(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) Int. Cl.[6]
G06F 15/78

(45) 공고일자 2002년03월08일
(11) 등록번호 10-0322277
(24) 등록일자 2002년01월14일

(21) 출원번호 10-1998-0049999
(22) 출원일자 1998년11월20일
(65) 공개번호 특1999-0046225
(43) 공개일자 1999년07월05일

(73) 특허권자 주식회사 에이디칩스 권 기 홍
서울 강남구 삼성동 21-4 삼광빌딩 4층조경연
부산시 해운대구 좌동 1271번지 신성아파트 104-1402
(72) 발명자 조 경 연
부산광역시 해운대구 좌동 1271번지 신성아파트 104동 1402호
(74) 대리인 박형근, 이홍길

심사관 : 정재훈

(54) 확장 명령어를 가진 중앙처리장치

요약

본 발명은 PC(Program Counter)와, SP(User/Supervisor Stack Pointer)와, LR(Link Rerister)와, MCU의 곱셈 및 나눗셈의 연산 결과를 임시적으로 저장하는 ML/MH(Multiply Result Low/High Register)와, 어드레스 발생부 및 연산처리부와 명령어를 래치하는 명령어 레지스터로 구성된 것에 있어서, 특수 레지스터의 일종으로 프로그래머가 접근 가능한 확장 레지스터를 더 포함하고, 상기 확장 레지스터에 확장 데이터를 기억하는 동작을 나타내는 OP코드와 즉시 상수 오퍼랜드 필드로 구성된 명령어를 상기 디코더/콘트롤부에서 해석하여 즉시 상수 오퍼랜드 필드를 기억시키는 동작을 수행하며, 상기 OP코드와 OP코드가 필요로 하는 상기 오퍼랜드 필드의 일부분으로 구성되는 기본 명령어는 상기 확장 레지스터에 기억된 상기 확장 데이터가 상기 기본 명령어의 상기 일부분 오퍼랜드 필드와 연결(concatenation)되어서 완전한 오퍼랜드를 가진 명령어를 구성하여 상기 디코더/콘트롤부에서 해석되고 실행되도록 구성되어, 메모리 어드레스(address), 변위(offset), 즉시 데이터(immediate)를 가변하는 고정길이 명령어(instruction)를 가지는 중앙처리 장치에 관한 것이다.

대표도

도1

명세서

도면의 간단한 설명

<1> 도 1 은 본 발명의 실시예에 의한 중앙처리장치를 나타낸 도면.

<2> 도 2 는 중앙처리장치의 파이프라인의 동작을 설명하는 도면.

<3> 도 3 은 파이프라인의 타이밍 다이아그램으로 명령어 패치와 디코딩의 실행을 나타낸 도면.

<4> 도 4 는 상태 레지스터(SR)에 확장 플래그가 세팅된 명령어 포맷을 나타낸 도면.

<5> 도 5 는 선행기술에 의한 중앙처리장치를 나타낸 도면.

<6> *도면의 주요 부분에 대한 부호의 설명*

<7> 10...데이터 래치부 20...프리명령어 레지스터

<8> 30...프리디코더 40...명령어 레지스터

<9> 50...디코더/콘트롤부 60...연산처리부

<10> 70...곱셈기/나눗셈기 80...레지스터 파일

<11> 90...어드레스 발생부

발명의 상세한 설명

발명의 목적

발명이 속하는 기술분야 및 그 분야의 종래기술

<12> 본 발명은 메모리 어드레스, 변위(offset), 그리고 즉시 데이터(immediate date)를 가변하는 고정길이 명령어를 가지는 중앙처리 장치에 관한 것이다.

<13> 선행 기술에 의한 중앙처리장치는 도 5 에 도시된 바와 같이, 사용자가 쉽게 접근할 수 있는 영역으로 아키텍처에 맞게 구성된 GPR(General Purpose Register) 및 특수한 목적으로 사용되는 SPR(Special Purpose Register)로 구성된 레지스터 파일(7)과, 메모리로부터 패치된 명령어를 래치하는 명령어 레지스터(4)와, 상기 명령어 레지스터(4)에 래치된 명령어를 OP코드와 오퍼랜드로 디코딩하고 명령어에 따라 소정의 제어신호들을 출력하는 디코더/콘트롤부(5)와, 상기 디코더/콘트롤부(5)에서 디코딩된 명령어의 연산을 처리하는 연산처리부(6)와, 메모리에 데이터를 쓸 때나 메모리로부터 데이터를 읽어올 때 래치하고 버퍼링하는 메모리 데이터 레지스터(1)와, 프로그램 카운터에서 계산된 어드레스를 래치하여 출력하는 메모리 어드레스 레지스터(2)와, 외부에서 입력되는 콘트롤시그날을 버퍼링하는 콘트롤시그날 레지스터(3)로 이루어져 있다.

<14> 상기와 같은 구성을 가진 중앙처리장치의 명령어는 기계어(machine language)라고도 하는데 2진수 비트의 나열로 표현되고 작용을 나타내는 OP 코드와 그 작용을 받는 객체인 오퍼랜드로 구성된다.

<15> ADD 명령어를 예를 들어 OP 코드와 오퍼랜드의 예를 들어보자.

<16> 'A = B + C'는 'B'와 'C'를 더하여 'A'에 저장하라는 의미로 여기서 '+'는 작용을 나타내는 OP 코드이고 'A', 'B' 그리고 'C'는 작용을 받는 객체이므로 오퍼랜드가 된다. 이 표현식을 기계어로 예를 들어 표현하면 '0001 0000 0001 0010'으로 나타낼 수 있는데 순서대로 '0001'은 OP 코드 즉, '+'를 기호화 한 것이고, '0000', '0001' 그리고 '0010'은 각각 오퍼랜드인 A,B 그리고 C를 기호화 한 것이다. 이러한 2진수의 표현은 길고 읽기가 어려워 보다 간단한 16진수로 표현하기도 한다. 그래서 위와 같은 예의 16진수 표현은 '0×1012'이 된다. 기계어에서 오퍼랜드는 레지스터, 메모리 어드레스, 변위, 그리고 즉시 데이터 등이 있다.

<17> 레지스터는 그 수가 한정이 되어 있어 32개 이하인 경우가 많다. 예를 들어 레지스터의 수가 16개라면 4비트의 오퍼랜드로 이를 표현할 수 있다(2**4=16). 메모리 어드레스인 경우 32 비트 중앙처리장치는 4G바이트의 메모리를 사용할 수 있는데 이를 표현하기 위해서는 32비트의 어드레스가 필요하게 된다. 따라서 이를 정의하게 되는 오퍼랜드의 길이가 길어지게 된다. 그리고 변위와 즉시 데이터의 경우도 메모리의 경우와 유사하게 오퍼랜드의 길이가 길어지게 된다. 오퍼랜드의 길이가 길어지면 기계어의 길이가 길어지고 기계어의 길이가 길어지면 프로그램의 크기가 증가하게 되어 비효율적이 된다.

<18> 이러한 이유로 각각의 중앙처리장치는 오퍼랜드를 효율적으로 표현하는 기법을 가지게 된다.

<19> IBM-PC에서 사용하는 80386은 여러 바이트 길이 명령어(Multi byte length instruction)를 가진다. 예를 들면 80386의 'MOVE' 명령 기계어는 오퍼랜드 길이에 따라 다음과 같이 정의된다.

<20> MOV AL, 12 → B012

<21> MOV AX, 1234 → B8 34 12

<22> MOV EAX, 12345678 → 66B8 78 56 34 12

<23> 또한, MC6800도 80386과 유사하게 여러 16비트 길이 명령어(Multi 16bit length instruction)를 가진다.

<24> 이와 같이 가변 길이 명령어(Variable length instruction)는 어떠한 길이의 오퍼랜드도 나타낼 수 있는 장점을 가지고 있으나 기계어의 길이가 변화하므로 명령어 디코더, 예외처리 등이 어려워지는 단점을 가지게 된다. 이러한 가변 길이 명령어를 가지는 중앙처리 장치를 CISC(Complex Instruction Set Computer)라고 칭한다.

<25> 한편, RISC(Reduced Instruction Set Computer)에서는 기계어의 길이가 고정되어 있다.

<26> 이것의 예로 MIPS-R3000, SPARC, ARM-7 등은 32비트 고정 길이 명령어(32 bit fixed length instruction)를 가지고, Hitachi의 SH-3은 16비트 고정길이 명령어를 가진다. 이러한 고정길이 명령어는 모두 기계어의 길이가 일정하므로 명령어디코드, 예외처리 등이 손 쉬어 지므로 파이프라인을 적용하기 쉬워 간단한 하드웨어로 고성능 중앙처리장치의 구현이 가능한 반면에 명령어의 길이가 고정이 되어 오퍼랜드 길이에 제약이 따른다.

<27> 예를 들어 MIPS-R3000에서는 32비트 용량의 메모리를 가지나 기계어에서 표현할 수 있는 변위는 16비트이다. 또한 32비트 중앙처리 장치이면서도 즉시 상수의 길이도 16비트로 한정된다. 이에 따라서 프로그램 작성이 어렵고 따라서 성능을 저하시키는 요인이 되고 있다.

<28> 또한 'MOVE' 명령은 레지스터의 내용을 다른 레지스터로 복사하는 내용인데, MIPS-R3000은 32개의 레지스터를 가지므로 레지스터 오퍼랜드는 5비트 길이가 되며, 'MOVE'를 나타내는 OP코드를 6비트로 정의하면 16비트 길이 명령어로 정의할 수 있다. 그러나 고정길이 명령어를 사용하기 위해서 이와 같이 16비트로 표현 가능한 명령어를 32비트로 표현하고 있다. 따라서, 32비트 고정 길이 명령어는 오퍼랜드 길이가 제한되는 단점을 가지면서 또한 불필요하게 긴 명령어를 가지게 되는 단점이 있다.

<29> 또 다른 예로 TR-4101을 보자.

<30> TR-4101은 16비트 고정 길이 명령어를 가지며 이러한 고정길이 명령어와 오퍼랜드를 일부 확장하는 기능을 가진다. 예를 들면, 메모리에서 데이터를 읽어오는 'LOAD'명령어는 'LOAD'를 나타내는 OP코드와, 읽어와서 저장되는 레지스터를 나타내는 목적 레지스터, 오퍼랜드와 메모리의 위치를 나타내는 인덱스 레지스터, 오퍼랜드와 인덱스로부터의 변위를 나타내는 변위 오퍼랜드로 구성된다. 이들 OP 코드와 여러 종류의 오퍼랜드를 16비트 길이의 명령어에 표현하기 위해서 TR-4101에서는 변위를 5비트로 제한하였다. 그러나 5비트 변위로는 메모리의 위치를 지정하기에는 충분하지 않다. 그래서 TR-4101에서는

'EXTEND' 명령어를 사용한다.

<31> 상기 'EXTEND' 명령어는 5비트의 OP코드와 11비트의 즉시 상수 오퍼랜드로 구성된다. 여기서 11비트의 즉시 상수 오퍼랜드는 'EXTEND' 명령어 다음에 위치하는 명령어에 따라서 다르게 해석된다. 예를 들어 'EXTEND' 명령어 다음에 'LOAD'가 나타나면 'EXTEND' 명령어의 11비트 즉시 상수 오퍼랜드와 'LOAD' 명령어의 5비트 변위가 연계(concatenation)되어서 16비트 변위를 나타낸다.

<32> 이러한 TR-4101의 명령어 확장 기술은 변위와 즉시 상수를 16비트로 확장하는데 그치므로 종래 RISC 중앙처리 장치가 가지고 있었던 오퍼랜드 길이의 제약은 해결하지 못하고 있다.

<33> 또한 오퍼랜드 확장이 가능한 명령어는 선행하는 'EXTEND' 명령어의 유무에 따라서 오퍼랜드 지정이 달라지게 된다. 따라서 'EXTEND' 명령어가 연계되는 명령어는 하나의 명령어로 취급되어 진다. 즉, 'EXTEND'명령어 다음에는 예외 처리가 수행될 수 없다는 단점을 가지고 있어서 주변장치의 응답요구를 실시간으로 처리할 수 없는 문제점이 있다..

발명이 이루고자 하는 기술적 과제

<34> 따라서, 본 발명은 상기와 같은 제반 결점을 해소하기 위하여 창출한 것으로서, 본 발명의 목적은 명령어의 오퍼랜드의 길이에 제약을 받지 않아 모든 길이의 메모리 어드레스, 변위 및 즉시 데이터를 표현하고, 고정 길이 명령어를 사용하여 명령어 디코더 회로를 간단히 함과 동시에 예외 처리가 손쉬워져 파이프라인 및 메모리 관리 장치(Memory Management Unit : MMU)가 간단한 확장 명령어를 가진 중앙처리장치를 제공하는데 있다.

<35> 또한, 본 발명의 다른 목적은 확장 명령어 다음에 예외 상황이 발생되어도 즉시 예외처리 프로시져를 수행한 후 처리가 중단된 확장 명령어의 다음 루틴으로 복귀할 수 있는 확장 명령어를 가진 중앙처리장치를 제공하는데 있다.

<36> 즉, 본 발명은 CISC와 RISC의 장점을 취해, 모든 길이의 메모리 어드레스, 변위 및 즉시 데이터를 표현할 수 있는 고정 길이 명령어를 가지는 중앙 처리 장치로 구성되는 확장 명령어 셋트 컴퓨터(Extendable Instruction Set Computer : EISC)에 관한 것이다.

발명의 구성 및 작용

<37> 상기의 목적을 달성하기 위하여 본 발명에 따른 확장 명령어를 가진 중앙처리장치는, 접근 속도가 빠른 소규모 기억 장치인 레지스터의 집합인 레지스터 파일과 상기 레지스터 파일과 접속되어 정보를 송수신하는 통로인 내부버스와 상기 내부버스에 연결되어 있으며 외부 버스를 연결하는 외부 버스 버퍼와 상기 내부버스에 연결되어 있으며 연산 기능을 수행하는 기능블록과 상기 내부버스에 연결되어 있으며 수행중인 명령어를 기억하는 명령 레지스터와 상기 명령 레지스터에 연결되어 있어서 명령어를 해석하여 상기 레지스터 파일과 상기 내부 버스와 상기 외부버스 버퍼와 상기 기능 블록과 상기 명령 레지스터에 제어 신호를 발생하는 디코더/콘트롤부로 구성되며, 상기 레지스터 파일은 프로그래머가 접근 가능하고 연산 원시 데이터나 연산 결과 데이터를 저장하거나 상기 연산 데이터를 기억하고 있는 메모리의 번지 등을 기억하는 하나 또는 다수개의 범용 레지스터와 프로그래머가 접근 가능하고 중앙처리장치 동작에 필요한 정보를 기억하는 하나 또는 다수개의 특수 레지스터와 프로그래머가 접근할 수 없으며 중앙처리장치 동작에 필요한 특수한 기능이나 연산의 중간 과정 등을 기억하는 하나 또는 다수개의 내부 레지스터로 구성되어 있고, 상기 특수레지스터의 일종으로 진행하고 있는 프로그램이 저장되어 있는 메모리 번지를 기억하는 프로그램 카운터를 가지며, 상기 프로그램 카운터가 지정하는 프로그램 번지가 상기 내부 버스에 출력되고, 상기 내부 버스에 출력된 프로그램 번지가 상기 외부 버스 버퍼를 통하여 프로그램을 기억하고 있는 외부 메모리의 번지로 출력되고, 상기 방법으로 지정된 외부 메모리로부터 명령어가 읽혀져서 상기 외부 버스 버퍼를 통하여 상기 내부 버스에 연결되고, 상기 명령어가 상기 명령 레지스터에 저장되어 상기 디코더/콘트롤부에서 상기 제어신호를 발생하며, 상기 명령 레지스터에 저장되어 상기 디코더/콘트롤부에서 해석되는 상기 명령어는 동작을 지정하는 OP코드 필드만으로 구성되거나 상기 OP코드 필드와 동작의 작용을 받는 하나 또는 복수개의 오퍼랜드 필드로 구성되고, 상기 OP코드 필드와 상기 오퍼랜드 필드는 하나 또는 복수개의 2진수 비트로 구성되고, 상기 오퍼랜드 필드는 상기 명령어의 종류에 따라서 메모리 번지를 나타내는 번지 오퍼랜드나 상기 범용 레지스터나 상기 특수 레지스터에 기억된 메모리 번지로부터의 변위를 나타내는 변위 오퍼랜드나 연산이나 메모리 번지나 제어에 사용되기 위한 즉시 상수를 나타내는 즉시 상수 오퍼랜드 등으로 해석하고 실행하는 중앙처리장치에서,

<38> 상기 특수 레지스터의 일종으로 프로그래머가 접근 가능한 확장 레지스터를 더 포함하고, 상기 확장 레지스터에 확장 데이터를 기억하는 동작을 나타내는 OP코드와 즉시 상수 오퍼랜드 필드로 구성된 명령어를 상기 디코더/콘트롤부에서 해석하여 즉시 상수 오퍼랜드 필드를 기억시키는 동작을 수행하며, 상기 OP코드와 OP코드가 필요로 하는 상기 오퍼랜드 필드의 일부분으로 구성되는 기본 명령어는 상기 확장 레지스터에 기억된 상기 확장 데이터가 상기 기본 명령어의 상기 일부분 오퍼랜드 필드와 연결(concatenation)되어서 완전한 오퍼랜드를 가진 명령어를 구성하여 상기 디코더/콘트롤부에서 해석되고 실행되는 것을 특징으로 한다.

<39> 상기와 같은 특징을 가진 본 발명은 종래 기술과 마찬가지로 범용 레지스터(General Purpose Register : GPR)와 특수 목적 레지스터(Special Purpose Register : SPR)를 가지고 있으며, 이에 더하여 확장 레지스터(Extension Register : ER)를 가진다. 또한 본 발명의 EISC는 종래 기술과 동일하게 중앙처리장치의 상태를 나타내는 상태 플래그들(Status Flags)과 이들 상태 플래그의 집합인 상태 레지스터(Status Register)를 가지며, 이에 더하여 명령어 확장 상태를 나타내는 확장 플래그(Extension Flag)를 상태 레지스터 내에 가진다. ER의 길이는 중앙 처리 장치의 워드 길이에 따라 변경되는데, 16비트 중앙처리 장치에서는 16비트 이하, 32비트 중앙 처리 장치에서는 32비트 이하, 64비트 중앙 처리 장치에서는 64비트 이하의 길이를 가진다. 본 발명의 실시예에서는 32비트 중앙 처리 장치에서 32비트 길이의 ER을 예로 들어서 설명하지만, 이것은 ER의 길이에 제한을 두거나 중앙 처리 장치의 워드 길이에 제한을 두기

위한 것이 아니며 설명을 용이하게 하기 위한 것임을 밝힌다. EF(Extension Flag)는 1비트 이상의 길이로 프로그래머(programmer)가 접근 가능한 형태이어야 한다. 중앙 처리 장치가 가지는 일부 플래그와 레지스터는 프로그래머가 접근할 수 없는 것이 있다.

<40> 본 발명의 실시예에서 EF는 상태 레지스터의 한 비트로 설명하고 있는데, 이 또한 설명의 용이성을 위한 것으로 EF의 구현 형태를 제한하고자 하는 목적이 아님을 밝힌다.

<41> 본 발명에 따른 중앙퍼리장치는 ER에 데이터를 저장하면서 동시에 EF를 '1'로 설정하는 명령어를 가지는데 본 발명의 기술에서는 설명의 편의성을 위하여 'LDERI(Load ER Immediate)'라는 명령어를 사용한다. 또한 32비트 중앙 처리 장치에서 16비트 고정 길이 명령어를 사용하고 'LDERI' 명령어의 오퍼랜드를 14비트 즉시 상수라고 정의하면

<42> LDERI #123

<43> 라는 명령문은 이 명령문 수행이전에 EF가 '1' 이었다면 ER을 왼쪽으로 14비트 길이만큼 산술이동(Arithmetic shift) 시키고, 이 명령문의 오퍼랜드 부분인 '123'을 ER에 더하는 작용을 한다. 또한 이 명령문 수행 이전에 EF가 '0' 이었다면 이 명령문의 오퍼랜드 부분인 '123'의 기호를 확장(Sign extend)하여 32비트를 만들어서 ER에 저장한다. 그리고 두 경우 모두에서 EF는 '1'로 설정된다.

<44> 앞의 기술에서 'LDERI' 명령어의 오퍼랜드 길이는 다른 명령어의 정의에 따라서 결정되는 것으로 본 발명에서는 그 길이에 제한을 두지는 않는다. 또한 'LDERI' 명령어에서 오퍼랜드로 즉시 상수를 사용하는 예를 보였는데, 오퍼랜드 종류는 메모리 어드레스 등의 다른 종류의 오퍼랜드를 사용할 수도 있음을 밝힌다. 즉 예를 들어서 PC 상대 어드레싱(PC relative addressing)을 가질 수도 있으며 본 발명에서는 ER에 데이터를 격납(load)하는 하나 또는 다수 개의 명령어 수단을 가지는 것을 의미한다.

<45> 본 발명의 EISC는 종래 기술이 가지는 일반적인 명령어를 가지고 있는데, 이중 가변하는 오퍼랜드를 필요로 하는 명령어는 EF 상태에 따라서 오퍼랜드 해석에 차이를 가진다.

<46> 예를 들어서 16비트 고정 길이 명령어를 가지는 32비트 중앙 처리 장치에서 8비트 변위를 가지는 'JMP' 명령어는 다음과 같이 표현된다.

<47> JMP offset

<48> 'JMP' 명령어는 프로그램의 순서를 바꾸는 명령어로 이 명령문 다음에는 현재 프로그램의 위치에서 변위('offset') 만큼 떨어진 위치의 명령문을 수행하게 된다. 그런데 변위의 길이가 8비트로 할당되어 있으므로 점프되는 위치는 -128 바이트에서 +127 바이트 이내로 국한된다.

<49> 본 발명의 EISC에서는 이 경우에 EF의 상태에 따라서 변위는 2가지로 해석된다. 먼저 'EF'가 '0'인 경우에는 변위의 길이를 8비트로 해석하며, EF가 '1'인 경우에는 ER의 값을 왼쪽으로 8비트 산술이동(Arithmetic shift)시킨 다음에 명령문에 나타난 8비트 변위를 더하여 변위를 산출한다. 따라서 32비트 변위를 가지도록 명령문의 오퍼랜드가 확장된다.

<50> 명령어에 따라서 ER을 왼쪽으로 이동시키는 거리는 다르다. 앞의 예에서도 'LDERI' 명령어는 14비트, 'JMP' 명령어는 8비트를 이동시키고 있다.

<51> EF플래그는 'LDERI' 명령어에 의하여 '1'이 되고, 'LDERI' 명령을 제외하고 ER 레지스터를 참조하는 모든 명령어에서 '0'이 된다. 그러나 구현 방식에 따라서 다른 값을 가질 수도 있으며, EF 플래그는 명령어의 확장 상태를 나타내는 다른 구현 형태를 가질 수 있다.

<52> 또 다른 예로써 16비트 고정 길이 명령어를 가지는 64비트 중앙 처리 장치에서 'LDERI'의 오퍼랜드 길이가 12비트 일 때

<53> LDERI #opr1 : 명령어 -1

<54> LDERI #opr2 : 명령어 -2

<55> LDERI #opr3 : 명령어 -3

<56> LDERI #opr4 : 명령어 -4

<57> JMP offset : 명령어 -5

<58> 라는 프로그램의 동작을 살펴보자.

<59> '명령어-1' 수행이전에 EF가 '0'이라고 가정하면, '명령어-1'에 의하여 ER은 'opr1'의 기호를 64비트로 확장한 값을 가지며, EF는 '1'이 된다. 다음 '명령어-2'에서 EF가 '1'이므로 ER은 'opr1'을 왼쪽으로 12비트 산술 이동시키고 'opr2'를 더한 값을 가진다. 이때 예외(exception)가 발생되면 '명령어-3'을 수행하지 않고 예외 처리를 먼저 수행해야 한다.

<60> 본 발명의 EISC에서는 ER과 EF는 프로그래머가 접근할 수 있는 레지스터 및 프로그램이므로 이들을 스택에 저장하거나 메모리내의 버퍼에 저장하는 등의 방법으로 그 내용을 보존할 수 있으며, 예외 처리를 마친 후에 저장된 ER과 EF를 재격납(reload)하면 '명령어-3' 이하를 올바르게 수행할 수 있다. 또한 예외 처리 중에 또 다른 예외가 발생하더라도 ER과 EF를 순서적으로 저장, 재격납 함으로써 프로그램의 올바른 수행을 보장할 수 있다. 이 프로그램에서 '명령어-5'의 offset 길이는 '명령어-1'부터 '명령어-4'까지의 오퍼랜드가 더해져서 48비트가 되며, 여기에 '명령어-5'가 가지는 8비트 offset을 더하면 56비트 길이가 된다.

<61> 이와 같이 본 발명의 EISC는 모든 명령어가 고정 길이이고, 모든 명령어가 독립적으로 수행되며, 가변 길이의 오퍼랜드를 가질 수 있다.

<62> 결론적으로 본 발명은 확장 레지스터 ER과 확장 레지스터의 상태를 나타내는 수단과 ER에 값을

저장하는 명령어와 확장 레지스터의 상태에 따라서 오퍼랜드 해석을 달리하는 명령어를 가지는 중앙 처리 장치에 대한 발명이다. 본 발명으로 고정 길이 명령어를 가지면서 메모리 어드레스, 변위, 즉시 상수의 길이가 가변되는 중앙 처리 장치의 구현이 가능하게 되었다.

<63> 이하, 예시된 도면을 참조하여 본 발명을 더욱 상세히 설명하면 다음과 같다.

<64> 본 발명의 내용을 상세히 설명하기 위해 32비트의 구조를 가진 MCU를 예로 들어 설명한다. 이는 데이터 경로, 버스 폭의 확장 및 축소가 동일한 구조로 가능하며, 또한 각각의 세부적인 블록들은 각기 다양한 방법으로 정의 될 수 있고, 본 실시예에서는 설명의 용이성을 위하여 단지 32비트의 MCU를 사용한다.

<65> 도 1 은 32비트 MCU의 블록도이다. 동 도면에서 MCU의 구조는 32비트의 버스와 연산처리부(ALSU Unit)(60)를 가지고 있으며, 명령어는 16비트로 구성되어 있고 3단계 파이프라인을 사용한다. 또한 메모리 구성은 8비트 데이터 폭을 가진 메모리 4개를 병렬로 연결하는 것으로 한다.

<66> 먼저, MCU의 기본 동작을 설명하면 다음과 같다.

<67> 처음에 리세트가 걸리면, 메모리로부터 시작 주소를 읽어서 데이터 래치부(10)에 저장되고 이 값이 레지스터 파일(80)내의 PC(프로그램 카운터)에 저장되며, 다음 사이클부터 프로그램이 순차적으로 진행하는 한 이 값은 메모리 구성에 따라 증가된다. 이 값을 어드레스 발생부(90)에서 참조하여 사용자가 의도하는 프로그램을 메모리로부터 읽어온다. 그래서 실제로 명령어들이 수행이 시작되려면 프리 명령어 레지스터(PIR:Pre-Instruction Register)(20)에 파이프라인을 위한 명령어가 패치 되고, 이것을 프리디코더(30)에서 대부류로 구분한다. 여기서 대부류라 함은 정의된 명령어를 파이프라인 수나 실행하는 방법 등을 토대로 비슷한 유형 몇 가지로 구분하여서 어떤 상태천이를 할 것인지를 결정하는 것이다. 그 후 명령어는 명령어 레지스터(IR-Instruction Register)(40)에 래치되고, 이 값과 프리디코더(30)의 출력을 참조하여 디코더/콘트롤부(50)에서 해당 명령어의 상태천이를 결정하고 각 상태에 따른 제어신호를 발생시킨다. 이 제어신호를 통해 내부적으로는 32비트 연산처리부(ALSU)(60)와 곱셈기/나눗셈기(70)를 동작시켜 결과를 레지스터 파일(80)에 저장시킨다. 레지스터 파일(80)의 상세도는 도 4에 도시된 바와 같이 크게는 GPR과 SPR로 나눌 수 있다. GPR은 사용자가 쉽게 접근할 수 있는 영역이며, 16개의 32비트 레지스터로 구성되어 있다.

<68> 반면에 SPR은 특수한 목적으로 사용되는 레지스터들로 구성되어 있으며, 그 각각을 설명하면 다음과 같다.

<69> PC(Program Counter): 프로그램의 순차적인 흐름을 유지하기 위해 사용된다.

<70> SP(User/Supervisor Stack Pointer):예외(Exceptions) 발생시나 비순차적인 프로그램의 흐름에 의해 분기가 필요할 때 등의 경우에 그 예외나 분기에 해당하는 동작을 수행하고 난 후, 현재 진행중인 프로그램의 흐름을 유지하기 위해 필요한 내용(PC, SR, etc)을 저장해야 하고, 이때 그 주소를 타나 내기 위해 사용된다.

<71> LR(Link Register): 비순차적인 프로그램의 흐름에 의해 분기가 발생할 때 위에서 설명한 것처럼 프로그램 카운터의 값을 메모리(Stack Area)에 저장하는데 프로그램의 특성상 말단 함수의 경우에는 바로 주소를 되돌려 받고, 다시 말단 함수를 호출하고 하는 동작을 반복하기 쉽다. 이러한 것을 고려하여 말단 함수의 호출일 경우에 그 주소를 메모리에 저장하지 않고 임시적으로 저장하기 위하여 이 Link Register를 사용한다. 이를 통하여 함수의 동작을 끝내고 그 주소 값을 돌려 받을 때 메모리를 읽지 않아도 되므로 성능을 향상시킬 수 있다.

<72> ML/MH(Multiply Result Low/High Register): 여기에서 예로 사용한 MCU에는 곱셈기와 나눗셈기가 존재한다. 그래서 이들의 연산 결과를 임시적으로 저장하기 위해 이 두 개의 레지스터를 사용한다.

<73> ER(Extension Register): 앞에서 설명한 것처럼 본 발명에서 의도하는 변위(Offset)이나 즉시(Immediate) 값을 임시적으로 저장하여 의도하는 양을 만들기 위해 사용한다.

<74> SR(status Register): 일반적인 모드 MCU에서처럼 연산하는 과정의 여러 가지 상태 값을 저장하기 위해 사용한다.

<75> 각각의 비트는 그림에서 보여지는 것과 같은 상태를 나타내고 주목할 것은 19번째 비트가 확장플래그(Extension Flag)로서 다음에 이어지는 명령어가 변위 또는 즉시 값으로서 짧은 값을 취할지, 확장된 값을 취할지를 결정해 준다는 것이다. 물론, 이 예에서는 상태 레지스터내의 한 비트 즉, 19번째 비트를 플래그로 사용하였지만, 이는 하나의 예일 뿐 실제로는 다양한 방법을 통해 하드웨어적으로 구현이 가능하다.

<76> 그리고 어드레스 발생부(90)에서는 여러 가지의 주소 요소들 중에 다음 명령어 및 데이터의 메모리 주소에 해당하는 요소를 선택하여 프로그램의 순차적, 또는 비순차적인 진행을 위해 사용되고, 그 주소에서 명령어를 읽어와 프리명령어 레지스터(20)에 래치시켜 프로그램을 순차적으로 진행시키거나, 또한 필요한 데이터를 메모리로부터 가져와 데이터 래치부(10)에 래치시켜 적절한 동작을 수행하도록 한다.

<77> 도 2 는 본 실시예에서 사용하는 MCU의 파이프라인이 어떻게 동작하는지를 나타내고 있다. 앞에서 언급되었듯이 패치 → 디코드 → 실행이라는 3단 파이프라인을 사용한다. 그림에서 Simple이란 명령어를 패치해서 디코드하고 이를 실행하는데 각각 1사이클씩 소요되는 명령어를 의미하고, LD/ST는 메모리로부터 데이터를 읽거나 쓰는 명령어들로서 이러한 명령어들은 유효 주소를 구하기 위해 부가적으로 1Stage가 더 소요된다. 또한 본 장치는 곱셈, 나눗셈 그리고 배럴 시프트(Barrel Shifter) 없이 다중 시프트(Multiple Shift) 명령어를 지원하는데 다중 (Multiple)이란 이러한 단일 사이클에 실행을 마칠 수 없는 명령어들로서 이들을 위한 파이프라인 모식도를 나타내는 것이다. 도 3 은 파이프라인에 대한 타이밍도로서 명령어가 어떻게 패치되어 디코드되고 실행하는지를 간략하게 나타내었다.

<78> 이상에서 본 실시예의 MCU의 일반적인 동작의 흐름을 살펴 보았다. 이제 구체적으로 Extension

명령어의 동작을 설명한다.

<79> Extension 명령어가 아닌 일반적인 명령어가 앞에서의 설명처럼 수행되다가 Extension 명령어가 들어오면 이에 의해 MCU는 다음의 두가지의 동작을 수행한다. 먼저, 상태 레지스터의 19번째 비트를 '1'(부 논리의 경우는 '0')로 변화시킨다. 19번째 레지스터가 1로 세트되면 이어지는 명령어는 이 플래그를 확인하여 메모리 읽거나 쓰는 동작이면 주소의 변위 값으로서, 또한 내부적인 ALSU 동작이면 즉시 값으로서 ER(Extension Register)이 참조되도록 안내하는 역할을 한다. 그리고 두 번째는 의도하는 변위 또는 즉시 값을 ER에 가져오는 것인데, 이는 또한 두가지로 나눌 수 있는데 첫째는 처음으로 Extension 명령어를 사용할 경우나, 또는 앞에 Extension 명령어가 나왔더라도 이미 ER 값을 사용한 경우이고, 둘째는 앞에서 이미 Extension 명령어가 수행되었으나 이 ER의 값이 사용되지 않았을 경우이다.

<80> 이것을 설명하기 위해 Extension 명령어를 다음의 표1과 같이 정의하자.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR[15:0]	&	&	*	*	*	*	*	*	*	*	*	*	*	*	*	*

<82> & : IR[15 : 14], Op-코드

<83> * : IR [13 : 0], 즉시 데이터

<84> 표1 확장 명령어의 한 예

<85> 첫째의 경우는 표1에서 표시된 즉시 14비트를 ER레지스터의 13~0 비트에 채우고, 그 상위 비트는 부호확장으로서 즉시 최상위 비트로서 채운다. 두 번째의 경우는 이미 앞에서 14비트를 ER에 채워 놓은 상태이므로 이 값을 보존하기 위해 14비트를 왼쪽으로 이동시킨 후에 즉시 14비트를 ER의 13~0 비트에 채운다. 이를 통해 확장 명령어의 반복 수행으로 무한대로 변위와 즉시 값을 확장시킬 수 있다.

<86> 다음은 확장 명령어에 이어서 나올 수 있는 명령어들의 동작을 예를 들어 설명한다.

<87> 첫째, 메모리에서 데이터를 읽어오거나, 쓰는 동작을 수행하는 데이터 이동 명령어들에서 이 명령어가 주소의 변위로서 어떻게 동작하는지 살펴보자. 이에 대한 설명의 편의를 위해 다음의 표2와 같은 명령어를 정의한다.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR[15:0]	&	&	&	&	#	#	#	#	*	*	*	*	%	%	%	%

<89> & : IR[15 : 12], Op-코드(Load/Store)

<90> # : IR[11 : 8], 소스/목적 레지스터

<91> * : IR [7 : 4], 변위 데이터

<92> % : IR [3:0], 인덱스 레지스터

<93> 표2 32비트 Load/Store 명령어의 일예

<94> 이 명령어를 실행하고자 할 때 디코더/콘트롤부(50)에서 상태 레지스터의 19번째 비트인 확장 플래그를 참조한다. 그래서 만약, 그 값이 '0'(부 논리의 경우에는'1')이라면 유효 주소는 인덱스 레지스터의 값에 변위를 더한 값이 된다. 단,여기서 또 한가지 고려해야 할 사항은 메모리의 데이터 폭이라 할 수 있다. 즉 사용자의 편의에 따라 8비트의 메모리를 병렬로 4개를 연결할 수도 있고, 또는 16비트의 메모리 2개를 병렬로 연결할 수도 있는 등 다양한 구성이 가능하기 때문이다. 전술한 바와 같이 본 실시예에서는 8비트의 메모리 4개를 병렬로 연결하는 것으로 가정한다. 이러한 경우 변위를 더할 때 변위 값을 2비트 왼쪽으로 이동시키는 것이 필요하다. 왜냐하면 현재 수행하는 명령어가 32비트 Load/Store이기 때문에 하위 두 비트는 의미가 없어지기 때문이다. 그러나 플래그 값이 '1'(부 논리의 경우에는 '0')이라면 이는 바로 전에 확장 명령어가 수행되어 ER에 확장할 변위 값을 옮겨 놓았음을 의미하고, 그러므로 유효 주소는 ER의 값을 4비트만큼 왼쪽으로 이동시키고, 명령어에 포함된 변위 4비트 중 하위 두 비트를 2비트 왼쪽으로 이동시키고 여기에 인덱스 레지스터 값을 더한 것이 유효 주소가 된다. 물론, 여기에서 ER을 왼쪽으로 이동시키는 양은 정의에 따라 다양할 수 있다. 여기서는 명령어들의 일관성을 위해 4비트만을 이동시키는 것으로 정의한다. 이 예에서 32비트 Load/Store 명령어로서는 인덱스 레지스터 +63까지의 주소를 엑세스할 수 있다. 그러나 변위가 63을 넘어서면 먼저 확장 명령어를 사용하여 15를 초과하는 값을 ER에 옮겨 놓고 32비트 Load/Store 명령어를 수행하여 원하는 주소를 엑세스할 수 있다는 것이다. 이것을 간단한 수식으로 나타내면 다음과 같다.

When E-Flag is '0',

$$EA <= (Zero\ extend\ IR[7:4]\ \&\ "00") + R_{index}$$

When E-Flag is '1',

$$EA <= (ER<<4\ \&\ IR[5:4]\ \&\ "00") + R_{index}$$

<96> 둘째, 데이터 이동 명령어 중에서 즉시 값을 바로 로드하는 경우를 예로 들어 보자. 이를 위해 다음 표3과 같은 명령어를 정의한다.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR[15:0]	&	&	&	&	#	#	#	#	*	*	*	*	*	*	*	*

<97>

<98> & : IR[15 : 12], Op-코드(로드 즉시 데이터)

<99> # : IR[11 : 8], 목적 레지스터

<100> * : IR [7 : 0], 부호화된 즉시 데이터

<101> 표3 로드 즉시 데이터 명령어의 일예

<102> 표3에서 알 수 있듯이 위의 로드 즉시 명령어를 통해서는 부호를 가진 7비트 즉시 값 즉, -256 ~ 255까지를 목적 레지스터에 로드할 수 있다. 그러나, 필요한 즉시 값이 -256 ~ 255의 범위를 벗어나는 경우에는 역시 확장 명령어를 먼저 수행하여 원하는 즉시 값을 ER 레지스터에 옮겨놓고 이 로드 즉시 명령어를 수행하여 원하는 즉시 값을 얻을 수 있다. 이 경우도 역시 로드 즉시 명령어가 명령어 레지스터(40)에 래치되고 이를 실행하기 위해 디코더/콘트롤부(50)에서 제어신호를 발생시킬 때 상태 레지스터의 19번째 비트를 참조하여 '0'(부 논리일 때는 '1')이면 명령어 내의 7비트 즉 -256 ~ 255 까지의 즉시 값을 직접 목적 레지스터에 로드하고, 만약 '1'(부 논리일 때는 '0')일 때는 ER 레지스터의 값을 4비트 왼쪽으로 이동시키고, 여기에 명령어내의 하위 4비트를 채워서 원하는 크기의 즉시 값을 로드할 수가 있다. 물론, 여기에서 ER 레지스터의 값을 이동시키는 양이나, 채우는 즉시 값은 명령어의 정의에 따라 다양하게 변화 시킬 수 있다. 만약, 여기에서 정의한 명령어로 32비트의 즉시 값을 원한다면, 확장 명령어를 두 번 수행하여 상위 24비트를 ER에 옮겨 놓은 후에 로드 즉시 명령어를 수행하여 32비트의 즉시 값을 얻을 수 있다. 이를 간단한 수식으로 나타내면 다음과 같다.

When E-Flag is '0',

$$R_{dst} \leftarrow \text{Sign extend } IR[7:0]$$

When E-Flag is '1',

$$R_{dst} \leftarrow ER<<4 \ \& \ IR[3:0]$$

<104> 셋째, ALU 명령어의 경우를 예로 들어보자. 이를 위해 다음의 표4와 같은 명령어를 정의한다.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR[15:0]	&	&	&	&	&	&	&	&	#	#	#	#	*	*	*	*

<106> & : IR[15 : 8], Op-코드(ALU 명령)

<107> # : IR[7 : 4], 목적 레지스터

<108> * : IR[3 : 0], if E='0' 인 경우에는 소스 레지스터,이외는 즉시 레 지스터

<109> 표4 ALU 명령어의 일예

<110> 일반적으로 ALU 명령어의 오퍼랜드는, 다음의 몇가지 유형으로 구분할 수 있겠다.

<111> 1.2개의 레지스터간의 연산.

<112> 2.1개의 레지스터와 Immediate 값과의 연산.

<113> 3.1개의 레지스터와 메모리 내용과의 연산 등

<114> 그러나 일반적인 RISC 구조에서는 Load/Store를 제외한 명령어는 메모리를 엑세스하지 않는다. 그래서 여기에서 예로 드는 구조도 그러한 RISC의 특징을 따라 ALU연산은 2가지 유형으로 구분하고, 이를 위 표4에서 보여지는 하나의 명령어로 나타내었다. 이 ALU명령어가 명령어레지스터(40)에 래치되어 실행을 위해 디코더/콘트롤부(50)에서 제어신호를 발생시킬 때도 또한 상태 레지스터의 E플래그를 참조한다. 만약, E플래그가 '0'(정 논리일 경우)이라면 이 명령어는 2개의 레지스터간 연산이 되고, '1'이라면 1개의 레지스터와 즉시 값과의 연산이 된다. 첫 번째의 경우는 명령어의 3번째 비트부터 최하위 비트까지의 4비트를 통해 16개의 레지스터중 하나를 선택하여 목적 레지스터(IR[7:4])와 연산을 수행하고 그 결과값은 목적 레지스터에 기록되게 된다. 반면에, 두 번째의 경우는 목적 레지스터와 ER 레지스터를 참조하여 즉시 값과 더한다. 즉 즉시 값과 연산을 수행하고자 할 때는 먼저 확장 명령어를 수행하여 원하는 양을 ER에 옮기고 난 후에 ALU명령어를 수행한다는 것이다. 이 예를 간단한 수식으로 나타내면 다음과 같다.

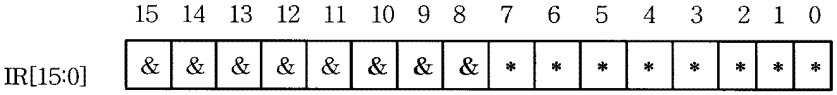
When E-Flag is '0',

$$R_{dst} \leftarrow R_{src} + R_{dst}$$

When E-Flag is '1',

$$R_{dst} \leftarrow (ER<<4 \ \& \ IR[3:0]) + R_{dst}$$

<116> 넷째, 분기 명령어의 경우를 예를 들어보자. 이를 위해 다음의 표5와 같이 분기명령어를 정의한다.



<118> & : IR[15 : 8], Op-코드(분기 명령)

<119> * : IR [7 : 0], 변위 데이터

<120> 표5 분기 명령의 일예

<121> 분기명령어에는 여러 가지의 종류가 있을 수 있으나 여기에서는 표5에서 보여지듯이 프로그램 카운터 관련(PC Relative)분기 명령어를 예로 사용하겠다. 이 명령어가 명령어 레지스터(40)에 래치되고 실행을 위해 디코더/콘트롤부(50)에서 필요한 제어 신호를 발생시킬 때 또한 E플래그를 참조한다. 그래서 만약 '0'이면 단순히 프로그램 카운터에 부호를 가진 8비트를 더한다. 즉, 현재의 프로그램 카운터에서 -512 ~ 511까지의 범위를 분기할 수 있다. 여기에서 데이터의 길이가 8비트인데 범위가 -512 ~511인 이유는 본 실시예에서 채택된 구조가 16비트의 명령어들을 사용하고 8비트 데이터 폭을 가진 메모리 4개를 병렬로 연결한 것을 전제로 하였기 때문에 최하위 비트는 의미가 없어지고, 따라서 실제 변위 값은 최하위 비트를 '0'으로 채우면 9비트가 되기 때문이다. 반면에 '1'이면 프로그램 카운터에 ER 레지스터를 참조하여 변위 값을 더하여 분기하고자 하는 유효 주소를 만든다. 즉, 분기하고자 하는 주소가 -512 ~ 511의 범위를 벗어나는 경우는 먼저 확장 명령어를 수행하여 필요한 양을 ER 레지스터에 옮겨 놓은 후 분기 명령어를 수행하여 필요한 변위를 얻을 수 있다는 것이다. 이 경우를 간단한 수식으로 표현하면 다음과 같다.

When E-Flag is '0',
EA <= (Sign extend IR[7:0] & '0') + PC

When E-Flag is '1',
EA <= (ER<<9 & IR[7:0] & '0') + PC

<123> 지금까지의 많은 명령어들 중, 몇 가지 대표적인 유형의 명령어에서 확장 명령어가 어떻게 적용될 수 있는지를 예를 통해 설명하였다. 여기에 설명되지는 않았지만 변위나, 즉시 값을 필요하는 명령어들도 이와 동일한 방법으로 확장 명령어의 적용이 가능하고 그 확장할 수 있는 양은 제한이 없다고 할 수 있다.

<124> 이제 확장 명령어가 수행될 때 예외가 발생하면 어떻게 처리되는지에 대해 살펴보자. 이때 고려해야 할 문제의 핵심은 확장 명령어 수행 중에 예외가 발생될 경우 그 상태를 예외 처리가 끝난 후까지 어떻게 유지하느냐와 또한 이미 옮겨 놓은 변위나 즉시 값을 어떻게 예외 처리 후 복원할 것인 가의 2 가지이다. 앞에서 설명하였듯이 본 발명은 이 문제를 해결하기 위해 E플래그와 ER레지스터를 사용하였다. 그리고 이를 구현하기 위한 예로서 도 4에서 보여지듯이 SPR 레지스터 파일 내에 ER 레지스터를 두고 또한 상태 레지스터내의 한 비트를 할당하여 E플래그로 사용하였다. 물론 이는 본 발명의 하드웨어로 구현하기 위한 많은 방법들 중에 설명의 용이성을 위하여 간략하게 나타낸 하나의 예라 하겠다.

발명의 효과

<125> 상술한 본 발명에 의하면 대부분의 MCU가 그렇듯이 예외가 발생하면 예외를 처리하기에 앞서 현재의 프로그램의 흐름을 잃지 않기 위해 몇 가지 요소를 스택 영역에 저장한다. 본 실시예의 구조도 프로그램 카운터 값과 상태 레지스터의 값을 스택 영역에 저장한다. 그래서 예외 처리 후에 프로그램 카운터와 상태 레지스터 값을 되돌려 받았을 때 E플래그 값을 참조하면 확장 명령어가 수행중일 때 예외가 발생할 경우 그 상태를 유지할 수 있고, 또한 ER레지스터를 소프트웨어적으로 접근 가능하게 하여 예외 처리 전에 이를 역시 스택영역에 옮겨 놓을 수 있도록 하여 예외 처리 후에 스택으로 부터 ER값을 복원 시킴으로서 두가지의 문제를 해결할 수 있다.

(57) 청구의 범위

청구항 1

접근 속도가 빠른 소규모 기억 장치인 레지스터의 집합인 레지스터 파일과 상기 레지스터 파일과 접속되어 정보를 송수신하는 통로인 내부버스와 상기 내부버스에 연결되어 있으며 외부 버스를 연결하는 외부 버스 버퍼와 상기 내부버스에 연결되어 있으며 연산 기능을 수행하는 기능블록과 상기 내부버스에 연결되어 있으며 수행중인 명령어를 기억하는 명령 레지스터와 상기 명령 레지스터에 연결되어 있어서 명령어를 해석하여 상기 레지스터 파일과 상기 내부 버스와 상기 외부버스 버퍼와 상기 기능 블록과 상기 명령 레지스터에 제어 신호를 발생하는 디코더/콘트롤부로 구성되며,

상기 레지스터 파일은 프로그래머가 접근 가능하고 연산 원시 데이터나 연산 결과 데이터를 저장하거나 상기 연산 데이터를 기억하고 있는 메모리의 번지 등을 기억하는 하나 또는 다수개의 범용 레지스터와 프로그래머가 접근 가능하고 중앙처리장치 동작에 필요한 정보를 기억하는 하나 또는 다수개의 특수 레지스터와 프로그래머가 접근할 수 없으며 중앙처리장치 동작에 필요한 특수한 기능이나 연산의 중간 과정 등을 기억하는 하나 또는 다수개의 내부 레지스터로 구성되어 있고,

상기 특수레지스터의 일종으로 진행하고 있는 프로그램이 저장되어 있는 메모리 번지를 기억하는 프로그램 카운터를 가지며, 상기 프로그램 카운터가 지정하는 프로그램 번지가 상기 내부 버스에 출력되고, 상기 내부 버스에 출력된 프로그램 번지가 상기 외부 버스 버퍼를 통하여 프로그램을 기억하고 있는 외부 메모리의 번지로 출력되고, 상기 방법으로 지정된 외부 메모리로부터 명령어가 읽혀져서 상기 외부

버스 버퍼를 통하여 상기 내부 버스에 연결되고, 상기 명령어가 상기 명령 레지스터에 저장되어 상기 디코더/콘트롤부에서 상기 제어신호를 발생하며,

상기 명령 레지스터에 저장되어 상기 디코더/콘트롤부에서 해석되는 상기 명령어는 동작을 지정하는 OP코드 필드만으로 구성되거나 상기 OP코드 필드와 동작의 작용을 받는 하나 또는 복수개의 오퍼랜드 필드로 구성되고, 상기 OP코드 필드와 상기 오퍼랜드 필드는 하나 또는 복수개의 2진수 비트로 구성되고, 상기 오퍼랜드 필드는 상기 명령어의 종류에 따라서 메모리 번지를 나타내는 번지 오퍼랜드나 상기 범용 레지스터나 상기 특수 레지스터에 기억된 메모리 번지로부터의 변위를 나타내는 변위 오퍼랜드나 연산이나 메모리 번지나 제어에 사용되기 위한 즉시 상수를 나타내는 즉시 상수 오퍼랜드 등으로 해석하고 실행하는 통상적이 중앙처리장치에서,

상기 특수 레지스터의 일종으로 프로그래머가 접근 가능한 확장 레지스터를 더 포함하고,

상기 확장 레지스터에 확장 데이터를 기억하는 동작을 나타내는 OP코드와 즉시 상수 오퍼랜드 필드로 구성된 명령어를 상기 디코더/콘트롤부에서 해석하여 즉시 상수 오퍼랜드 필드를 기억시키는 동작을 수행하며,

상기 OP코드와 OP코드가 필요로 하는 상기 오퍼랜드 필드의 일부분으로 구성되는 기본 명령어는 상기 확장 레지스터에 기억된 상기 확장 데이터가 상기 기본 명령어의 상기 일부분 오퍼랜드 필드와 연결되어서 완전한 오퍼랜드를 가진 명령어를 구성하여 상기 디코더/콘트롤부에서 해석되고 실행되는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 2

제 1항에 있어서, 상기 확장 레지스터의 길이는 상기 범용 레지스터의 길이와 같거나 짧은 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 3

제 2항에 있어서, 상기 확장 레지스터에 상기 확장 데이터를 기억하는 상기 명령어를 수행하면 상태가 변경되는 프로그래머가 접근 가능한 하나 또는 다수개의 비트로 구성되는 확장 플래그를 더 포함하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 4

제 3항에 있어서, 상기 확장 레지스터에 기억된 상기 확장 데이터를 연결하여 상기 오퍼랜드 필드를 형성하는 상기 기본 명령어가 수행되면 상기 확장 플래그의 상태를 변경하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 5

제 4항에 있어서, 상기 확장 플래그의 상태에 따라서 상기 기본 명령어가 가지는 상기 일부분 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장하여 상기 오퍼랜드를 형성하거나 상기 확장 레지스터에 기억된 확장 데이터와 상기 일부분 오퍼랜드를 연결하여 상기 오퍼랜드를 형성하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 6

제 2항에 있어서, 상기 확장 레지스터에 기억된 상기 확장 데이터를 연결하여 상기 오퍼랜드 필드를 형성하는 상기 기본 명령어가 수행되면 상기 확장 레지스터에 일정한 데이터를 기억시키는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 7

제 6항에 있어서, 상기 확장 레지스터에 기억된 확장 데이터의 값에 따라서 상기 기본 명령어가 가지는 상기 일부분 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장하여 상기 오퍼랜드를 형성하거나 상기 확장 레지스터에 기억된 확장 데이터와 상기 일부분 오퍼랜드를 연결하여 상기 오퍼랜드를 형성하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 8

제 4항에 있어서, 상기 확장 레지스터에 상기 확장 데이터를 기억하는 상기 명령어의 즉시 상수 오퍼랜드 필드의 길이가 상기 확장 레지스터의 길이보다 작으며, 상기 확장 레지스터에 상기 확장 데이터를 기억하는 명령어는 상기 확장 플래그의 상태에 따라서 즉시 상수 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장 레지스터의 길이만큼 확장하여 상기 확장 레지스터에 저장하거나 상기 확장 레지스터에 기억된 확장 데이터를 즉시 상수 오퍼랜드 길이만큼 왼쪽으로 이동시키고 즉시 상수 오퍼랜드를 연결하여 확장 데이터를 형성하여 상기 확장 레지스터에 저장하는 기능을 가지는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 9

제 8항에 있어서, 상기 확장 플래그의 상태에 따라서 상기 기본 명령어가 가지는 상기 일부분 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장하여 상기 오퍼랜드를 형성하거나 상기 확장 레지스터에 기억된 확장 데이터와 상기 일부분 오퍼랜드를 연결하여 상기 오퍼랜드를 형성하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 10

제 6항에 있어서, 상기 확장 레지스터에 상기 확장 데이터를 기억하는 상기 명령어의 즉시 상수

오퍼랜드 필드의 길이가 상기 확장 레지스터의 길이보다 작으며, 상기 확장 레지스터에 상기 확장 데이터를 기억하는 명령어는 상기 확장 레지스터에 저장되어 있는 상기 확장 데이터에 따라서 즉시 상수 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장 레지스터의 길이만큼 확장하여 상기 확장 레지스터에 저장하거나 상기 확장 레지스터에 기억된 확장 데이터를 즉시 상수 오퍼랜드 길이만큼 왼쪽으로 이동시키고 즉시 상수 오퍼랜드를 연결하여 확장 데이터를 형성하여 상기 확장 레지스터에 저장하는 기능을 가지는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 11

접근 속도가 빠른 소규모 기억장치인 레지스터의 집합인 레지스터 파일과 상기 레지스터 파일과 접속되어 정보를 송수신하는 통로인 내부버스와 상기 내부버스에 연결되어 있으며 외부버스를 연결하는 외부버스 버퍼와 상기 내부버스에 연결되어 있으며 연산 기능을 수행하는 기능블록과 상기 내부버스에 연결되어 있으며 수행중인 명령어를 기억하는 명령 레지스터와 상기 명령 레지스터에 연결되어 있어서 명령어를 해석하여 상기 레지스터 파일과 상기 내부 버스와 상기 외부버스 버퍼와 상기 기능 블록과 상기 명령 레지스터에 제어 신호를 발생하는 디코더/콘트롤부으로 구성되며,

상기 레지스터 파일은 프로그래머가 접근 가능하고 연산 원시 데이터나 연산 결과 데이터를 저장하거나 상기 연산 데이터를 기억하고 있는 메모리의 번지 등을 기억하는 하나 또는 다수개의 범용 레지스터와 프로그래머가 접근 가능하고 중앙처리장치 동작에 필요한 정보를 기억하는 하나 또는 다수개의 특수 레지스터와 프로그래머가 접근할 수 없으며 중앙처리장치 동작에 필요한 특수한 기능이나 연산의 중간 과정 등을 기억하는 하나 또는 다수개의 내부 레지스터로 구성되어 있고,

상기 특수 레지스터의 일종으로 진행하고 있는 프로그램이 저장되어 있는 메모리 번지를 기억하는 프로그램 카운터를 가지며, 상기 프로그램 카운터가 지정하는 프로그램 번지가 상기 내부버스에 출력되고, 상기 내부 버스에 출력된 프로그램 번지가 상기 외부버스 버퍼를 통하여 프로그램을 기억하고 있는 외부 메모리의 번지로 출력되고, 상기 방법으로 지정된 외부메모리로부터 명령어가 읽혀져서 상기 외부 버스 버퍼를 통하여 상기 내부 버스에 연결되고, 상기 명령어가 상기 명령 레지스터에 저장되어 상기 디코더/콘트롤부에서 상기 제어신호를 발생하며,

상기 명령 레지스터에 저장되어 상기 디코더/콘트롤부에서 해석되는 상기 명령어는 동작을 지정하는 OP코드 필드만으로 구성되거나 상기 OP코드 필드와 동작의 작용을 받는 하나 또는 복수개의 오퍼랜드 필드로 구성되고, 상기 OP코드 필드와 상기 오퍼랜드 필드는 하나 또는 복수개의 2진수 비특로 구성되고, 상기 오퍼랜드 필드는 상기 명령어의 종류에 따라서 메모리 번지를 나타내는 번지 오퍼랜드나 상기 범용 레지스터나 상기 특수 레지스터에 기억된 메모리 번지로부터의 변위를 나타내는 변워 오퍼랜드나 연산이나 메모리 번지나 제어에 사용되기 위한 즉시 상수를 나타내는 즉시 상수 오퍼랜드 등으로 해석하고 실행하는 통상적인 중앙처리장치에 있어서, 상기 특수 레지스터의 일종으로 프로그래머가 접근 가능한 확장 레지스터를 더 포함하고,

상기 확장 레지스터는 확장 데이터 필드와 중앙처리장치의 동작에 필요한 정보나 플래그 및 기타 필드로 구성되고,

상기 확장 레지스터의 확장 데이터 필드에 확장 데이터를 기억하는 동작을 나타내는 OP코드와 즉시 상수 오퍼랜드 필드로 구성된 명령어를 상기 디코더/콘트롤부에서 해석하여 즉시 상수 오퍼랜드 필드를 확장 레지스터의 확장 데이터 필드에 기억시키는 동작을 수행하며,

상기 OP코드와 OP코드가 필요로 하는 상기 오퍼랜드 필드의 일부분으로 구성되는 기본명령어는 상기 확장 레지스터의 확장 데이터 필드에 기억된 상기 확장 데이터가 상기 기본 명령어의 상기 일부분 오퍼랜드 필드와 연결(concatenation)되어서 완전한 오퍼랜드를 가진 명령어를 구성하여 상기 디코더/콘트롤부에서 해석되고 실행되는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 12

제 11항에 있어서, 상기 확장 레지스터의 확장 데이터 필드에 상기 확장 데이터를 기억하는 상기 명령어를 수행하면 상태가 변경되는 프로그래머가 접근 가능한 하나 또는 다수개의 비트로 구성되는 확장 플래그를 더 포함하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 13

제 12항에 있어서, 상기 확장 레지스터의 확장 데이터필드에 기억된 상기 확장 데이터를 연결하여 상기 오퍼랜드 필드를 형성하는 상기 기본 명령어가 수행되면 상기 확장 플래그의 상태를 변경하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 14

제 13항에 있어서, 상기 확장 플래그의 상태에 따라서 상기 기본 명령어가 가지는 상기 일부분 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장하여 상기 오퍼랜드를 형성하거나 상기 확장 레지스터의 확장 데이터 필드에 기억된 확장 데이터와 상기 일부분 오퍼랜드를 연결하여 상기 오퍼랜드를 형성하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 15

제 11항에 있어서, 상기 확장 레지스터의 확장 데이터 필드에 기억된 상기 확장 데이터를 연결하여 상기 오퍼랜드 필드를 형성하는 상기 기본 명령어가 수행되면 상기 확장레지스터의 확장 데이터 필드에 일정한 데이터를 기억시키는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 16

제 15항에 있어서, 상기 확장 레지스터의 확장 데이터 필드에 기억된 확장 데이터의 값에 따라서

상기 기본 명령어가 가지는 상기 일부분 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장하여 상기 오퍼랜드를 형성하거나 상기 확장 레지스터의 확장 데이터 필드에 기억된 확장 데이터와 상기 일부분 오퍼랜드를 연결하여 상기 오퍼랜드를 형성하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 17

제 13항에 있어서, 상기 확장 레지스터의 확장 데이터 필드에 상기 확장 데이터를 기억하는 상기 명령어의 즉시 상수 오퍼랜드 필드의 길이가 상기 확장 레지스터의 확장 데이터 필드의 길이보다 작으며, 상기 확장 레지스터의 확장 데이터 필드에 상기 확장 데이터를 기억하는 명령어는 상기 확장 플래그의 상태에 따라서 즉시 상수 오퍼랜드를 양의 저수나 2의 보수로 보아서 확장 데이터 필드 길이만큼 확장하여 상기 레지스터의 확장 데이터 필드에 저장하거나 상기 확장 레지스터의 확장 데이터 필드에 기억된 확장 데이터를 즉시 상수 오퍼랜드 길이만큼 왼쪽으로 이동시키고 즉시 상수 오퍼랜드를 연결하여 확장 데이터를 형성하여 상기 확장 레지스터의 확장 데이터 필드에 저장하는 기능을 가지는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 18

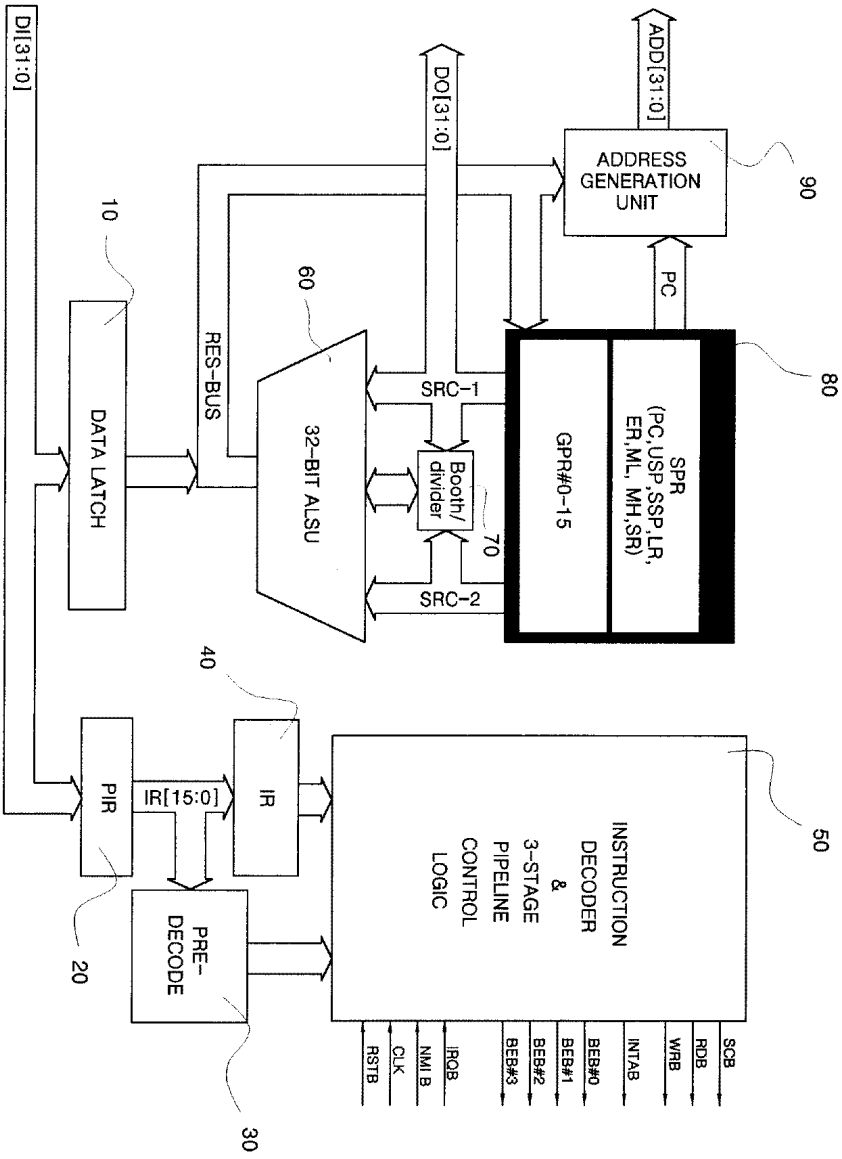
제 17항에 있어서, 상기 확장 플래그의 상태에 따라서 상기 기본 명령어가 가지는 상기 일부분 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장하여 상기 오퍼랜드를 형성하거나 상기 확장 레지스터의 확장 데이터 필드에 기억된 확장 데이터와 상기 일부분 오퍼랜드를 연결하여 상기 오퍼랜드를 형성하는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

청구항 19

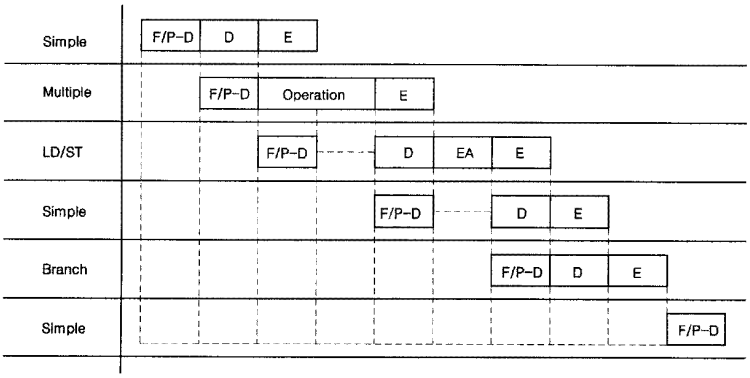
제 15항에 있어서, 상기 확장 레지스터의 확장 데이터 필드에 상기 확장 데이터를 기억하는 상기 명령어의 즉시 상수 오퍼랜드 필드의 길이가 상기 확장 레지스터의 확장 데이터 필드와 길이보다 작으며, 상기 확장 레지스터의 확당 데이터 필드에 상기 확장 데이터를 기억하는 명령어는 상기 확장 레지스터의 확장 데이터 필드에 저장되어 있는 상기 확장 데이터에 따라서 즉시 상수 오퍼랜드를 양의 정수나 2의 보수로 보아서 확장 레지스터의 확장 데이터 필드 길이만큼 확장하여 상기 확장 레지스터의 확장 데이터 필드에 저장하거나 상기 확장 레지스터의 확장 데이터 필드에 기억된 확장 데이터를 즉시 상수 오퍼랜드 길이만큼 왼쪽으로 이동시키고 즉시 상수 오퍼랜드를 연결하여 확장 데이터를 형성하여 상기 확장 레지스터의 확장 데이터 필드에 저장하는 기능을 가지는 것을 특징으로 하는 확장 명령어를 가진 중앙처리장치.

도면

도면1

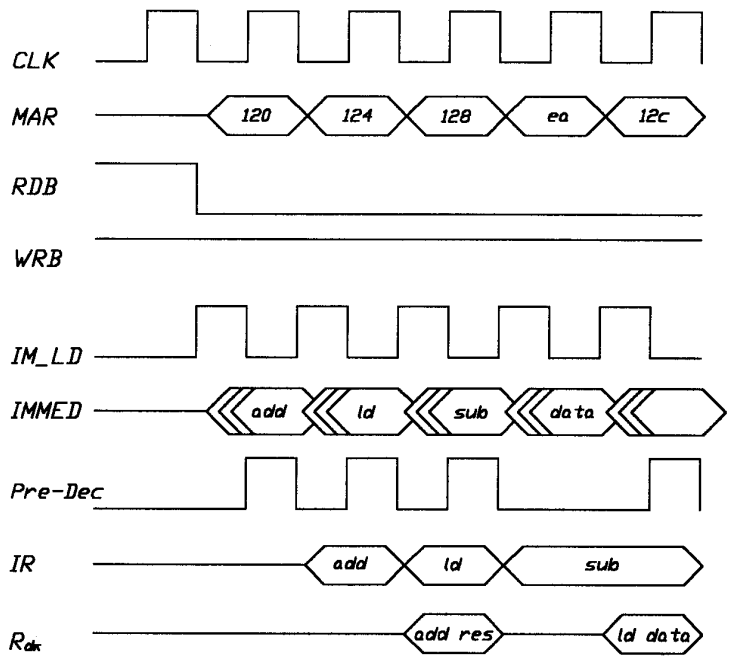

DI[31:0]
DO[31:0]
ADD[31:0]
ADDRESS GENERATION UNIT
90
10
60
PC
80
RES-BUS
SRC-1
DATA LATCH
32-BIT ALSU
Booth/ divider
70
GPR#0-15
SPR (PC,USP,SSP,LR, ER,ML, MH,SR)
SRC-2
40
PIR
IR[15:0]
IR
INSTRUCTION DECODER & 3-STAGE PIPELINE CONTROL LOGIC
50
PRE-DECODE
20
30
RSTB
CLK
NMI B
IRQB
BEB#3
BEB#2
BEB#1
BEB#0
INTAB
WRB
RDB
SCB

도면2

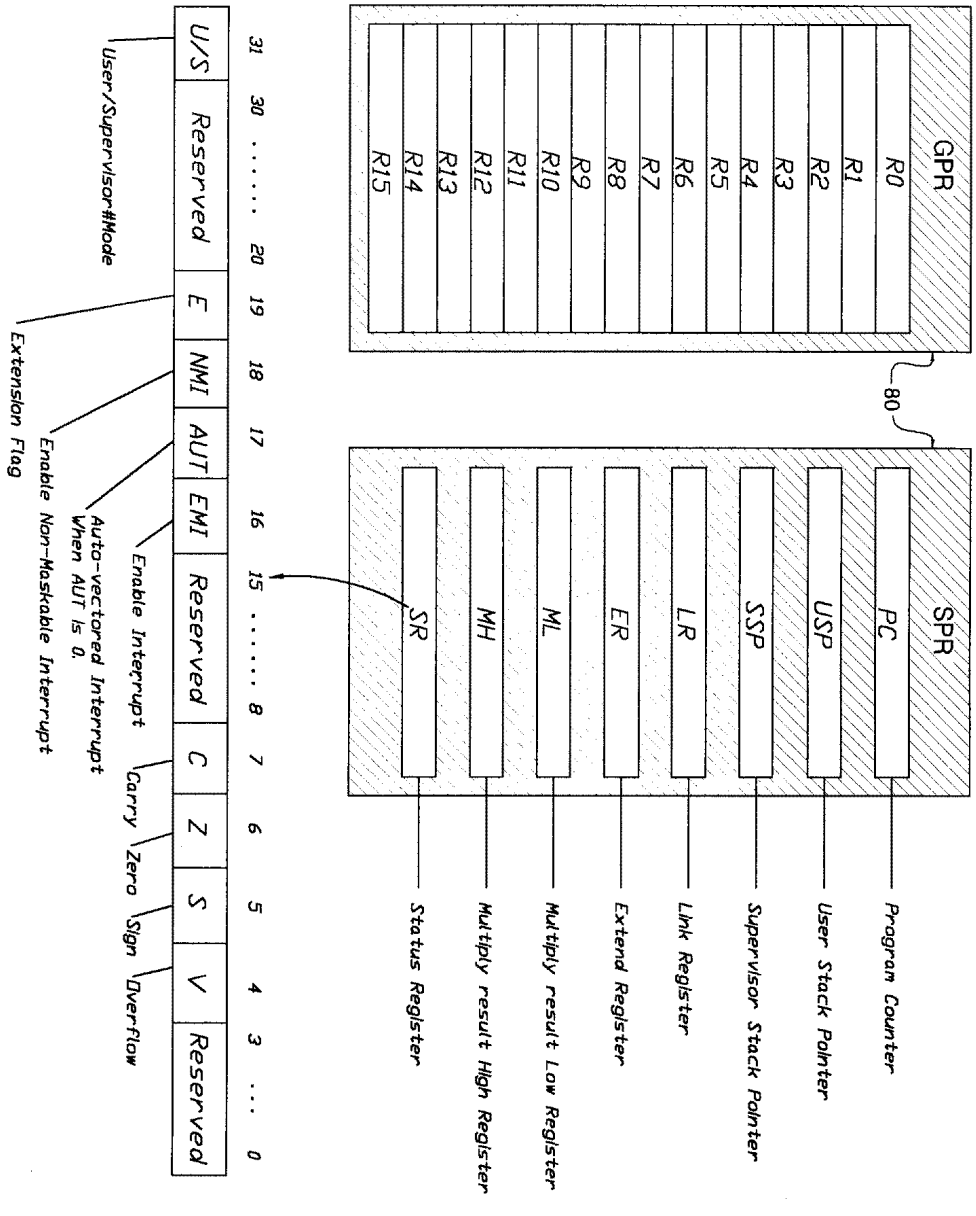


F/P-D : Fetch,Pre-Decode Stage
D : Decode Stage
E : Execution/Transfer Stage
EA : Effective Address Caculation stage

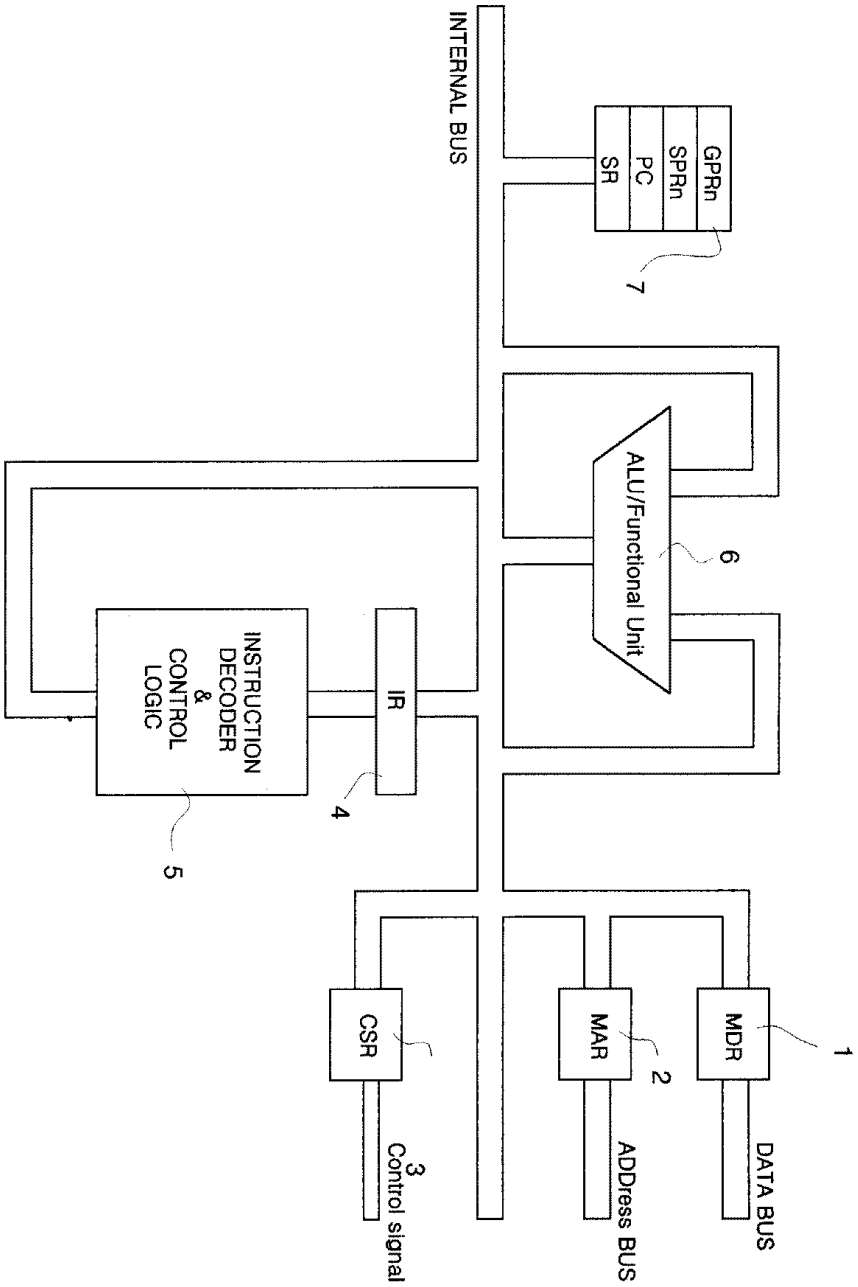
도면3



도면4


GPR
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
80
SPR
PC
USP
SSP
LR
ER
ML
MH
SR
Program Counter
User Stack Pointer
Supervisor Stack Pointer
Link Register
Extend Register
Multiply result Low Register
Multiply result High Register
Status Register
31 30 ..... 20 19 18 17 16 15 ..... 8 7 6 5 4 3 ... 0
U/S
Reserved
E
NMI
AUT
EMI
Reserved
C
Z
S
V
Reserved
User/Supervisor#Mode
Extension Flag
Enable Non-Maskable Interrupt
Auto-vectored Interrupt When AUT is 0.
Enable Interrupt
Carry
Zero
Sign
Overflow

도면5


INTERNAL BUS
GPRn
SPRn
PC
SR
7
ALU/Functional Unit
6
INSTRUCTION DECODER & CONTROL LOGIC
IR
4
5
CSR
MAR
MDR
1
2
3
Control signal
ADDress BUS
DATA BUS