어는 외부 메모리와 범용 레지스터 사이에서 데이타를 이동시킨다. 데이다는 워드 단위로 이동되며, 바이트와 하프워드의 데이타 형식은 지원하지 않는다. 그러나 어드레스의 하위 2 비트가 바이트 포인터 레지스터에 저장되어 있으므로 소프트웨어에 의해서 처리할 수 있다. 16 비트 이하의 상수를 Load 할

때에는 ADDIU(ADD Immediate Unsigned) 명령어로 수행할 수 있으며, 16 비트 이상의 상수를 Load 하고자 할 때에는 하위 16 비트는 ADDIU를 사용하고, 상위 16비트는 LDH (Loa Dupper Halfword)를 사용하여 수행한다.

그리고 Load/Store 명령어에서 변위 값은 부호가 확장된 값이 Rb에 더해지며, 더해진 결과는 어드레스를 표시하는 것이므로 언제나 양수로 해석된다. 변위가 양수일 때 캐리가 발생하면 하드웨어 익셉션으로 처리된다. 음수일 때에 계산된 결과에 캐리가 발생하면 이를 무시하고 계속 수행하며, 음수인 변위의 절대 값이 Rb보다 크면, 그 결과는 정상이 아니나 익셉션으로 처리되지 않는다.

시스템이 콘텍스트 스위칭 등으로 인하여 어떤 레지스터 블럭을 이동할 필요가 있을 때, 보통 Multiple Load/Store 명령이 사용된다. HARP에서는 중첩된 메모리 액세스를 사용하지만 외부 버스가 분리되어 있어 LD나 STD(STore with Displacement) 명령어를 반복 수행할 수 있다. 따라서 LD 명령어를 여러번 수행하는 것과 Multiple Load 사이에 실행시간의 차이가 없고, 단지 프로그램의 코드 스페이스를 줄이는 것 이외에는 다른 이득이 없으므로 Load/Store 명령어의 반복으로 이를 수행한다.

범용 레지스터 사이의 데이타 이동에는 ADDU 명령어의 Rb나 Rc 대신에 R0를 사용하여 수행한다. CPU 외부와 특수 레지스터 사이에서는 상수를 제외하고는 직접적인 데이타 이동을 할 수 없다. 범용 레지스터와 특수 레지스터 사이에는 RD/WR 명령어를 사용하여 데이타 이동이 가능하다.

나. ALU 명령어

HARP에서는 Unsigned Number와 Signed Number를 처리하는 2가지로 구별되는데 Unsigned Number를 수행하는 명령어에서는 오버플로우(Overflow)를 무시하며, 캐리가 발생하면 특수용 레지스터의 캐리 레지스터에 저장된다. Signed Number를 수행하는 명령어에서는 캐리를 무시하고, 오버플로우가 발생하면 익셉션이 발생되며 익셉션 핸들러가 처리하게 된다. HARP에서는 캐리에 대한 특별한 명령어를 두지않고 특수용 레지스터를 제어하는 RD/WR 명령어를 이용하여 Set이나 Reset을 시킨다. ADD/SUB 명령어에 사용되는 상수 값은 모두 16 비트 이하의 값이며, 상위 16 비트는 Unsigned Number에서는 0이 채워지며 Signed Number에서는 부호가 확장된다.

논리계산에 사용되는 상수의 상위 하프워드는 0이 채워지며, 비트 각각이 Boolean Operation을 한다. SHD (Sift Double) 명령어는 2개의 레지스터를 이용하여 사용자 카운터 레지스터의 내용에 따라 이동시킨다. 그리고 Rb, Rc에 같은 레지스터를 사용하면 Rotate Right와 Rotate Left를 SHD 명령어로 수행할 수 있다. Arithmetic Shift는 Rc 레지스터 값에 따라 이동되는 양이 결정되며 Shift에 의해 생기는 빈자리는 부호 비트가 채워진다.

데이타 추출(Extraction)을 위해 EXTB와 EXTH 명령어를 사용한다. 축출된 값은 Ra 레지스터의 바이트 0 혹은 하프워드 0에 Write되며 나머지는 0으로 채워진다. 데이타 삽입(Insertion)을 위해 INSB, INSH 명령어가 사용되며, Rb의 내용이 Ra에 끼어들게 되므로 변화되기 전에 Ra에 있던 값은 상실된다. 바이트의 지정은 명령어의 콘디션 필드의 MSB가 1이면 그 필드의 하위 두 비트가 바이트를 지정하는 상수가 되며, MSB가 0이면 바인트 포인터 레지스터에 있는 2비트가 바이트 또는 하프워드를 지정하게 된다.

곱셈과 나눗셈은 차후에 FPU가 설계되면 지원할 수 있으며 현재는 소프트웨어에 의해서 해결한다.

다. 분기 명령어

프로그램 내에서 분기 명령어를 사용하여 어느 곳이든 분기가 가능해야 한다. BR(BRanch)명령어는 Ra와 Rb를 비교하여 조건을 만족시키면 PC에 변위를 더하여 새로운 PC 값을 만든다. 이때 변위는 부호가 확장된다. BR 명령어는 현재 PC가 가리키고 있는 위치에서 앞뒤 가까운 곳으로 분기할 때 유용하나, 그 범위가 너무 좁은 것을 감안하여 먼 곳으로 분기가 가능하도록 BRR 명령어를 따로 두었다. JMP는 PC에 상수 값을 더하여 분기하는 명령어로 현재 PC가 가리키고 있는 위치에서 멀리 분기할 때 효과적으로 쓰인다. JMPR은 어떤 세그먼트 내에서 현재의 PC와 상관없이 임의의 위치로 분기할 때 사용된다. JMPC는 HARP에서 콘디션에 의해 분기를 하는 유일한 명령어로 특수 레지스터에 있는 캐리 레지스터가 1이면 분기한다.

TRAP은 시스템 Call에 사용되는 명령어로 Ra와 Rb를 비교하여 조건을 만족시키면 상수를 Trap Number에 곱하여 새로운 PC를 만든다. HARP는 워드 단위로 어

드레싱하므로 상수는 4가 되며, 2 비트는 Shift로 곱셈을 대신한다.

라. 기타 명령어

여기서는 앞에서 기술한 명령어 그룹에서 제외된 명령어에 대하여 기술한다. 공용 변수를 액세스하고 있는 어느 프로세스 이외에는 다른 프로세서들이 공용 변수를 변경하지 못하도록 Critical Section을 만들어야 한다. TS(Test and Set)는 이처럼 상호 배제(Mutual Exclusion)를 위해 사용되는 명령어로 메모리 M(Ra+Rb)에 있는 내용을 읽어서 범용 레지스터에 저장하고 메모리 내용의 LSB를 1로 Set시킨다. 일단 TS를 시작하면 인터럽트를 받지않고 한꺼번에 처리한다. HARP에서는 하드웨어에 의해 TS 명령어를 수행한다.

익셉션을 처리할 때, 파이프라인이 파괴되지 않고 다시 시작할 수 있도록 하기 위하여 이미 수행 중인 명령어들의 어드레스는 PC 체인 중의 PC1, PC2, PC3, PC4 그리고 PC5를 사용하여 저장한다. PC 체인에 있는 명령어의 어드레스들과 특수 레지스터에 있는 내용은 PUSHSR 명령어에 의해 스택에 저장된다. 그리고 복귀될 때 스택에 저장된 3개의 어드레스는 POPSR 명령어에 의해 SR13부터 SR15에 일시 저장된다. 범용 레지스터에 있는 내용을 스택에 저장하거나 스택에서 레지스터로 가져올 때에 PUSH, POP을 사용한다. RTU 명령어는 익셉션 수행 후 원래의 프로그램이 수행될 수 있도록 SR13부터 SR15에 있는 내용을 PC에 Write 한다.

HARP에서 거의 모든 명령어들은 1사이클에 수행되며 마이크로 코드로 된 명령어는 지원하지 않는다. 따라서, 1 사이클 이상의 수행 시간을 요하는 명령어들은 어셈블리 레벨에서 1 사이클에 수행되는 명령어들의 순서로 구성된 매크로 명령어(Macro Instrution)로 지원한다.[20]

Ⅳ. 결 론

HARP는 개발 비용, 인력 및 설계능력을 감안하여 짧은 시간에 개발이 가능한 32 비트 RISC 형태의 구조를 가지고 있다.

HARP의 기본 모듈은 CPU와 2개의 HACAM 칩으로 구성된다. CPU는 현재 IPU 만으로 구성되어 있으나 차후에는 FPU를 포함하게 될 것이며, 5단계의 파이프라인으로 동작한다. 현재 39개의 IPU 명령어를 정의하였으며, Condition Code의 기능을 Compare and Branch로 대신한다. 그리고 한글의 기초 단위인 16 비트 데이타를 지원할 수 있도록 명령어를 설계하였으며, 영상 처리 등의 많은 양의 데이타를 사용하는 분야에서도 사용 가능하도록 18 비트 세그먼트 레지스터를 사용하여 48 비트 가상 어드레스를 제공토록 하였다.

현재까지 HARP의 기본 구조에 중점을 두어 명령어 세트, 어드레싱 모드, 파이프 라인, MMU의 구조 및 익셉션 처리에 관한 연구가 진행되었다. 그러나 지금까지 설계된 HARP의 구조는 시뮬레이션 과정을 거치지 않았으므로 Architecture Simulation을 통한 수정 보완 작업이 필요하다. 물 물

앞으로는 HARP의 구현에 중점을 두어 하드웨어 분야에서는 CPU와 HACAM의 마이크로 구조(Micro-Architecture)와 로직 설계 및 CPU Board을 설계할 것이다. 그리고 소프트웨어 분야에서는 시뮬레이션 환경의 구축에 중점을 두어 칩이 존재하지 않는 상태에서 HARP 명령어 세트의 수행과 시스템 소프트웨어의 개발을 가능하게 할 것이다.

〈參 考 文 獻〉

1. Motorola, Motorola MC88000 User's Manual, Motorola Texas, 1988.
2. Hewlett Packard, Precision Architecture and Instruction Reference Manual (Second Edition), June 1987.
3. Sun Microsystems, Inc., A RISC Tutorial, 1987.
4. Mark Horowitz, Paul Chow, et al., "MIPS-X : A 20-MIPS Peak, 32-bit Microprocessor with On-Chip Cache," IEEE Journal of Solid-state Circuits, Vol. SC-22, No. 5. Oct. 1987, pp. 790~799.
5. Charles Gimarc, Veljko Miutinovic, "A Survey of RISC Processors and Computers of The Mid-1980s," IEEE Computer, Sept. 1987, pp. 59~69
6. Alexander Silbey, Veljko Milutinovic, Victor Mendoza-Grado, "A Survey of Advanced Microprocessors and HLL Computer Architectures," IEEE Computer, Aug. 1986, pp. 72~85
7. Ken Sakamura, "Archtecture of The TRON VLSI CPU," IEEE Micro, April 1987, pp. 17~31.
8. 한국전자통신연구소, 64 Bit Parallel Processing System 설계기술 개발에 관한 연구(최종보고서), 1987. 5.