(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2010년02월04일
(11) 등록번호 10-0940465
(24) 등록일자 2010년01월28일

(51) Int. Cl.
G06F 9/30 (2006.01)
(21) 출원번호 10-2000-7010577
(22) 출원일자 1999년03월04일
심사청구일자 2004년03월04일
(85) 번역문제출일자 2000년09월18일
(65) 공개번호 10-2001-0082524
(43) 공개일자 2001년08월30일
(86) 국제출원번호 PCT/US1999/004887
(87) 국제공개번호 WO 1999/47999
국제공개일자 1999년09월23일
(30) 우선권주장
09/044,087 1998년03월18일 미국(US)
(뒷면에 계속)
(56) 선행기술조사문헌
US05522083
US05673410
전체 청구항 수 : 총 41 항

(73) 특허권자
퀄컴 인코포레이티드
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(72) 발명자
시길버트씨
미국92124캘리포니아주샌디에고피핏플레이스7804
조우퀴즈헨
미국92126
캘리포니아주샌디에고웨스트뷰파크웨이11507
(뒷면에 계속)
(74) 대리인
특허법인코리아나

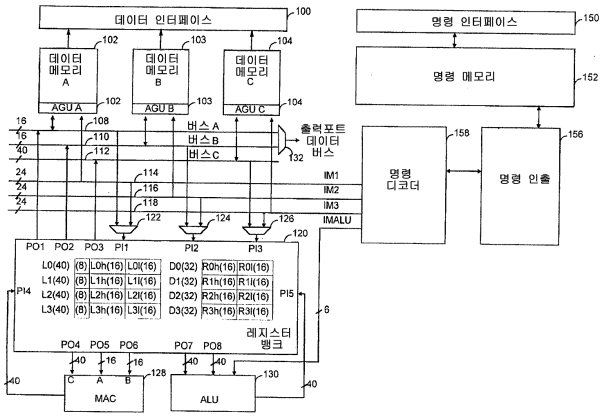
심사관 : 임영희

(54) 디지털 신호처리기

(57) 요 약

디지털 신호처리회로는 가변 길이 명령집합의 사용을 필요로 한다. 전형적인 DSP 는 데이터가 레지스터 뱅크 (120) 및 3개의 데이터 메모리 (102, 103, 104) 와 교환될 수 있는 일군의 3개의 데이터버스 (108, 110, 112) 를 포함한다. 적어도 2개의 처리부 (128, 130) 에 의해 액세스가능한 레지스터를 갖는 레지스터 뱅크 (120) 가 사용될 수 있다. 명령 메모리 (152) 에 저장된 가변 길이의 명령을 수신하는 명령인출부 (156) 가 포함될 수 있다. 명령 메모리 (152) 는 상기 일군의 3개의 데이터 메모리 (102, 103, 104) 로부터 분리될 수 있다.

대 표 도



(72) 발명자

자산자이케이

미국92126캘리포니아주샌디에고신테일러레인7415

강인엽

미국92122
캘리포니아주샌디에고토스카나웨이넘버3265385

린지안

미국92126캘리포니아주샌디에고스킬링애브뉴7164

모티왈라퀘이드

미국92122
캘리포니아주샌디에고팔밀라드라이브넘버53297665

존디푸

미국92037
캘리포니아주샌디에고라졸라리젠츠로드9232아파트먼트이

장리

미국92126
캘리포니아주샌디에고카미노루이즈넘버1110162

장하이타오

미국92037캘리포니아주라졸라미라말스트리트3765비

리웨이싱

미국92129캘리포니아주샌디에고파우커드웨이8555

사카마키찰스이

미국92126캘리포니아주샌디에고아쿠아리우스드라이브8771

칸타크프라샨트에이

미국92037
캘리포니아주라졸라빌라라졸라드라이브8524

(81) 지정국

국내특허 : 알바니아, 아르메니아, 오스트리아, 오스트레일리아, 아제르바이잔, 보스니아 헤르체고비나, 바베이도스, 불가리아, 브라질, 벨라루스, 캐나다, 스위스, 중국, 쿠바, 체코, 독일, 덴마크, 에스토니아, 스페인, 핀란드, 영국, 그라나다, 그루지야, 가나, 감비아, 크로아티아, 헝가리, 인도네시아, 이스라엘, 인도, 아이슬랜드, 일본, 케냐, 키르키즈스탐, 북한, 대한민국, 카자흐스탄, 세인트루시아, 스리랑카, 리베이라, 레소토, 리투아니아, 룩셈부르크, 라트비아, 몰도바, 마다가스카르, 마케도니아공화국, 몽고, 말라위, 멕시코, 노르웨이, 뉴질랜드, 폴란드, 포르투칼, 루마니아, 러시아, 수단, 스웨덴, 싱가포르, 슬로베니아, 슬로바키아, 시에라리온, 타지키스탄, 투르크맨, 터어키, 트리니아드토바고, 우크라이나, 우간다, 우즈베키스탄, 베트남, 세르비아 앤 몬테네그로, 짐바브웨

AP ARIPO특허 : 가나, 감비아, 케냐, 레소토, 말라위, 수단, 시에라리온, 스와질랜드, 우간다, 짐바브웨

EA 유라시아특허 : 아르메니아, 아제르바이잔, 벨라루스, 키르키즈스탐, 카자흐스탄, 몰도바, 러시아, 타지키스탄, 투르크맨

EP 유럽특허 : 오스트리아, 벨기에, 스위스, 사이프러스, 독일, 덴마크, 스페인, 핀란드, 프랑스, 영국, 그리스, 아일랜드, 이탈리아, 룩셈부르크, 모나코, 네덜란드, 포르투칼, 스웨덴

OA OAPI특허 : 부르키나파소, 베닌, 중앙아프리카, 콩고, 코트디브와르, 카메룬, 가봉, 기니, 기니 비사우, 말리, 모리타니, 니제르, 세네갈, 차드, 토고

(30) 우선권주장

09/044,088 1998년03월18일 미국(US)
09/044,089 1998년03월18일 미국(US)
09/044,104 1998년03월18일 미국(US)
09/044,086 1998년03월18일 미국(US)
09/044,108 1998년03월18일 미국(US)

특허청구의 범위

청구항 1

가변 수의 명령 프래그먼트를 포함하는 가변 길이 명령에 응답하여 디지털 신호처리를 수행하는 디지털 신호처리기로서,

하나 이상의 최대길이 완전 명령에 대하여 충분한, 상기 가변 길이 명령을 포함하는 명령 데이터를 인출하는 명령 인출부;

상기 명령을 디코딩하고 제어신호를 생성하는 명령 디코더;

처리될 데이터를 저장하는 레지스터 뱅크;

제 1 데이터를 저장하는 제 1 메모리 뱅크;

제 2 데이터를 저장하는 제 2 메모리 뱅크;

제 3 데이터를 저장하는 제 3 메모리 뱅크;

상기 제 1 데이터를 판독하여 상기 레지스터 뱅크내의 제 1 레지스터로 입력하는 제 1 데이터 버스;

상기 제 2 데이터를 판독하여 상기 레지스터 뱅크내의 제 2 레지스터로 입력하는 제 2 데이터 버스;

상기 제 1 레지스터 및 상기 제 2 레지스터내에서 처리될 데이터를 처리하고 결과를 상기 레지스터 뱅크내의 제 3 레지스터로 기입하는 제 1 처리부; 및

상기 결과를 상기 제 3 레지스터로부터 상기 제 3 메모리 뱅크로 기입하는 제 3 데이터 버스

를 포함하는 것을 특징으로 하는 디지털 신호처리기.

청구항 2

제 1 항에 있어서,

상기 결과를 처리하여 더 처리된 결과를 생성하고 상기 더 처리된 결과를 상기 레지스터 뱅크내의 포스 (forth) 레지스터내에 저장하는 제 2 처리부를 더 포함하고,

상기 제 3 데이터 버스는 상기 더 처리된 결과를 상기 포스 레지스터로부터 상기 제 3 메모리로 더 기입하는 것을 특징으로 하는 디지털 신호처리기.

청구항 3

제 1 항에 있어서,

상기 제 3 데이터 버스는 상기 제 1 데이터 버스 및 상기 제 2 데이터 버스보다 더 넓은 것을 특징으로 하는 디지털 신호처리기.

청구항 4

제 2 항에 있어서,

상기 레지스터 뱅크내의 제 1 군의 레지스터는 상기 제 1 처리부 및 상기 제 2 처리부 모두에 기입할 수 있고, 상기 레지스터 뱅크내의 제 2 군의 레지스터는 상기 제 1 처리부 또는 상기 제 2 처리부 집합 중의 하나에 기입할 수 있는 것을 특징으로 하는 디지털 신호처리기.

청구항 5

제 2 항에 있어서,

상기 레지스터 뱅크내의 제 1 군의 레지스터는 상기 제 1 처리부 및 상기 제 2 처리부 모두로부터 판독할 수 있고, 상기 레지스터 뱅크내의 제 2 군의 레지스터는 상기 제 1 처리부 또는 상기 제 2 처리부 집합 중의 하나로부터 판독할 수 있는 것을 특징으로 하는 디지털 신호처리기.

청구항 6

명령 데이터에 응답하여 디지털신호를 처리하는 디지털 신호처리기로서, 상기 디지털신호처리기는,

처리되는 데이터를 저장하는 레지스터 뱅크;

제 1 데이터를 저장하는 제 1 메모리 뱅크;

제 2 데이터를 저장하는 제 2 메모리 뱅크;

제 3 데이터를 저장하는 제 3 메모리 뱅크;

상기 레지스터 뱅크의 제 1 레지스터로 상기 제 1 데이터를 판독하는 제 1 데이터 버스;

상기 레지스터 뱅크의 제 2 레지스터로 상기 제 2 데이터를 판독하는 제 2 데이터 버스;

상기 제 1 레지스터 및 상기 제 2 레지스터에서 처리되는 데이터를 처리하고, 그리고 상기 레지스터 뱅크의 제 3 레지스터로 결과를 기입하는 제 1 처리부; 및

상기 제 3 레지스터로부터 상기 제 3 메모리 뱅크로 상기 결과를 기입하는 제 3 데이터 버스를 포함하고,

상기 제 1, 제 2 및 제 3 데이터 버스는 동시에 동작가능하고 상기 레지스터 뱅크와 상기 제 1, 제 2 및 제 3 메모리 뱅크 사이에서 각각 데이터를 교환하는 것을 특징으로 하는 디지털 신호처리기.

청구항 7

제 6 항에 있어서,

상기 결과를 처리함으로써 더 처리된 결과를 생성하고, 상기 결과를 상기 레지스터 뱅크의 제 4 레지스터에 저장하는 제 2 처리부를 더 포함하고,

상기 제 3 데이터 버스는 상기 제 4 레지스터로부터 상기 제 3 메모리로 상기 더 처리된 결과를 더 기입하는 것을 특징으로 하는 디지털 신호처리기.

청구항 8

명령 데이터에 응답하여 디지털신호를 처리하는 디지털 신호처리기로서, 상기 디지털신호는,

제 1 메모리와 레지스터 뱅크 사이에 데이터를 판독 및 기입하기 위한 제 1 데이터 버스;

제 2 메모리와 상기 레지스터 뱅크 사이에 데이터를 판독 및 기입하기 위한 제 2 데이터 버스;

제 3 메모리와 상기 레지스터 뱅크 사이에 데이터를 판독 및 기입하기 위한 제 3 데이터 버스를 포함하는 군을 포함하고,

가변 길이의 명령을 인출하는 명령 인출부로서, 상기 가변 길이 명령은 일군의 동작을 요청하는, 명령 인출부; 및

상기 가변 길이의 명령을 디코딩하고 상기 일군의 동작이 수행되게 하는 명령 디코더를 더 포함하며,

상기 제 1 데이터 버스, 상기 제 2 데이터 버스, 및 상기 제 3 데이터 버스는 동시에 동작하는 것을 특징으로 하는 디지털 신호처리기.

청구항 9

제 6 항에 있어서,

상기 제 3 데이터 버스는 상기 제 1 데이터 버스 및 상기 제 2 데이터 버스보다 넓은 것을 특징으로 하는 디지털 신호처리기.

청구항 10

제 1 메모리 뱅크로부터 제 1 데이터 버스를 통해 제 1 데이터를 수신하는 단계;

상기 제 1 데이터를 제 1 레지스터에 저장하는 단계;

제 2 메모리 뱅크로부터 제 2 데이터 버스를 통해 제 2 데이터를 수신하는 단계;

상기 제 2 데이터를 제 2 레지스터에 저장하는 단계;

상기 제 1 데이터 및 상기 제 2 데이터를 사용하여 결과를 생성하는 단계;

상기 결과를 제 3 레지스터에 저장하는 단계;

상기 결과를 제 3 데이터 버스를 통해 제 3 메모리 뱅크로 기입하는 단계를 포함하고,

상기 제 1, 제 2 및 제 3 데이터 버스는 동시에 동작가능하고 상기 레지스터들과 상기 제 1, 제 2 및 제 3 메모리 뱅크 사이에서 각각 데이터를 교환하는 것을 특징으로 하는 데이터 처리방법.

청구항 11

제 10 항에 있어서,

상기 제 1 데이터는 제 1 데이터값을 포함하고, 그리고

상기 제 2 데이터는 제 2 데이터 값을 포함하는 것을 특징으로 하는 데이터 처리방법.

청구항 12

제 10 항 또는 제 11 항에 있어서,

상기 제 3 데이터 버스는 상기 제 1 데이터 버스보다 넓은 것을 특징으로 하는 데이터 처리방법.

청구항 13

제 10 항 또는 제 11 항에 있어서,

상기 제 3 데이터 버스는 상기 제 1 데이터 버스 및 상기 제 2 데이터 버스보다 넓은 것을 특징으로 하는 데이터 처리방법.

청구항 14

제 10 항에 있어서,

제 1 처리부를 사용하여 상기 결과를 생성하는 단계; 및

상기 제 3 레지스터에 접속된 제 2 처리부를 사용하여 상기 제 1 결과로부터 제 2 결과를 생성하는 단계를 더 포함하는 것을 특징으로 하는 데이터 처리방법.

청구항 15

제 7 항에 있어서,

상기 레지스터 뱅크의 제 1 세트의 레지스터는 상기 제 1 처리부 및 상기 제 2 처리부 모두로 기입할 수 있고, 그리고

상기 레지스터 뱅크의 제 2 세트의 레지스터는 상기 제 1 처리부 또는 상기 제 2 처리부의 어느 하나로 기입할 수 있는 것을 특징으로 하는 디지털 신호처리기.

청구항 16

제 7 항에 있어서,

상기 레지스터 뱅크의 제 1 세트의 레지스터는 상기 제 1 처리부 및 상기 제 2 처리부 모두로부터 판독할 수 있고, 그리고

상기 레지스터 뱅크의 제 2 세트의 레지스터는 상기 제 1 처리부 또는 상기 제 2 처리부의 어느 하나로부터 판독할 수 있는 것을 특징으로 하는 디지털 신호처리기.

청구항 17

제 6 항, 제 7 항, 제 9 항 또는 제 15 항 중 어느 한 항에 있어서,

상기 명령 데이터는 가변 길이 명령을 포함하고, 상기 가변 길이 명령은 가변 개수의 명령 프래그먼트를 포함하며,

상기 디지털 신호처리기는, 하나 이상의 최대길이 완전 명령에 대하여 충분한, 상기 가변 길이 명령을 포함하는 명령 데이터를 인출하는 명령 인출부; 및 상기 명령을 디코딩하고 제어 신호를 생성하는 명령 디코더를 더 포함하는 것을 특징으로 하는 디지털 신호처리기.

청구항 18

제 6 항에 있어서,

상기 제 1 데이터 버스는 상기 제 1 메모리 뱅크와 상기 레지스터 뱅크 사이에서 데이터를 더 기입하고;

상기 제 2 데이터 버스는 상기 제 2 메모리 뱅크와 상기 레지스터 뱅크 사이에서 데이터를 더 기입하고;

상기 제 3 데이터 버스는 상기 제 3 메모리 뱅크와 상기 레지스터 뱅크 사이에서 데이터를 더 기입하며,

상기 제 1 데이터 버스, 상기 제 2 데이터 버스, 및 상기 제 3 데이터 버스는 동시에 동작하는 것을 특징으로 하는 디지털 신호처리기.

청구항 19

제 18 항에 있어서,

가변 길이의 명령을 인출하는 명령 인출부로서, 상기 가변 길이 명령은 일군의 동작을 요청하는, 명령 인출부;

상기 가변 길이의 명령을 디코딩하고 상기 일군의 동작이 수행되도록 하는 명령 디코더를 더 포함하는 것을 특징으로 하는 디지털 신호처리기.

청구항 20

제 18 항에 있어서,

상기 제 1 처리부와 동시에 상기 레지스터 뱅크의 데이터를 프로세싱하는 제 2 처리부를 더 포함하는 것을 특징으로 하는 디지털 신호처리기.

청구항 21

제 6 항에 있어서,

상기 제 1 데이터 버스는 상기 제 3 데이터 버스보다 좁은 것을 특징으로 하는 디지털 신호처리기.

청구항 22

제 6 항에 있어서,

상기 제 1 데이터 버스, 상기 제 2 데이터 버스, 및 상기 제 3 데이터 버스를 제어하는 제어 시스템을 더 포함하는 것을 특징으로 하는 디지털 신호처리기.

청구항 23

제 7 항에 있어서,

상기 제 1 처리부는 곱셈-누산 유닛 (Multiply-Accumulate Unit; MAC) 이고,

상기 제 2 처리부는 산술 논리 유닛 (Arithmetic Logic Unit; ALU) 인 것을 특징으로 하는 디지털 신호처리기.

청구항 24

제 7 항에 있어서,

상기 처리부들은 각각의 데이터 버스로부터 데이터를 수신하도록 동작가능한 것을 특징으로 하는 디지털 신호처리기.

청구항 25

제 7 항에 있어서,

제 1 세트의 레지스터는 상기 제 1 처리부 및 상기 제 2 처리부 모두로 데이터를 기입하도록 동작가능하고;

제 2 세트의 레지스터는 상기 제 1 처리부로 데이터를 기입하지만, 상기 제 2 처리부로는 데이터를 기입하지 않도록 동작가능한 것을 특징으로 하는 디지털 신호처리기.

청구항 26

각각이 동작을 요청하는 일군의 명령 프래그먼트를 갖는 가변 길이 명령을 사용하여 디지털 신호처리기를 동작시키는 방법으로서,

a) 제 1 클록사이클의 제 1 클록위상 동안, 제 1 레지스터로부터 사전에 처리된 데이터를 판독하여 제 1 처리부로 입력하는 단계;

b) 상기 일군의 명령 프래그먼트로부터의 제 1 명령 프래그먼트에 기초하여 상기 사전에 처리된 데이터를 처리하고, 상기 제 1 클록사이클 동안, 2 배의 처리된 데이터를 산출하는 단계;

c) 상기 일군의 명령 프래그먼트로부터의 제 2 명령 프래그먼트에 기초하여 새로운 데이터를 처리하고, 상기 제 1 클록사이클 동안, 새로운 처리된 데이터를 산출하는 단계;

d) 상기 제 1 클록사이클의 제 2 위상 동안, 상기 새로운 처리된 데이터를 상기 제 1 레지스터로 기입하는 단계; 및

e) 상기 제 1 클록사이클의 상기 제 2 위상 동안, 상기 2 배의 처리된 데이터를 제 2 레지스터로 기입하는 단계

를 포함하는 것을 특징으로 하는 디지털 신호처리기를 동작시키는 방법.

청구항 27

제 26 항에 있어서,

단계 b) 는 제 1 처리부에 의해 수행되고, 단계 c) 는 제 2 처리부에 의해 수행되는 것을 특징으로 하는 디지털 신호처리기를 동작시키는 방법.

청구항 28

제 26 항에 있어서,

상기 가변 길이 명령을 포함하는 명령 데이터를 판독하는 단계;

다음 명령 길이를 결정하는 단계; 및

상기 다음 명령 길이와 동일한 상기 명령 데이터 내의 데이터양을 디코딩하는 단계를 더 포함하는 것을 특징으로 하는 디지털 신호처리기를 동작시키는 방법.

청구항 29

제 1 입력 데이터 및 제 2 입력 데이터에 응답하여 결과 데이터를 생성하는 복수의 처리부;

상기 제 1 및 제 2 입력 데이터를 상기 복수의 처리부 중 적어도 제 1 및 제 2 처리부로 각각 전송하는 제 1 데이터 버스 및 제 2 데이터 버스;

상기 결과 데이터를 상기 복수의 처리부 중 적어도 제 3 처리부로 전송하는 제 3 데이터 버스; 및

상기 복수의 처리부 중 적어도 상기 제 1 및 제 2 처리부에 의하여 액세스가능한 레지스터들을 가지고, 상기 복수의 처리부 중 2 이상에 의하여 동시에 수행되는 다중 데이터 동작을 용이하게 하는, 레지스터 뱅크를 포함하고,

상기 제 1 및 제 2 데이터 버스 중 하나 이상은 상기 제 3 데이터 버스보다 좁고, 그리고

단일 명령 및 연관된 레지스터의 데이터에 응답하여, 상기 제 1, 제 2 및 제 3 데이터 버스는 대응하는 상기 복

수의 처리부 중 적어도 제 1, 제 2, 및 제 3 처리부에 통신가능하게 접속되는 것을 특징으로 하는 디지털 신호처리기.

청구항 30

제 29 항에 있어서,

상기 제 1, 제 2 및 제 3 데이터 버스의 통신가능한 접속을 제어하는 제어 로직을 더 포함하는 것을 특징으로 하는 디지털 신호처리기.

청구항 31

제 29 항에 있어서,

상기 복수의 처리부 중 상기 제 1 처리부는 곱셈-누산 유닛 (MAC) 인 것을 특징으로 하는 디지털 신호처리기.

청구항 32

제 31 항에 있어서,

상기 복수의 처리부 중 상기 제 2 처리부는 산술 논리 유닛 (ALU) 인 것을 특징으로 하는 디지털 신호처리기.

청구항 33

제 29 항에 있어서,

상기 복수의 처리부는 하나 이상의 산술 논리 유닛 (ALU) 및 하나의 곱셈-누산 유닛 (MAC) 을 포함하는 것을 특징으로 하는 디지털 신호처리기.

청구항 34

제 29 항에 있어서,

상기 레지스터 뱅크는 상기 복수의 처리부 중 적어도 제 1 및 제 2 처리부에 의해 데이터의 인터럽트되지 않은 파이프라인 처리를 더 용이하게 하는 것을 특징으로 하는 디지털 신호처리기.

청구항 35

삭제

청구항 36

삭제

청구항 37

삭제

청구항 38

삭제

청구항 39

디지털 신호처리기를 동작시키는 방법으로서,

제 1 클록사이클의 제 1 클록위상 동안, 레지스터로부터의 처리된 데이터를 판독하여 제 1 처리부로 입력하는 단계;

상기 제 1 클록사이클 동안, 상기 제 1 처리부를 사용하여 상기 처리된 데이터를 처리하고 더 처리된 데이터를 산출하는 단계;

상기 제 1 클록사이클 동안, 제 2 처리부 내의 다른 데이터를 처리하여 새로운 처리된 데이터를 산출하는 단계;

상기 제 1 클록사이클의 제 2 위상 동안, 상기 새로운 처리된 데이터를 상기 레지스터에 기입하는 단계

를 포함하는 것을 특징으로 하는 디지털 신호처리기를 동작시키는 방법.

청구항 40

제 39 항에 있어서,

상기 제 1 클록사이클의 상기 제 2 위상 동안, 상기 더 처리된 데이터를 제 2 레지스터에 기입하는 단계를 더 포함하는 것을 특징으로 하는 디지털 신호처리기를 동작시키는 방법.

청구항 41

삭제

청구항 42

삭제

청구항 43

어드레스 공간을 갖는 메모리내에 저장된 명령을 사용하여 디지털 신호처리기를 제어하는 방법으로서,

제 1 명령의 제 1 부분을 포함하는 상기 메모리의 제 1 데이터 워드를 기입하는 단계; 및

상기 제 1 명령의 제 2 부분 및 제 2 명령의 제 1 부분을 포함하는 상기 메모리의 제 2 데이터 워드를 기입하는 단계를 포함하고,

상기 제 1 명령 및 상기 제 2 명령은 일군의 명령 프래그먼트로 구성되고, 각 명령 프래그먼트는 특정 동작을 수행하기 위한 것임을 특징으로 하는 디지털 신호처리기를 제어하는 방법.

청구항 44

어드레스 공간을 갖는 메모리내에 저장된 명령을 사용하여 디지털 신호처리기를 제어하는 방법으로서,

제 1 명령의 제 1 부분을 포함하는 상기 메모리의 제 1 데이터 워드를 기입하는 단계; 및

상기 제 1 명령의 제 2 부분 및 제 2 명령의 제 1 부분을 포함하는 상기 메모리의 제 2 데이터 워드를 기입하는 단계를 포함하고,

상기 제 1 명령은 복수의 명령 프래그먼트로 구성되고, 각 명령 프래그먼트는 특정 동작을 수행하는 것을 특징으로 하는 디지털 신호처리기를 제어하는 방법.

청구항 45

제 43 항에 있어서,

상기 제 1 명령 및 상기 제 2 명령은 상이한 길이인 것을 특징으로 하는 디지털 신호처리기를 제어하는 방법.

청구항 46

삭제

청구항 47

제 43 항에 있어서,

상기 명령 프래그먼트는 제 1 명령 프래그먼트 및 제 2 명령 프래그먼트를 포함하고, 상기 제 1 명령 프래그먼트는 상기 제 2 명령 프래그먼트에 의해 수행된 일군의 동작의 서브세트인 일군의 동작을 요청하는 것을 특징으로 하는 디지털 신호처리기를 제어하는 방법.

청구항 48

제 47 항에 있어서,

상기 제 1 명령 프래그먼트는 상기 제 2 명령 프래그먼트보다 더 짧은 것을 특징으로 하는 디지털 신호처리기를 제어하는 방법.

청구항 49

삭제

청구항 50

삭제

청구항 51

삭제

청구항 52

삭제

청구항 53

삭제

청구항 54

삭제

청구항 55

삭제

청구항 56

삭제

청구항 57

삭제

청구항 58

삭제

청구항 59

삭제

청구항 60

삭제

청구항 61

삭제

청구항 62

삭제

청구항 63

삭제

청구항 64

삭제

청구항 65

삭제

청구항 66

삭제

청구항 67

삭제

청구항 68

삭제

청구항 69

삭제

청구항 70

삭제

청구항 71

삭제

청구항 72

삭제

청구항 73

삭제

청구항 74

삭제

청구항 75

삭제

청구항 76

삭제

청구항 77

삭제

청구항 78

삭제

청구항 79

삭제

청구항 80

삭제

청구항 81

삭제

청구항 82

삭제

청구항 83

삭제

청구항 84

삭제

명 세 서

기 술 분 야

[0001] 본 발명은 디지털 신호 처리기에 관한 것이다. 특히, 본 발명은 유일한 것은 아니지만, 고 병렬, 고 파이프라인 처리 기술을 이용한 디지털 신호 처리의 응용에 관한 것이다.

배 경 기 술

[0002] 일반적으로, 디지털 신호 처리기 (digital signal processor; 이하, DSP 라 칭함) 는 디지털 신호의 실시간 처리에 사용된다. 디지털 신호는 통상적으로 대응하는 아날로그 신호를 나타내는데 사용되는 일련번호 또는 디지털 값이다. DSP 는 소형 디스크 플레이어와 같은 오디오 시스템, 및 셀룰라 전화기와 같은 무선 통신 시스템을 포함하는 다양한 응용에 사용된다.

[0003] DSP 는 종종 마이크로프로세서의 특정화된 형상으로 고려된다. 마이크로프로세서와 마찬가지로, DSP는 실리콘 기재의 반도체 집적회로 상에서 일반적으로 구현된다. 또한, 마이크로프로세서와 같이, DSP의 계산능력은 축소명령세트 (reduced instruction set; RISC) 계산기술을 사용함으로써 증대된다. RISC 계산기술은 각각의 명령이 동일한 양의 시간에서 실행되는 DSP의 동작을 제어하기 위하여 다수의 같은 크기의 명령을 사용하는 것을 포함한다. RISC 계산기술의 사용은 명령이 수행되는 속도, 클럭속도를 증가시킬 뿐만 아니라, DSP 내의 명령 파이프라이닝 (pipelining) 의 양을 증가시킨다. 이것은 DSP의 전체 계산능력을 증대시킨다.

[0004] RISC 계산기술을 사용한 DSP 의 구성은 원치않는 특성도 발생시킨다. 특히, RISC 기반의 DSP는 다수의 명령을 실행하여 주어진 태스크를 수행한다. RISC 기반의 DSP 의 향상된 클럭속도에 의해 이들 명령을 실행하는 시간이 감소되더라도, 부가명령의 실행은 DSP 의 전력소모를 증가시킨다. 또한, 다수 명령의 사용은 DSP 내의 온-칩 (on-chip) 명령 메모리의 크기를 증가시킨다. 메모리 구성은 DSP 내의 실질적 회로 영역을 요구하여 (종종 전체영역의 50% 이상), DSP 의 크기 및 비용을 증가시킨다. 이와 같이, RISC 기반의 DSP 의 사용은 저비용, 저전력, 디지털 셀룰라 전화기 또는 다른 형태의 배터리 동작 무선 통신 시스템과 같은 응용에 대하여 이상적이지 못하다.

[0005] 도 1은 종래기술에 따라 구성된 디지털 신호 처리기의 간략화된 블록도이다. 산술논리장치 (arithmetic logic unit; ALU; 16) 가 ALU 레지스터 뱅크 (ALU resister bank; 17) 에 접속되고, 멀티플라이 어큐뮬레이트 회로 (multiply accumulate circuit; MAC; 26) 가 MAC 레지스터 뱅크 (27) 에 접속된다. 데이터 버스 (20) 는 MAC 레지스터 뱅크 (27), ALU 레지스터 (17), 및 (온-칩) 데이터 메모리 (10) 를 접속시킨다. 명령버스 (22) 는 MAC 레지스터 뱅크 (27), (온-칩) 명령 메모리 (12), MAC 레지스터 뱅크 (27), 및 ALU 레지스터 뱅크 (17) 를 접속시킨다. 명령 디코드 (18) 는 MAC (26) 및 ALU (16) 에 접속되며, 몇몇의 종래 기술 시스템에서는 명령 디코드 (18) 가 명령 메모리 (12) 에 직접 접속된다. 또한, 데이터 메모리 (10) 는 데이터 인터페이스 (11) 에 접속되며, 명령 메모리 (12) 는 명령 인터페이스 (13) 에 접속된다. 데이터 인터페이스 (12) 및 명령 인터페이스 (13) 는 오프-칩 (off-chip) 메모리 (6) 와 데이터 및 명령을 교환한다.

[0006] 동작시, 명령 메모리 (12) 에서의 명령이 명령 디코드 (18) 에 의해 디코드된다. 이에 대하여, 명령 디코드

(18) 는 ALU (16) 및 MAC (26) 에 가해지는 내부 제어신호를 발생한다. 제어신호들은 ALU 로 (16) 로 하여금 일반적으로 ALU 레지스터 뱅크 (17) 및 데이터 메모리 (10) 또는 명령 메모리 (12) 사이에서 교환된 데이터를 갖게 한다. 또한, 제어신호들은 ALU (16) 및 MAC (26) 로 하여금 ALU 레지스터 뱅크 (17) 및 MAC 레지스터 뱅크 (27) 각각에 저장된 데이터에 응답하고, 이 데이터 상에서 다양한 동작을 수행하게 한다.

[0007] 예시 동작에서, 명령 메모리 (12) 는 ALU (16) 및 MAC (26) 에 의한 사용을 위한 계수 데이터를 포함할 수 있고, 데이터 메모리 (10) 는 처리될 데이터 (신호 데이터) 를 포함할 수 있다. 계수 데이터는 공통실행인, DSP 를 사용하는 주파수 필터를 구현하기 위함이다. 필터링이 수행될 때, 데이터 메모리 (10) 로부터의 신호 데이터 및 명령 메모리 (12) 로부터의 계수 데이터 모두가 MAC 레지스터 (27) 에서 판독된다. 또한, 명령 메모리 (12) 내의 부가 명령 데이터는 데이터 버스 (22) 또는 직접접속을 통하여 명령 디코드 (18) 에 인가된다. 부가 명령 데이터는 MAC (26) 에 의해 수행되는 동작을 지정한다. 통상적으로, MAC (26) 에 의해 발생되는 결과는 다시 판독되어 데이터 메모리 (10) 로 입력된다.

[0008] 이러한 종래 기술의 처리로부터 많은 처리 비효율이 초래된다. 이러한 처리 비효율은, MAC 레지스터 (26) 및 명령 디코드 (18) 모두에 명령 데이터를 공급해야 하는, 명령 메모리 (12) 에 대한 예컨대, 버스 또는 액세스 경합 (contention) 을 포함할 뿐만 아니라, 신호 데이터를 판독하고 출력 데이터에 기록해야 하는 데이터 메모리 (10) 에 대한 버스 또는 액세스 경합을 포함한다. 또한, 많은 경우에서, 출력 데이터 상에서의 부가적인 처리가 ALU (16) 에 의해서 수행된다. 이것은 출력 데이터가 MAC 레지스터 뱅크 (27) 로부터 데이터 메모리 (10) 로 기입된 다음, ALU 레지스터 (17) 에서 판독되어야 하기 때문에, 데이터 메모리 (10) 에 대한 처리를 더욱더 악화시켜, 데이터 버스 (20) 에 대한 경합을 발생시킨다. 이러한 판독 및 기록 동작은 버스 (20) 상에서 수행되므로, 부가적인 버스 사이클을 소모한다. 이러한 비효율은 DSP 의 처리성능을 감소시킨다.

발명의 상세한 설명

[0009] 본 발명은 상술한 문제 및 비효율을 제기함으로써 DSP의 성능 및 유용성을 향상시킬 뿐만 아니라, 응용을 통하여 설명된 다른 특징 및 활용을 제공한다.

[0010] 본 발명은 새롭고 개선된 디지털 신호 처리방법 및 회로를 제공하는 데 목적이 있다.

[0011] 본 발명의 일태양에 따라, 메모리 및 처리 장치 사이의 가변 길이 데이터의 전송이 최적화되도록 선택되는 복수개의 제 2 선택가능 버스를 통하여 메모리가 복수개의 제 1 처리장치와 접속할 수 있는 디지털 신호 처리기가 제공된다.

[0012] 본 발명의 여러 다른 태양은 여기에 첨부된 청구범위에 정의된다.

[0013] 본 발명은 가변 길이 명령 집합의 사용에 의해 실현될 수 있다. 가변 길이 명령의 일부는 메모리 워드 경계 (memory word boundary) 에 걸쳐 발생하는 명령의 시작과 종료에 의해 메모리 공간 내의 인접한 위치에 저장될 수 있다. 본 발명의 부가적인 태양은 가변 수의 명령 프래그먼트를 포함하는 명령을 가지는 것에 의해 실현될 수 있다. 각각의 명령 프래그먼트는 특정 동작, 또는 동작들로 하여금 각각의 클럭 사이클 동안 다중 동작을 허용하여 수행되게 한다. 이와 같이, 다중 동작이 각각의 클럭 사이클 동안 수행되어, 태스크를 수행하는데 필요한 클럭 사이클의 전체수를 감소시킨다.

[0014] 예시의 DSP는 데이터가 레지스터 뱅크 및 3개의 데이터 메모리와 교환될 수 있는 일군의 3 개의 데이터 버스를 포함한다. 2 개 이상의 데이터 버스, 특히 3 개의 데이터 버스의 사용은, 버스 경합이 현저하게 감소되는 본 발명의 또 다른 태양을 실현한다. 본 발명의 일 실시예는 하나의 넓은 버스와 2 개의 좁은 버스를 포함하는 데이터 버스를 요구한다. 넓은 버스는 넓은 데이터 메모리에 접속되고, 2 개의 좁은 버스는 2 개의 좁은 데이터 메모리에 접속된다.

[0015] 본 발명의 실시예의 일태양은 적어도 2 개의 처리부에 의해 액세스할 수 있는 레지스터를 갖는 레지스터 뱅크의 사용이다. 이것은 메모리로의 데이터 판독 및 메모리로의 데이터 기록 없이, 다중 처리부에 의해 특정의 일군의 데이터 상에서 수행되는 다중 동작을 허용한다. 본 발명의 예시 실시예의 처리부는 산술논리장치 (ALU) 및 멀티플라이 어큐뮬레이트 (MAC) 장치를 포함한다. 다중 버스 구조의 사용, 고병렬 명령, 또는 이들 모두가 병합될 때, 본 발명의 부가적 태양은 고 파이프라인, 다중-동작, 처리가 수행되는 곳에서 실현된다.

[0016] 본 발명의 다른 태양은 명령 메모리에 저장된 가변 길이의 명령을 받는 명령 인출부를 포함하는 것에 의해 실현된다. 또한, 본 발명의 또 다른 태양은 상기 일군의 3 개의 데이터 메모리와 이격된 명령 메모리에 의해 실현된다. 명령 디코더는 제어신호로 하여금 명령 메모리로부터의 명령을 디코드하여, 다양한 레지스터, 데이

터 메모리, 및 각각의 클럭 사이클 동안 수행되는 다중 동작을 허용하는 기능부 사이에서 교환되는 데이터를 발생시킨다.

[0017] 본 발명의 다양한 태양은 상승적으로 병합하여, 예기치 못한 바람직한 결과를 제공한다. 예컨대, 메모리 내에 연속적으로 저장된 가변 길이 명령의 사용은 DSP 의 필수 회로 영역을 감소시킨다. 이러한 감소는 DSP 에 다중 데이터 버스를 부가하는 것을 용이하게 할 뿐만 아니라, 다중 처리부에 의해 액세스할 수 있는 레지스터의 부가를 용이하게 하여, DSP 의 전체 성능을 증가시킨다. 본 발명의 다양한 태양의 병합에 의해 제공되는 다른 상승적인 이익은 하기에서 명백해지며, 더욱더 상세하게 설명된다.

[0018] 본 발명의 상기 특징 및 또 다른 특징은 특히 첨부된 청구범위에서 설명되며 본 발명의 장점은 첨부된 도면을 참조하여 주어진 다음의 본 발명의 예시적인 실시예의 상세한 설명의 고려로부터 명백해질 것이다.

실 시 예

[0029] 본 발명은 신규하고 향상된 디지털 신호 처리 방법 및 회로이다. 응용 내내 다양한 신호, 명령 및 데이터에 대해 언급한다. 이들 신호, 명령 및 데이터는 대전, 광학 또는 자기 입자나 이들의 어떤 병합을 포함하는 전류축적, 전기적 전압, 전류에 의해 표시되는 것이 바람직하며, 이러한 사용은 공지되었다. 일반적으로, 이러한 신호, 명령 및 데이터를 나타내는 다양한 화학적 생물학적 합성물은 이러한 항목을 사용하고, 제어하고, 조종하는데의 어려움으로 인하여 바람직하지 않더라도, 본 발명의 사용과 마찬가지로 잘 일관된다.

[0030] 또한, 본 발명의 다양한 태양, 이익, 특징, 또는 장점이 언급된다 (특별하게 언급하지 않은 경우, 여기에서는 총괄하여 태양으로서 칭함). 본 발명의 몇몇 실시예에서는, 발명의 어떠한 다른 태양의 존재 없이도, 이러한 다른 태양들만이 실현될 수 있다. 그러나, 본 발명의 다른 실시예에서는, 2 이상의 발명의 태양이 함께 실현되어, 발명의 2 이상의 접속 태양의 단지 하나의 태양만을 실현하는 발명의 실시예에 의해 제공되는 것보다 큰 상승적이고 예기치 못한 장점을 유도한다.

[0031] **Ⅰ. DSP 동작 및 명령저장**

[0032] 도 2는 본 발명의 예시적인 실시예에 따라 구성된 디지털 신호 처리기 (DSP) 회로의 일부분에 대한 블록도이다. 데이터 메모리 (102~104) 가 어드레스 발생부 (address generation units; AGU; 105~107) 를 통하여 데이터 버스 (A, B, C) 에 각각 접속됨과 더불어 데이터 인터페이스 (100) 에 접속된다. 데이터 버스 (A, B, C) 는 멀티플렉서 (122~126)를 통하여 레지스터 뱅크 (120) 의 출력포트 (PO1, PO2, PO3) 에 각각 접속됨과 더불어, 레지스터 뱅크 (120) 의 입력포트 (PI1, PI2, PI3) 에 각각 접속된다. 바람직하게, 데이터 버스 (A, B, C) 는 데이터 메모리 (102~104) 와 레지스터 뱅크 (120) 내의 레지스터 사이에서 데이터를 판독하고 기록한다.

[0033] 3 개의 데이터 버스 및 3 개의 데이터 메모리의 사용은 버스 경합의 발생없이 레지스터 뱅크와 데이터 메모리 사이에서 교환되는 더 많은 데이터를 허용한다. 예컨대, 3 개의 인출동작은 3 개의 데이터 버스 (A, B, C) 를 사용하여 3 개의 메모리 (102~104) 로부터 동시에 수행될 수 있다. 유사하게, 3 개의 기록 동작도 수행될 수 있고, 이에 따라 3 개의 인출 및 기록 동작의 어떠한 병합도 수행될 수 있다.

[0034] 4 번째의 데이터 버스의 부가는 수행되는 동작의 더욱더 많은 수를 허용하며, 본 발명의 몇몇 실시예와 일관된다. 그러나, 3 개의 버스가 필터링과 같은 DSP 에 의해서 일반적으로 수행되는 많은 태스크들의 수행을 용이하게 하기 때문에, 3 개의 데이터 버스만을 사용하는 것은 특별한 장점을 갖는다. 이와 같이, 4 번째 데이터 버스의 부가는 3 번째 데이터 버스와 동일한 증가성능 향상을 제공하지는 않으며, 동일한 양의 부가적인 회로영역을 요구하지 않는다. 이에 따라, 4 번째 데이터 버스의 부가는 3 번째 버스의 부가보다는 적은 이익을 제공한다. 그래서, 본 발명의 많은 실시예에서는 단지 3 개의 데이터 버스만은 사용하는 것이 바람직하다.

[0035] 레지스터 뱅크 (120) 의 출력포트 (PO4, PO5, PO6) 가 멀티플라이 어큐뮬레이트 장치 (MAC; 128) 에 접속되며, MAC (128) 의 출력은 레지스터 뱅크 (120) 의 입력포트 (PI4) 에 차례로 접속된다. 레지스터 뱅크 (120) 의 출력포트 (PO7, PO8) 는 산술논리장치 (ALU; 130) 에 접속되고, ALU (130) 의 출력은 레지스터 뱅크 (120) 의 입력포트 (PI5) 에 접속된다.

[0036] 명령 메모리 (152) 는 명령 인출부 (156) 및 명령 인터페이스 (150) 에 접속된다. 명령 디코더 (158) 는 명령 인출부 (156) 에 접속됨과 더불어, 즉시 버스 (immediate bus) (ImALU) 뿐만 아니라 즉시 버스 (Im1), 즉시 버스 (Im2) 및 즉시 버스 (Im3) 에 접속된다. 즉시 버스 (Im1, Im2, Im3) 는 멀티플렉서 (122, 124, 126)

에 접속된다. 즉시 버스 (ImALU) 는 ALU (130) 에 접속된다. 상술한 데이터 접속에 부가하여, 디코더 (158) 가 제어 접속 (도면에 도시되지 않음) 에 의해 나타낸 여러 하위시스템에 접속된다.

[0037] 레지스터 뱅크 (120) 는 L0~L3 및 D0~D3 로 표시된 8 개의 레지스터를 포함한다. 레지스터 (L0~L3) 는 40 비트의 넓은 레지스터이며 또한, 고 워드 레지스터 (L0h~L3h) 및 저 워드 레지스터 (L01~L31)를 통하여 16 비트 프래그먼트에서 액세스될 수 있다. 레지스터 (D0~D3) 는 32 비트 와이드이며, 하위레지스터 (R0~R7) 를 통하여 16 비트 프래그먼트에서 액세스될 수 있다. 일반적으로, 레지스터 및 하위레지스터는 제공된 특정 레지스터 개수에 의해 명백해지는 레지스터의 특별한 특징을 갖는, 간단하게 "레지스터" 로 칭한다.

[0038] 발명의 일 태양은 다중 입력 및 출력 포트에 접속되어 이들에 의해 액세스될 수 있는 몇몇의 레지스터를 가지는 것에 의해 실현된다. 일 실시예에서, 이러한 다중 접속은 각각의 레지스터의 입력에 접속됨과 더불어, 각각의 출력포트에 접속된 멀티플렉서를 사용하는 것에 의해 제공된다. 다중접속을 제공하는 다른 방법이 명백해질 것이며, 예컨대, 데이터 버스 및 어드레스가능한 메모리의 사용을 포함하는 본 발명의 몇몇 태양의 사용과 일관된다. 그러나, 멀티플렉서는 여러 레지스터 및 포트에 빠르고 제어가능한 액세스를 제공하기 때문에, 멀티플렉서의 사용은 몇몇 실시예에서 바람직하다.

[0039] 본 발명의 다른 태양은 즉시 데이터 버스 (immediate data bus) 를 사용하는 발명의 실시예에서 실현되며, 이 실시예는 여기에서 제공되는 예시적인 실시예의 경우이다. 예컨대, 명령 데이터에 포함된 데이터는 메모리 (102~105) 와 인터페이스할 필요없이 레지스터 뱅크 (120) 로 판독될 수 있다. 이와 같이, 부가적인 데이터가 데이터 메모리와 인터페이스 없이 명령 처리 시스템으로부터 제공될 수 있어서, 버스 경합을 더욱 더 감소시킨다.

[0040] 도 3은 레지스터 뱅크 (120) 내의 레지스터 집합과 입력포트 (PI1~PI5) 집합 사이의 접속을 나타내는 블록도이다. 레지스터는 L0h~L3h, L01~L31, 및 R0~R7 로서 정의된다. 레지스터 (L0) 는 레지스터 (L0h, L01) 로 구성된다. 도 3 및 도 4의 관계에 있어서, 레지스터 (L0h~L3h) 는 24 비트이고, 레지스터 (L01~L31, R0~R7) 는 16 비트이어서, 레지스터 (L0~L3)를 40 비트 와이드로 만든다. 유사하게, 입력포트 (PI3~PI5) 는 총 40 비트에 대하여 24 비트의 입력포트 (PI3h~PI5h) 및 16 비트의 입력포트 (PI31~PI51) 로 구성된다. 입력포트 (PI1, PI2)는 단지 16 비트이며, 레지스터 (L0h~L3h)에 기록하는 데 사용되는 경우, 24 비트 중 적어도 중요한 16 비트에만 기록하는 것이 유용하다.

[0041] 도 3에 도시된 바와 같이, 몇몇의 레지스터는 모든 입력포트로부터 데이터를 받지만, 다른 레지스터는 입력포트의 단지 몇 개 또는 일부로부터 데이터를 받는다. 특히, 모든 레지스터 (L0~L3) 는 멀티플렉서 (500~514) 로부터 레지스터 (L0~L3) 내의 고 레지스터 및 저 레지스터 모두에 기록할 수 있는 16 비트 입력 포트에 의해, 모든 입력포트 (PI1~PI5) 로부터 데이터를 받는다. 이와 같이, 레지스터 (L0~L3) 는 (입력포트 PI0~PI3 에 대응하는) 소정의 버스 (A~C) 로부터 입력을 받음과 더불어 (입력포트 PI4 및 PI5 에 대응하는) MAC 부 (128) 및 ALU (130) 로부터 입력을 받는다. 레지스터 (R0~R7) 는 멀티플렉서 (516~530) 를 통하여 버스 (A~C) 로부터 입력 데이터를 받는다. 그러나, 레지스터 (R0~R7) 의 어떠한 레지스터도 MAC 부 (128; 입력포트 PI4) 로부터 입력 데이터를 받지 않는다. 또한, 레지스터 (R0~R3) 는 멀티플렉서 (516, 518, 524, 526) 를 통하여 ALU 부 (130) 로부터 입력 데이터를 받는다.

[0042] 도 3에 도시된 실시예는 여러 가지 장점을 갖는다. 특히, 이 실시예는 입력포트와 레지스터 사이에 충분한 접속성을 제공하여 대부분의 일반적인 동작을 용이하게 하지만, 전체 접속성은 최소로 유지되어 회로를 구현하는 데 요구되는 전체 회로영역을 감소시킨다. 예컨대, MAC 부 (128) 의 출력은 긴 레지스터 (L0~L3) 에만 접속된다. 이것은 멀티플라이 및 어큐뮬레이트 동작의 결과가 일반적으로 32 비트를 초과하기 때문에 이익이며, 그래서 레지스터 (D0~D3) 에 MAC 부 (128) 에 출력을 접속하는 것이 최소의 이익을 제공한다. 또 다른 예에서, ALU 부 (130) 는 레지스터 (L0~L3, R0~R3) 로 출력할 수 있다. 이것은 ALU 부 (130) 로부터의 데이터가 여러 레지스터에 기록될 수 있는 융통성을 증가시키며, ALU 부 (130) 가 많은 레지스터로 데이터를 출력하는데 유용한 많은 여러 동작을 수행하기 때문에 유용하다. 그러나, ALU 부 (130) 는 모든 레지스터에 접속되지 않으며, 이에 따라 불필요하고 과도한 접속성이 방지된다.

[0043] 도 4는 본 발명의 일 실시예에 따라 수행되는 경우 레지스터 뱅크 (120) 의 출력포트를 레지스터에 접속하는 것을 나타내는 블록도이다. 도시된 바와 같이, 버스 (A) 로 출력하는 출력포트 (PO1) 는 멀티플렉서 (540) 를 통하여, 하위 레지스터로서 액세스되는 경우 모든 유용한 레지스터를 포함하는 레지스터 (L0h~L3h, L01~L31, R0~R7) 에 접속된다. 유사하게, 버스 (B) 로 출력하는 출력포트 (PO2) 는 멀티플렉서를 통하여 레지스터 (L0h~L3h, L01~L31, R0~R7) 에 접속된다. 40 비트 와이드 버스 (C) 로 출력하는 출력포트 (PO3) 는 멀

티플렉서 (530) 에 의해, 완전한 레지스터로서 액세스되는 경우 모든 유용한 레지스터를 포함하는 레지스터 (L0~L3, D0~D3) 에 접속된다.

[0044] MAC 부 (128) 의 40-비트 입력으로 접속된 출력포트 (PO4) 는 멀티플렉서 (532)를 통하여 레지스터 (L0~L3) 로 접속된다. MAC (128) 에 누산된 값은 수행된 연산을 곱셈하고 누산하는 특성으로 인하여 크게 되기 때문에, 출력포트 (PO4) 를 40 비트의 "긴" 레지스터 (L0~L3) 로만 접속하는 것은, 다양한 곱셈 연산의 누산이 전형적으로 32 비트를 초과하게 되면, 레지스터 (D0~D3) 로의 추가의 접속을 제공함으로써 얻어진 유용성이 낮기 때문에 최적화 된 접속 배치를 제공한다.

[0045] MAC 부 (128) 의 하나의 16 비트 입력에 접속된 출력포트 (PO5) 는 멀티플렉서 (534) 에 의해 레지스터 (L0h~L3h, R0, R2, R4 및 R6) 로 접속된다. MAC 부 (128) 의 제 2 의 16 비트 입력포트로 접속된 출력포트 (PO6) 는 레지스터 (L0h~L3h, L01~L31 및 R0~R7) 로 접속된다. MAC 부 (128) 의 하나의 16 비트 입력을 모든 사용가능한 레지스터로 접속시킴으로써, 제 2 의 16 비트 입력포트를 사용가능한 레지스터의 서브세트로 접속시키면서도, 유용한 절충이 얻어진다. 특히, 레지스터 공간이 제한될 때, 처리되어야 하는 데이터의 적어도 한 부분이 어느 사용가능한 레지스터에 위치될 수 있다. 반면, 다른 입력에 접속되는 레지스터의 수를 제한함으로써, 접속 회로의 총량이 줄어들며, 이는 다른 레지스터, 입력포트, 및 출력포트 사이의 연결성을 높이는 등의 또 다른 기능과 특성을 제공하게 한다.

[0046] ALU (130) 의 입력으로 접속된 출력포트 (PO7) 는 멀티플렉서 (546) 에 의해 레지스터 (L0~L3, L0h~L3h, 및 R0~R3) 로 접속되는데, 여기서 L0h~L03h 및 R0~R3 은 논리 0 의 집합과 연관하여 출력된다. 즉, 레지스터 (L0h~L03h 및 R0~R3) 는 PO7 비트의 비트 31~16 (0~39 의 번호의 비트) 으로 출력되는데, 비트 0~15 는 논리 0 으로, 비트 39~32 는 비트 31 을 사용하여 부호 확장된다. ALU (130) 의 또 다른 입력으로 접속된 출력포트 (PO8) 는 멀티플렉서 (548) 에 의해 일련의 논리 0 과 연관하여 레지스터 (L0~L3), 및 레지스터 (R0~R7) 로 접속된다. 이러한 방식으로 ALU(130) 의 입력을 접속함으로써 모든 사용가능한 긴 레지스터 (L0~L3) 에 대하여, 따라서, 큰 수에 대한 논리 연산 수행이 가능하도록 하여, 평균화 (normalizing) 및 스케일링 (scaling) 과 같은 유용한 많은 유형의 신호처리연산에서 유용하도록 한다. 또한, 레지스터 (R0~R7 및 L0h~L3h 및 R0~R7) 사이에서 산술연산이 수행될 수 있어, 필요한 접속수를 제한하여 이에 의해 필요한 회로면적을 제한하면서도, 사용할 수 있는 레지스터 집합의 면에서 높은 수준의 유동성을 제공한다. 이해해야 할 것은, 가능한 논리 및 산술 연산은 전술한 것에 제한되지 않는다는 것이다.

[0047] 다중 데이터 버스와 다중 처리부 모두에 의해 액세스 될 수 있는 레지스터를 사용함으로써, 다양한 이점을 얻을 수 있다. 예컨대, 레지스터는 데이터 버스와 처리부 사이에 인터페이스를 제공하여, 각 데이터 버스를 각 처리부로 라우트 할 필요성을 감소시킨다. 데이터버스 라우팅을 줄임으로써, 회로면적에 여유를 주고 칩가격을 낮춘다.

[0048] 더욱이, 적어도 몇몇 (일군)의 레지스터를 다중 처리부로 접속함으로써, 데이터 버스 상의 데이터를 판독하고 메모리에 기입할 필요없이, 다중 처리부를 사용하여 동일한 데이터상에서 다중 연산이 수행되도록 한다. 이것은 버스 사이클에 여유를 주어, 버스 경합을 감소시킨다. 제 1 명령 사이클 동안 제 1 처리부에 의해 처리되는 데이터는 동일한 레지스터내에서 바로 제 2 처리 사이클 동안 제 2 처리부에 의해 처리될 수 있으므로, 명령 처리 파이프라이닝 (pipelining) 이 또한 채용될 수 있다.

[0049] 반면, 전형적으로 모든 데이터가 다중 처리부에 의해 처리될 필요는 없기 때문에, 다른 레지스터 (레지스터 집합) 는 하나의 처리부에 의해 또는 두 개보다 많은 처리부가 존재하는 경우에는 처리부의 총 수보다 작은 처리부에 의해 액세스 될 수 있다. 이러한 다른 레지스터 집합을 사용함으로써 접속의 수을 줄이고, 회로면적으로 줄여, 레지스터 접속성과 회로면적 사이의 (따라서, 성능과 효율성 사이의) 최적화된 균형이 제공된다.

[0050] 더욱이, 본 발명의 일 실시예에서는, 레지스터 뱅크 (128) 내에 2-위상 클록 레지스터를 사용함으로써 파이프라이닝이 더욱 향상된다. 2-위상 클록 레지스터는 동일한 전 클록 ("처리") 사이클내에서, 클록의 제 1 위상에서 판독된 다음, 클록의 제 2 위상에서 기입된다. 따라서, 특정의 처리 사이클중, MAC (128) 등의 제 1 처리부에 의해 이미 처리된 데이터는 제 1 클록 위상중 판독된 다음, 처리 사이클의 나머지 부분 내에 ALU (130) 등의 제 2 처리부에 의해 처리될 수 있다.

[0051] 또한, 처리 사이클의 제 2 위상중, MAC (128) 에 의해 바로 처리된 새로운 데이터가 동일한 레지스터로 기입되어, 하나의 처리 사이클동안 2 개의 처리부 사이의 완벽한 파이프라인 처리를 가능하게 한다. 다시 말하면, 이러한 동작은 어느 내부 버스에 대하여도 데이터를 이동하지 않고 수행되므로, 버스 경합의 증가를 막는다.

[0052] 도 2 를 참조하면, 연산중, 명령 인출부 (156) 는 명령 메모리 (152) 로부터, 또는 명령 메모리 (152) 내에서 없으면 외부에 위치한 메모리로부터 2 진 명령을 검색한다. 외부 메모리는 종래 기술에서 잘 알려진 바와 같이, DRAM 및 SRAM 또는 이를 활용한 장치, 자기 또는 광 하드디스크 메모리 또는 공지된 다른 데이터 기억매체 일 수 있다.

[0053] 본 발명의 실시예에 따르면, 명령은 가변길이형이며, 명령 인출부는 명령의 길이와 각 처리 또는 클록 사이클중 인출되어야 할 추가의 명령량을 결정한다. 또한, 명령은 내부 메모리 및 외부 메모리내의 연속적인 메모리 위치내에 저장된다. 이하, 인출부 (156) 와 메모리 및 외부 메모리 내의 명령 데이터의 동작을 설명한다.

[0054] 명령 디코더 (158) 는 명령 인출부 (156) 에 의해 검색된 명령을 수신하여, 명령을 데이터 메모리, 레지스터 뱅크, MAC 및 ALU를 포함한 DSP를 구성하는 하나 이상의 하부시스템으로 인가되는 제어신호로 변환한다. 또한, 명령 디코더 (158) 는 수신 명령내에 포함된 즉시 데이터를 라우트하여, 즉시 버스 (Im1, Im2, Im3, 또는 ImALU) 를 통하여 적절한 시스템으로 라우트시킬 수 있다. 즉시 데이터는 전형적으로 데이터 메모리 (102 ~ 106∏) 내에 저장된 데이터 상에서 연산을 수행하거나, 어드레스를 지정 또는 변경하는 데 사용되는 명령 데이터내에 저장된 산술값이다.

[0055] 도 2 의 DSP 에 의해 수행되는 연산은 데이터 버스중 하나를 통하여 데이터 메모리로부터 레지스터 위치로 데이터를 로드하는 것을 포함한다. 또한, MAC (128) 또는 ALU (130) 는 레지스터 뱅크 (120) 내의 하나 이상의 레지스터 내에 저장된 데이터에 대하여 연산을 수행할 수 있으며, 그 결과는 전형적으로 레지스터 뱅크 (120) 내의 레지스터로 재기입된다.

[0056] 전술한 DSP 아키텍쳐는 많은 이점을 제공한다. 예컨데, 3 개의 데이터버스를 사용함으로써 데이터의 파이프라인 처리가 인터럽트 되지 않게 한다. 예를 들어, DSP 에 의해 수행되는 필터링 중, 필터링되어야 할 데이터 (신호 데이터) 는 하나의 데이터 메모리에 저장되며, 데이터로 인가되어야할 계수는 다른 데이터 메모리에 저장된다. 연산의 결과는 전형적으로 피연산자보다 많은 비트수를 요구하므로, 두 개의 좁은 메모리에 신호 데이터와 계수 데이터를 저장하는 것이 바람직하다. 다음, 계수 및 신호 데이터는 레지스터 뱅크 (120) 로 판독되어, MAC 부 (128) 에 의해 곱셈되고, 누산된다. 이러한 연산의 결과는 레지스터 뱅크 (120) 내의 제 2 레지스터에 저장되거나, 입력 데이터가 이미 저장된 레지스터 뱅크로 중복 기입될 수 있다. 어떤 결과라도 전형적으로 제 3 버스 (BUS C) 의 레지스터에서 더 넓은 (메모리 C) 데이터 메모리로 기입된다.

[0057] 출력 데이터는 제 3 버스를 통하여 제 3 메모리로 기입되고, 입력 데이터 집합은 제 1 및 제 2 데이터 버스를 통하여 제 1 및 제 2 데이터 메모리로부터 판독되므로, 메모리 액세스 충돌 또는 버스 경합이 거의 발생하지 않는다. 따라서, 데이터의 처리는 인터럽트 되지 않고 진행되어, 내부 버스 또는 기타 하부시스템 보다 고속도로 어떤 메모리 시스템 또는 데이터 버스를 클록시킬 필요성을 감소시킨다. 이것은 처리 속도를 유지 또는 증가시키면서도, 전력소비를 감소시킨다.

[0058] 또한, 인터럽트 없이 DSP를 통하는 데이터를 처리함으로써, 주어진 시간에 다수의 다른 데이터 값들이 DSP 내의 서로 다른 스테이지에서 따로 처리되는 데이터의 파이프라인 처리를 가능하게 한다. 또한, 전술한 바와 같이 병렬 명령의 사용이 병행될 때, 이러한 고 효율의 파이프라인 처리와 연관하여 상당한 처리 유동성을 얻을 수 있으며, 따라서, 높은 다목적성, 효율성을 갖는 파워풀한 DSP 시스템을 제공할 수 있다.

[0059] 이해해야할 것은, 다중 버스의 사용은 버스 충돌을 줄이는 다양한 추가의 방법으로 DSP 주위로 데이터를 이동시키는 능력을 향상시킨다는 것이다. 예컨데, 나눗셈 되어야 할 데이터가 데이터 버스 C를 통하여 메모리 C 로부터, 메모리 A 및 데이터 버스 A 등의 또 다른 메모리 및 버스에 의해 제공되는 제수와 함께, 입력 데이터로 인가될 수 있다. 그 결과는 나머지 버스 (데이터 버스 B)를 통하여 나머지 메모리 (메모리 B) 에 저장될 수 있다.

[0060] 예를 들어, 다중 데이터 버스 및 메모리를 제공함으로써 가능한 연산에서는, MAC 부 (130) 에 의해 누산되어야 할 데이터가 제 1 메모리 및 제 1 버스 (예컨데, 메모리 A 및 데이터 버스 A)를 통하여 제공된다. 일련의 누산 연산이 수행된 후, 그 결과의 데이터가 데이터 버스 (C)를 통하여 메모리 (C) 로 기입될 수 있다. 동시에, 논리적으로 시프트되어야 할 데이터가, 데이터 버스 C 가 MAC 부 (128) 로부터 결과 데이터를 이송하지 않는 처리 사이클 중에, 데이터 버스 (C)를 통하여 메모리 C 로부터 ALU 부 (130) 로 제공되며, 이는 일련의 누산 연산이 수행된 후 그 결과 데이터가 가용화되는 대부분의 시간이 된다. 논리적으로 시프트 된 데이터는 데이터 버스 B를 통하여 메모리 B 로 동시에 기입된다. 따라서, 다중 데이터 버스 및 메모리를, 일반적으로, 특히 다중 처리부와 병행하여 사용함으로써, DSP 내의 데이터 이동 가능성을 더 많이 제공하여 다중 연산을

수행 가능하게 한다.

[0061] 전술한 바와 같이, 본 발명의 다른 이용예는 다중 처리부, 예컨데, MAC 부 (128) 및 ALU 부 (130) 에 의해 액세스 가능한 레지스터를 사용함으로써 실현된다. 다중 처리부에 의해 액세스 가능한 레지스터는 처리부에 의해 처리되어야 할 데이터가 어떤 내부 데이터 버스도 경유하여 데이터를 이동시키지 않고도, 액세스될 수 있게 한다. 예컨대, 제 1 처리부에 의해 데이터가 레지스터로 기입된 다음, 그 레지스터를 액세스 할 수 있는 제 2 처리부에 의해 더 처리될 수 있다. 이것은 버스 경합과 충돌을 완화시켜 데이터 처리량을 향상시킨다.

[0062] 또한, 이하 상술되는 바와 같이, 병렬 연산 명령 및 병렬 처리 기능과 병행될 때, 고효율로 데이터를 파이프라인, 다중연산, 및 데이터 처리하는 능력이 더욱 향상된다. 이와 비교하여, 전형적인 파이프라이닝은 일련의 연산의 서로 다른 위상 (즉, 인출, 디코드, 처리)을 스태거링하여, 각 명령의 시작간의 처리시간이 감소된다. 다중연산 파이프라이닝은 데이터가 일련의 다른 연산을 통과하도록 하는 추가의 이점을 제공하며, 여기서 상기 연산은 다른 데이터 집합에 대하여 동시에 수행된다. 이러한 다중 연산 파이프라이닝은 종래의 명령 파이프라이닝에 비해 사이클당 수행되는 명령의 수를 더욱 증가시킨다.

[0063] 이하, 전술한 아키텍처의 시너지 효과를 다음의 처리예를 통하여 나타낸다. 필터링 동작을 예로 들면 (전술한 바와 같은), MAC (26) 에 의해 발생된 결과는, 누산이 수행된 산물의 수가 증가하는 만큼 크기 (값을 나타내는데 사용되는 비트 수 및 절대값의 면에서) 가 증가한다. 결국, 결과는 스케일 되거나 "정규화" 되며, 이는 전형적으로 ALU 부 (130) 에 의한 논리 시프트 연산을 요구한다.

[0064] 전술한 시스템에서, 스케일링 동작은 필터링의 멀티플라이 및 어큐뮬레이트 연산과 동시에 수행될 수 있다. 상기 동시 처리가 수행되는 처리 사이클중, 처리되지 않은 신호 데이터 및 필터링 계수는 데이터 메모리 (102 및 103) 로부터 판독되어, 레지스터 뱅크 (120) 내의 레지스터 (예컨대, L0h 및 L01) 로 간다. 동시에, MAC 부 (128) 는 상기 레지스터 (L0h 및 L01) 에 미리 저장된 값을 판독하여, 멀티플라이 및 어큐뮬레이트 연산을 수행하며, 그 출력은 제 2 레지스터 (예컨대, L1) 로 기입된다. 또한, 동시에, ALU 부 (130) 는 제 2 레지스터 (L1) 에 미리 저장된 데이터를 판독하여, 스케일 연산을 수행하여, 스케일 된 값을 제 3 레지스터 (예컨대, L2) 로 기입한다. 또한, 동일한 처리 사이클중, 제 3 레지스터 (D0) 에 미리 저장된 값은 버스 C (112)를 사용하여 데이터 메모리 (104) 로 기입된다. 전술한 바에서 알 수 있듯이, 특정한 연산은 수행되는 특정의 작업에 따라 변할 수 있다. 전술한 바에서 알 수 있듯이, 다중연산이 더욱더 고효율로 수행되도록 하는 병렬 명령을 사용함으로써 더욱 더 고효율의 파이프라인 다중명령 연산이 가능하게 된다. 고효율의 병렬 명령은 각 처리 사이클중 파이프라인 되어야할 연산을 다르게 지정할 수 있도록 한다.

[0065] 전술한 예와 같은, 2 위상 판독-기입 연산을 사용함으로써, 그 처리가 하나의 처리 사이클중에 모두 수행될 수 있으며, 여기서 데이터는 제 1 클록 위상중 각 레지스터로부터 판독되어, 처리부에 의해 처리되고, 제 2 클록 위상중 그 결과는 구 데이터에 대한 레지스터로 기입된다. 이해해야할 것은, 단일 클록 사이클중 이러한 처리의 모든 단계에 동일한 값이 해당되는 것은 아니며, DSP 를 통해 일련의 값들이 파이프라인 되어, 처리가 수행됨에 따라 다음 단계로 각각 이동한다는 것이다.

[0066] 이하 설명된 바와 같이, 본 발명의 다양한 변형예에 의해 많은 다른 연산이 가능하다. 예를 들어, 보코딩 (vocoding) 은 음성 데이터를 코드화하는 처리이다. 보코딩은 수행되어야 할 많은 다른 형태의 수행 연산을 요구하는데, 그 중 어떤 것은 독립적으로 수행될 수 있으며, 따라서 동시에 수행될 수 있다. 다중 데이터 버스 및 다중 처리부를 사용함으로써 이러한 연산을 수행하는 것이 가능하게 된다.

[0067] 분리형의 명령 메모리 및 명령 디코더를 사용함으로써 더욱 더 이점이 제공된다. 예를 들어, 전술한 데이터 처리와 동시에, 명령 인출 (156) 에 의해 명령 메모리 (152) 로부터 명령이 판독되고, 이어서, 명령 디코더 (158) 가 DSP 내의 다른 많은 하부시스템의 연산을 제어하는 제어신호를 발생시킨다. 다시 말하면, 데이터 버스가 명령데이터를 이송할 필요는 없으며, 따라서, 신호 데이터는 명령 데이터로부터 인터럽트 없이 이송 및 처리 될 수 있다. 따라서, 데이터 처리로부터 명령 처리를 분리함으로써 성능이 더욱 더 향상되며, 이는 명령 데이터 이동에 대한 데이터 버스 사이클의 소모 필요성을 제거한다.

[0068] 도 5 는 본 발명의 일 실시예에 따른 도 2 의 명령 메모리 (152) 의 어드레싱 가능한 메모리 공간의 일부분내의 일련의 가변길이명령의 패킹을 나타낸다. 본 발명의 몇몇 실시예에서, 가변길이명령이 도 2 에 도시된 바와 같이 외부 메모리 시스템내에 저장되어 부가의 메모리 효율성을 실현할 수 있다. 메모리 (275) 의 중간 및 우측 칼럼에 도시된, 32 비트 워드를 지정한 각 어드레스와 함께, 본 실시예의 어드레스가 좌측 컬럼에 도시되었다. 각 데이터 워드의 중간 칼럼은 16 비트 고순위 서브워드를 나타내며, 맨 우측 칼럼은 16 비트 하순위

서브워드를 나타낸다. 본 발명의 바람직한 실시예에 있어서는, 필요한 어드레스 논리의 량을 줄이기 위하여, 고순위 및 하순위의 서브워드는 개별적으로 어드레싱 할 수 있지는 않다.

[0069] 메모리 (275) 내에서, 가변길이명령 (A~L) 은 도시된 패킹 형태로 저장된다. 명령 A 는 어드레스 워드 (0x0000) 에 저장된 제 1 의 2 개의 더블-바이트 (A(1) 및 A(2)) 및 어드레스 (0x0001) 의 고순위 서브워드에 저장된 제 3 의 더블-바이트 (A(3))를 갖는 48 비트 명령이다. 명령 A 에 이은, 명령 B 는 어드레스 (0x0001) 의 하순위 워드에 저장된 제 1 의 더블-바이트 (B(1)) 및 어드레스 (0x0002) 의 고순위 서브워드에 저장된 제 2 의 더블-바이트 (B(2))를 갖는 32 비트 명령이다. 명령 C 는 어드레스 (0x0002) 의 하순위 서브워드에 저장된 제 1 의 유일한 더블-바이트 (C(1))를 갖는 16-비트 명령이다.

[0070] 명령 A~C 의 저장위치에서 명백한 바와 같이, 본 발명은 동일한 어드레스 워드내에 다른 명령 부분들을 저장함으로써 일련의 명령을 저장하는데 필요한 메모리 (275) 크기나 양을 줄인다. 예컨대, 명령 A 의 제 3 의 더블-바이트 (A(3)) 는 명령 B 의 제 1 의 더블-바이트 (B(1)) 와 함께 저장된다.

[0071] 워드 경계를 넘어, 또는 더 자세하게는 메모리 어드레스 공간내에 연속된 위치내에, 가변길이명령을 저장함으로써, 본 발명은 주어진 명령의 수를 저장하는데 요구되는 명령 메모리의 양을 감소시킨다. 명령 메모리의 양을 줄임으로써, 주어진 명령 캐쉬 용량을 DSP 에 부여하는데 필요한 틀 (die) 의 크기와 용량을 감소시킨다. 도 3 에 도시된 바와 같이, 메모리 (275) 내에 가변길이명령 (D~L) 을 더 위치시킴으로써, 명령의 패킹이 더 표현된다.

[0072] 이해해야할 것은, 전술한 바와 같이 연속된 위치에 모든 명령을 패킹하는 것은 본 발명의 몇몇 실시예에서는 필요치 않다는 것이다. 예컨데, 본 발명의 다른 실시예에서는 메모리 공간내의 연속된 위치에 명령의 본질적인 부분 (예컨데, 약 90% 이상) 만을 패킹한다. 본 발명의 다른 실시예에서는, 바람직하게도, 명령의 주요 부분 (25 ~ 50 %) 만이 연속된 메모리 공간내에 패킹된다. 본 발명의 다른 실시예에서는 또 다른 퍼센티지 패킹된 명령을 사용할 수 있다.

[0073] 또한, 연속된 위치의 사용이 필요치는 않다. 명령은, 명령 데이터의 총량보다 기본적으로 크지 않는 총 메모리 공간내에 쉽게 위치되어야 한다. 이것은 메모리 공간내 인접한 위치에 명령을 위치시킴으로써 얻어지는 것이 바람직하지만, 명령이 의도된 실행 순서로 읽혀질 수 있는 한, 메모리 공간 전체를 통해 혼합배분 (suffled) 될 수 있다. 당업자라면, 메모리 공간의 재매핑 (remapping) 과 같은 이러한 유형의 소정의 혼합배분 (suffling) 은 바람직하지 않은 복잡함을 부가하지 않으며, 본 발명의 연산에 영향을 주지 않는다는 것을 인식할 것이다.

[0074] 유사하게는, 큰 명령 집합에 대하여 패킹 구조가 채용되도록하는 것이 바람직하다. 예컨대, 적어도 열 개의 명령에 대하여 패킹 구조를 채용하는 것이 본 발명의 몇몇 실시예에서 바람직하다.

[0075] 또한, 본 발명의 실시예에 사용된 특정의 패킹 구조는 본 발명의 몇몇 다른 실시예에서는 필요치 않다. 예컨데, 본 발명의 몇몇 다른 실시예는 연속된 메모리 위치내에 명령을 가지지 않을 수 있다. 더욱이, 명령은 명령 분리기 코드 (instruction seperator code) 의 사용을 포함하여, 몇몇 소량의 메모리 공간에 의해 분리될 수 있다. 소량의 메모리 공간은 메모리 워드 경계상의 명령 경계를 유지하는데 필요한 메모리 공간량보다 작은 것이 바람직하다. 전술한 패킹은 그 간소함, 완전성 및 효율성으로 인하여, 많은 예에서 선호된다. 일반적으로, 패킹의 완전성과 패킹구조의 복잡함 사이의 트레이드오프의 선택은 본 발명의 각종 실시예에서 다를 수 있다.

[0076] 또한, 전술한 바와 같이, 본 발명의 몇몇 실시예는 전체 가용한 일련의 명령에 대해서가 아니라, 명령의 일부분에 대하여만 패킹구조를 채용한다. 예컨대, 특정의 작업 또는 서브루틴을 수행하는데 필요한 일련의 명령에 대해서만 명령 패킹이 수행될 수 있다.

[0077] 주지해야할 것은, 가변길이명령은 요구되는 연산을 요청하는데 필요한 데이터량만을 소모하며, 고효율로 패킹된 명령 저장장치는 가변길이명령의 집합에 의해 소모되는 것과 같도록, 따라서 최소로 총 메모리를 유지하므로, 가변길이명령을 사용하여 고효율로 패킹된 명령 저장장치를 병행시킴으로써, DSP 의 메모리 요구량을 줄일 수 있다는 것이다.

[0078] 감소된 DSP 크기 및 이에 의한 DSP 가격절감의 이점에 더하여, 고효율로 패킹된 명령 및 가변길이명령은 전술한 아키텍처의 다른 특징과 병행될 때 뜻하지 않던 추가의 이점을 제공한다. 예컨데, 명령 메모리의 크기를 줄임으로써, DSP 내의 3 개 데이터 버스의 사용을 위한 추가의 회로 면적이 사용가능하며, 이것은 인터럽트 되지 않고, 고효율로 파이프라인된 데이터 처리 및 DSP 내에서 동시에 다중 연산을 수행하는 능력을 포함한 전술

한 이점을 제공한다. 따라서, 조밀하게 패킹된 명령은 다중 버스 아키텍처와 병행하여, 추가의 뜻하지 않은 이점을 제공하여, 성능 및 효율의 향상을 가져온다.

[0079] 메모리 공간의 연속된 위치내에 가변길이명령을 저장하는 능력은 그러한 구성에 저장된 가변길이명령을 인출하고 처리할 수 있는 DSP를 제공함으로써 가능해 질 수 있다. 도 6 은 본 발명의 일실시예에 따라 명령 메모리 (152) 로부터 명령을 인출할 때 명령 인출부 (156) 의 연산의 흐름도이다. 단계 (200)에서 처리가 시작되며, 단계 (202)에서 제 1 명령 데이터 집합이 명령 메모리 (152) 로부터 판독된다. 본 발명의 실시예에서, 2 개의 32-비트 워드, 또는 64 비트의 명령 데이터가 단계 (202) 에서 검색된다.

[0080] 단계 (204)에서, 검색된 명령 데이터중 64 비트에 포함된 제 1 명령이 명령 디코더 (158) 에 의해 처리된다. 본 발명의 실시예에서, 명령은 16, 32, 또는 48 비트의 길이일 수 있다. 명령 길이는 이하 상술되는 바와 같이 명령 길이를 나타내는 각 명령에 포함된 일련의 헤더 비트에 의해 결정된다. 분명하게는, 2 개의 명령의 구분하여 분리하는 코드의 사용, 또는 이어 오는 몇몇 명령 집합의 길이를 지정하도록 하는 수퍼 헤더 명령의 사용을 포함한, 명령 길이를 지정하기 위한 각종 다른 방법이 있다. 헤더 비트의 사용은, 명령 길이 정보가 명령에 근접하게 유지되어, 명령 처리에 관한 상태 정보를 저장 또는 유지할 필요를 줄이므로, 몇몇 예에서 선호된다.

[0081] 검색된 명령 데이터의 64 비트내에 포함된 제 1 명령이 처리된 후, 단계 206에서, 48 비트 또는 그 이상의 처리되지 않은 명령 데이터가 64 비트의 검색된 명령 데이터에 남아있는지 결정된다. 48 비트 또는 그 이상의 처리되지 않은 명령 데이터가 남아있다면, 나머지 48 비트의 처리되지 않은 데이터 내에 포함된 다음 명령이 단계 204에서 다시 처리된다.

[0082] 단계 (206)에서, 48 비트 이하의 처리되지 않은 데이터 명령이 검색된 명령 데이터에 남아있다면, 명령 메모리 (152) 로부터 추가의 명령 데이터가 로드된다. 추가의 명령을 로드하기 위한 다양한 방법이 고려된다. 본 발명의 실시예에서는, 명령 메모리로부터 충분한 추가의 명령 데이터가 로드되어, 명령 인출부에 저장된 처리되지 않는 데이터 량을 48 비트로 리턴한다. 48 비트의 처리되지 않은 데이터가 명령 인출부내에 저장되도록 함으로써, 적어도 하나의 완전한 명령이 명령 디코더 (158) 에서 사용가능 하도록 보장한다.

[0083] 본 발명의 바람직한 실시예에서, 명령 인출 부는 처리된 특정 양의 데이타에 따라서 48 비트 미만의 처리되지 않은 데이타가 남아 있을 때, 가변량의 데이타를 검색한다. 특히, 처리된 데이타의 양이 데이타 워드 (32 비트) 와 같거나 크다면, 새로운 명령 데이타의 추가적인 데이타 워드 (32 비트) 가 검색된다. 만약, 미리 처리된 데이타의 양이 2 개의 데이타 워드 (64) 비트와 같거나 크면, 2 개의 새로운 데이타 워드가 명령 인출 부에 의해서 검색된다.

[0084] 처리된 데이타의 워드 수에 기초하여 검색된 데이타의 양을 결정하는 것은, 그것이 충분한 양의 처리되지 않은 데이타를 명령 디코더 (158) 에 유용하도록 하는 한편, 더 효율적인 워드-길이 (word-length) 접근을 명령 메모리를 구성하는 메모리 뱅크에 허용하기 때문에, 바람직한 것이다. 일단, 추가적인 처리되지 않은 명령 데이타가 단계 (206) 에서 검색되면, 다음 명령이 현재 가용한 처리되지 않은 명령 데이타 전체 내에서 처리된다.

[0085] 도 7 은 본 발명의 일실시예에 따라 구성된 명령 인출 부 (156) 및 명령 메모리 (152) 의 블록도이다. 명령 메모리 (152) 는 짝수 메모리 뱅크 (302; RAM0) 및 홀수 메모리 뱅크 (300; RAM1) 으로 구성되며, 이들 각각은 32 비트 데이타 워드를 판독하고 기입한다. 메모리 뱅크는 홀수와 짝수로 라벨링되는데, 이것은 그 양자 모두가 동일한 어드레스 공간 내에서 어드레싱되지만 짝수 어드레스는 짝수 메모리 뱅크 (302) 로 향하고 홀수 어드레스는 홀수 메모리 뱅크 (300) 로 향하기 때문이다.

[0086] 8, 16, 24, 48 및 64 비트 워드를 포함한 다른 워드 사이즈를 판독하고 기입하는 메모리 뱅크는 본 발명의 또다른 실시예에서 사용될 수 있다. 추가적으로 다른 수의 메모리 뱅크가, 1 -8 메모리 뱅크를 포함해서, 사용될 수 있다. 그러나, 32 비트 워드로 된 2 개의 메모리 뱅크의 사용이 바람직한데, 이것은 전체적인 복잡성을 감소시키는 한편, 명령 데이타가 제어하기 쉬운 양으로 어드레싱되도록 허용하기때문이다.

[0087] 제어 논리 (304) 는 데이타 워드가 메모리 뱅크 (300 및 302) 로부터 명령 레지스터 (106 및 107) 로 판독되게 한다. 판독된 특정한 기억 장소가 어드레스 선 (310 및 314) 에 의해 특정되고, 명령의 판독이 이네이블 선 (enable lines; 332, 315, 316 및 318) 에 의해 제어된다. 명령 레지스터 (306 및 307) 의 32 비트 출력은 16 비트 부분으로 로테이터 (308) 의 입력 (A, B, C, 및 D) 으로 인가된다. 로테이터 (308) 는 48 비트의 명령 데이타 (324) 를 출력한다. 48 비트의 명령 데이타 (324) 는, 이하에서 더 자세히 설명하는 바와 같이, 각 입력 세트가 16 비트를 포함하고 있는 4 개의 입력(A, B, C 및 D) 중의 3 개 입력 (3:4) 으로 구성된다.

[0088] 동작하는 동안, 제어 논리 (304) 는 도 5 를 참조하여 설명한 방법에 따라 명령 메모리 뱅크 (300 및 302) 로부터 명령 데이타를 로드 (load) 한다. 특히, 제어 논리 (304) 는 우선, 홀수 메모리 뱅크 (302) 및 짝수 메모리 뱅크 (300) 양자 모두로부터의 32 비트 데이타 워드를 판독함으로써 64 비트의 처리되지 않은 명령 데이타 전체를 명령 레지스터 (306 및 307) 로 로드한다. 16 비트 명령이 처리된다면, 명령 레지스터 (306 및 307) 가 아직도 48 비트의 처리되지않은 명령 데이타를 포함하기 때문에, 새로운 데이타는 로드되지 않는다. 32 비트 명령이 처리된다면, 48 비트 미만의 처리되지 않은 명령 데이타가 남아있기 때문에, 명령 레지스터 (306) 는 32 비트 워드의 추가적인 명령 데이타로 로드된다. 일단 다시, 32 비트 명령 워드를 로드하는 것은, 16 비트의 처리되지 않은 것은 레이스터 (307) 로, 다음의 32 비트는 레지스터 (306) 로 위치시켜면서, 48 비트의 처리되지 않은 명령 데이타를 레지스터 (306 및 307) 에 위치시킨다.

[0089] 그 다음, 48 비트 명령이 처리되면, 처리되지 않은 명령 데이타는 남아있지 않게 되어, 레지스터 (306 및 307) 양자 모두는 32 비트 워드의 명령 데이타로 로드되며, 그것은 64 비트의 처리되지 않은 명령 데이타이며, 명령 데이타의 48 비트보다 더 큰 것이다. 64 비트의 명령 데이타 모두를 로드하는 것이 특별히 필요한 것이 아니지만, 그것은 유용한데, 이유는 본 산업 분야에서 일반적인 2 개의 32 비트 워드 명령 메모리 및 레지스터의 사용을 허용하기 때문이다. 충분한 양의 처리되지 않은 명령 데이타를 유지하는 다른 방법의 사용은 본 발명의 양상을 사용하는 것과 양립한다.

[0090] 일단, 64 비트의 새로운 명령 데이타가 명령 레지스터 (306 및 307) 로 로드되면, 제어 논리 (304) 는 또한, 1) 명령 어드레스 공간 내에서의 명령 데이타의 위치, 2) 처리된 명령 데이타의 세트, 및 3) 이전의 처리된 명령의 길이에 기초하여, 입력 (A, B, C 및 D) 상에서 검색된 다음 48 비트의 명령 데이타를 출력하는 제어 신호 (320) 를 사용하여 로테이터 (308) 를 구성한다. 특히, 로테이터 (308) 는 다음 인 라인 비트 (the next-in-line) 의 명령 데이타를 가장 중요한, 또는 맨 왼쪽의, 위치에 놓아서, 처리될 다음 인 라인의 48 비트의 명령 데이타 세트를 출력하도록 구성된다.

[0091] 예를 들어, 첫째의 2 개 워드, 또는 64 비트의 명령 데이타를 레지스터 (306 및 307) 로 로딩할 때, 짝수 명령 레지스터 (307) 에서의 명령 데이타가 다음 인 라인이면, 로데이터 (308) 의 출력은 입력 (A, B 및 C (ABC)) 상에 이 순서대로 검색된 명령 데이타로 구성된다. 홀수 명령 레지스터 (306) 내의 명령 데이타가 다음 인 라인이면, 로테이터 (308) 는 입력 (C, D 및 A (CDA)) 상에 이 순서대로 검색된 명령 데이타를 출력하도록 구성된다.

[0092] 명령이 처리됨에 따라, 전술한 바와 같이, 새로운 명령 데이타가 데이타 레지스터 (306 및 307) 로 로드되고, 로테이터 (307) 는 이전에 처리된 명령의 사이즈에 기초하여 출력 (324) 상에 다음 인 라인 명령 데이타를 이어서 출력하도록 구성된다. 이전에 처리된 명령의 사이즈는 출력 (324) 의 처음 5 비트의 복사에 해당되는 헤더 데이타 (322) 에 의해서 제어 논리 (304) 로 전달된다. 앞에서 언급한 바와 같이, 명령 길이가 명령 데이타로부터 직접 결정되는 것을 허용하기 때문에 처음 5 비트의 사용이 바람직하다하더라도, 명령 길이를 제어 논리 (304) 로 특정화하기 위한 어떠한 소정의 방법도 본 발명의 실시와 양립한다.

[0093] 본 발명의 일 실시예에서, 이전의 명령의 사이즈는 표 1 에 따라 2 비트의 상태 정보 (I1 및 I0) 로 코딩된다.

표 1

[0094]

I1 I0	명령 크기
0 0	브랜치/스톨 (stall)/리세트
0 1	16 비트
1 0	32 비트
1 1	48 비트

전 명령 포맷 (full instruction format)

이에 더하여, 로테이터 (308) 의 구성은, 표 2 에 표시된 바와 같이 코딩된 제어 (320) 을 이루는 2 개의 선택 비트 (S1 및 S0) 에 의해서 제어된다.

표 2

S1 S0	로테이터 출력
0 0	ABC
0 1	BCD
1 0	CDA
1 1	DAB

로테이터 선택 제어 비트 및 출력

분명하게 알 수 있는 바와 같이, S1 및 S0 의 상태가 증가함에 따라, 로테이터 (308) 의 출력은 좌측-회전 (left-rotated), 또는 배럴 시프트된다 (barrel- shift). 좌측-회전은 각각의 입력 그룹 (A, B, C 및 D) 가 출력 상에 좌측으로 시프트되도록 하는 것이다. 출력의 맨 좌측 위치에 있는 입력 그룹은 제거된다. 이전에 출력에서 표명되지 않은 입력 그룹은 다음에 맨 우측 위치에서 출력된다.

S1 및 S1 의 상태와 그에 따른 로테이터 (308) 의 구성은 다양한 길이의 명령에 응답하여 변하는 양에 의해서 갱신되거나 회전된다. 특히, 처리되는 명령의 길이를 나타내는 값 (I1 I0) 은 제어 비트 (S1 및 S0) 에 더해지고, 어떤 실행 값 (carryout value) 도 버려진다 (discard). 즉,

수학식 1

S1(t+1),S0 (t+1) = S1(t),S0 (t) + I0, I1

브랜치 또는 리세트 조건에 대해서, S1 및 S0 의 값은 프로세싱이 브랜치되거나 리세트되는 특정 명령에 기초하여 리세트되어, 수학식 1 이 이용되지 않는다. 브랜치, 리세트 및 스톨 명령을 프로세싱하기 위한 다양한 방법들이 본 기술분야에서 잘 알려져 있고, 이 프로세싱은 본 발명과 특별히 관련되지 않기때문에 더이상 설명하지 않겠다.

예시적인 프로세싱에서, 로테이터 (308) 는 ABC 의 출력과 00 에서의 신호 비트 (S1 및 S0) 로 시작된다. 16 비트 명령이 수신되면, I1 및 I0 의 해당하는 명령 길이 비트가 S1 및 S0 에 더해져서, 01 의 S1 및 S0 을 산출하고, 이것은 BCD 의 로테이터 (308) 로부터의 출력 (324) 에 해당한다. BCD 의 출력은 처음 16 비트의 명령 데이타 (입력 A) 가 처리된 후에 다음 인 라인 세트의 명령이 된다.

다음 명령이 32 비트 명령이면, 명령 길이 (I1 및 I0) 는 01 인 현재 S1 및 S2 상태에 더해져서, 11 을 산출한다. 결과적인 출력은 DAB 가 되고, 이것은 처리되지 않은 다음 48 비트의 명령 데이타에 해당하여, 입력 D 상에 수신된 다음 인 라인 명령 데이타가 가장 중요한, 또는 맨 좌측의 위치에 위치하도록 구성된다. 그 이전에, 입력 (B 및 C) 로부터 명령 데이타가 이미 처리된 상태이다. 이전의 32 비트 명령의 처리동안에 새로운 데이타가 전술한 명령 데이타를 로딩하는 프로세스에 따라서 명령 레지스터 (307) 로 로드된다는 것에 유의하여야 한다.

48 비트 명령이 처리되면, 선택 비트 (S1 및 S0) 의 상태는 11 의 명령 길이 (I0 및 I1) 에 의해서 증가되어, 10 의 S1 및 S0 을 산출하고, 이것은 로테이터 (308) 를 출력 (CDA) 로 구성한다. CDA 의 출력은 처리될 다음 48 비트의 명령 데이타에 해당하고 다음 인 라인 비트는 가장 중요한 위치로 배치된다. 그 전에, 입력 (D, A 및 B) 상의 명령 데이타가 처리되었다. 이전의 명령이 처리되는 동안에 새로운 명령 데이타가 레지스터 (306) 으로 판독된다는 것에 또한번 다시 유의하여야 한다. 관련된 명령 데이타 프로세싱을 수행하는 전술한 논리는 실시예의 목적으로 제공된 것이며, 전술한 명령 데이타 프로세싱을 수행하는 다른 논리의 사용은 분명하고 본 발명의 사용과 양립한다.

따라서, 실시예에서, 제어 논리는 로테이터 (308) 를 명령 레지스터 (306 및 307) 로부터 수신된 다음 48 비트의 명령 데이타를 출력하도록 구성되어, 처리될 다음 명령 비트가 맨 좌측 위치에 위치하게 된다. 처리될 다음 세트의 명령 데이타가 확인될 수 있는 소정의 배치로 명령 데이타를 명령 프로세싱 부 (158) 에 제공하는 것은 또한 본 발명의 사용과 양립한다는 사실을 본 기술 분야의 당업자들은 알수 있다.

전술한 바와 같이, 처리되지 않은 다음 48 비트의 명령 데이타를 출력하는 것에 더하여, 로테이터 (308) 는 또한 처리될 다음 명령의 사이즈를 제어 논리 (304) 에 지시한다. 특히, 로테이터 (308) 는 처리될 다음 5 비트의 명령 데이타의 부가적인 복사를 논리 제어 (304) 에 출력한다. 본 발명의 바람직한 실시예에서, 명령의

길이는 명령의 처음 5 비트에 의해서 특정된다.

도 8 은 본 발명의 일 실시예에 따라 구성될 때의 MAC 부 (128) 의 블록도이다. 시프트 라이트 (shift right; 900) 는 누산될 40 비트 입력을 수신하고 0 또는 16 비트만큼 값을 시프트하여, 출력이 멀티플렉서 (901) 의 하나의 입력으로 인가된다. 멀티플렉서 (901) 의 다른 입력은 0 ×800 값을 수신한다. 곱셈기 (멀티플라이어; 902) 는 각 입력에 대한 총 17 비트에 대해 명령 디코더 (158) 로부터의 신호 비트와 함께 곱하여질 2 개의 16 비트 값을 수신한다.

곱셈기 (902) 의 출력은 시프트 레프트 (shift left; 904) 에 의해 수신되는데, 이것은 명령 디코더 (158) 에 의해서 특정되는 바와 같이, 0, 1, 2 또는 3 비트만큼 출력을 시프트한다. 가산회로/감산회로 (906) 는 명령 디코더 (158) 에 의해 지시되는 대로 2 개의 입력 값의 가산 또는 감산을 수행하고 그 결과를 출력하며, 이것은 본 발명의 일례에서의 레지스터 뱅크 입력 포트 (PI4) 로 인가된다.

본 발명의 실시예에서 MAC 부 (128) 내의 시프트 라이트 16 유닛 (900) 의 사용은 다른 형태의 MAC 부에 대한 추가적인 이용을 제공한다. 더 자세하게 말해서, 시프트 라이트 16 유닛 (900) 의 사용은 감소된 수의 클럭 사이클 에서의 2 배 정밀도 동작을 수행하는 것을 용이하게 한다. 예를 들어, 32 비트 수 (A) 가 16 비트 수 (B) 와 곱해지는 2 배 정밀도 동작을 수행하기 위해서, 32 비트 수의 아래 16 비트 (Al) 가 먼저 제 1 클럭 사이클 동안에 16 비트 수 (B) 와 곱해져서, 레지스터 뱅크 (120) 내에 저장된 중간값 (I) 을 산출한다. 제 2 클럭 사이클 동안에, 중간값 (I) 은 시프트 라이트 16 유닛 (900) 으로 입력되고 16 비트만큼 우측으로 시프트된다. 이에 더하여, 16 비트 수 (B) 와 32 비트 수 (A) 의 위 16 비트 (Ah) 가 곱해지고, 그 결과는 시프트 라이트 16 유닛 (900) 으로부터온 우측으로 시프트된 중간값 (I) 과 가산된다. 따라서, 2 배 정밀도 곱하기는 3 이라기 보다 2 개의 클럭 사이클에서 수행된다. 일반적으로, 많은 2 배 정밀도 동작은 하나 또는 그 이상의 변수가 다른 변수에 대해 시프트되도록 요구하며, 멀티플라이 또는 어큐뮬레이트 동작의 하나로서 동일한 클럭 사이클동안에 수행되는 시프팅 단계를 허용하는 것은 2 배 정밀도 동작을 수행하는 데 필요한 사이클의 수를 감소시킨다.

보통, 3 개 클럭 사이클이 요구되는데, 이는 제 1 곱하기 동작, 시프트 동작, 및 제 2 곱하기 동작 각각은 전형적으로 클럭 사이클이 요구되기 때문이다. 따라서, 시프트 회로의 사용은 2 배 정밀도 곱하기를 수행하는 데 요구되는 클럭의 수를 감소시킨다. 다른 크기의 피연산자 (operands) 에 관련된 다른 2 배 정밀도 동작은 또한, 시프트 라이트 부 (900) 의 사용에 의해서 용이하게 된다.

Ⅱ. 명령 구성 (INSTRUCTION CONFIGURATION)

A. 개관.

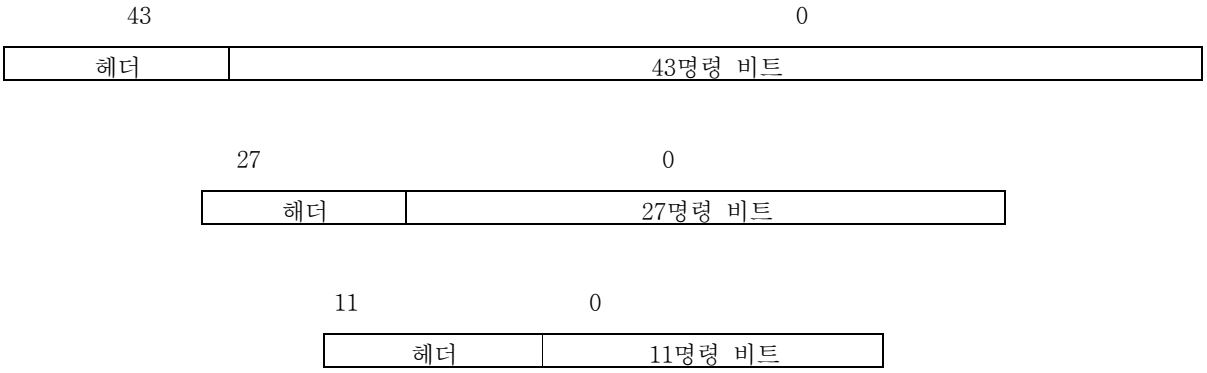
도 9 는 본 발명의 실시예에서 사용된 명령 체계를 설명하는 블록도이다. 블록 (402) 은 DSP 의 동작을 제어하는 16, 32 또는 48 비트로 이루어진 가변 길이 전 명령 (variable length full instruction)을 나타낸다. 가변 길이 명령은 블록 (403) 에 나타난 바와 같이, 메모리 무브 및 프로그램 플로우 (memory move and program flow) MMPE 와 일반 명령 프래그먼트를 포함하는 명령 프래그먼트로 구성된다. 본 발명의 실시예에서 사용된 일반 명령 프래그먼트는 MAC8, MAC16, ALU8, ALU16, DMOV16, DMOV24 및 DL40 명령 프래그먼트를 포함한다. MMPF 명령 프래그먼트는 OneMem11, TwoMem19, TwoMov19 및 TreeMem24 명령 프로그래먼트를 포함한다. MMPF 명령 프래그먼트는 블록 (406) 에 나타난 MMPF 명령 서브프래그먼트로 구성된다. MMPF 명령 서브프래그먼트는 LD(A), LD(B), ST(A), ST(B), LS(C), DMOVA, DMOVB, 및 PF8 을 포함한다. 다양한 전 명령, 명령 프래그먼트 및 명령 서브프래그먼트는 아래에서 더 자세히 설명되어있다.

B. 전 명령 (Full Instruction)

본 발명의 바람직한 실시예에서, DSP 는 16, 32 및 48 비트의 길이를 갖는 전 명령을 사용하여 제어된다. 전 명령은 하나 또는 그 이상의 명령 프래그먼트를 조합하여 형성된다. 전 명령은 명령 메모리 (152) 내에서의 연속적인 저장 및 DSP 에 의한 프로세싱을 감안하도록 구성된다. 전 명령의 포맷 (format) 및 구성은 아래 설명되어있고, 그 다음에 명령 프래그먼트의 포맷과 구성이 뒤따른다. 동작동안, DSP 는 전 명령의 각각의 클럭 사이클을 처리한다. 따라서, 곱하기 동작은, 개별적인 동작이 개별적인 세트의 선택된 명령 프래그먼트에 의해 결정되면서, 각각의 전 명령의 프로세싱동안에 수행될 수 있다.

본 발명의 실시예에서 사용된 3 개의 전 명령의 포맷은 표 3 에 나타나 있다.

표 3



전 명령 포맷

각각의 전 명령에 사용된 5 비트 헤더는 전 명령의 길이를 지시하고, 어떤 추가적인 포맷은 전체 포맷의 내용에 관한 것이다. 본 발명의 실시예에서 사용된 헤더의 포맷은 표 4 에 제공되어 있다.

표 4

5비트 헤더					명령 길이
0	0	0	0	X	16비트 명령 (2 타입;2 types)
0	0	0	1	X	32비트 명령 (2 타입)
0	0	1	X	X	48비트 명령 (4 타입)
0	1	X	X	X	32비트 명령 (8 타입)
1	X	X	X	X	48비트 명령 (16 타입)

전 명령 헤더 포맷

각각의 전 명령 (16, 32 및 48 비트 길이) 은 하나 또는 그 이상의 명령 프래그먼트를 포함한다. 표 5 는 본 발명의 실시예에서의 이용 가능한 명령 프래그먼트의 목록을 제공한다. 명령 프래그먼트의 포맷 및 동작의 더 자세한 설명은 전 명령에 대한 논의 후에 제공된다.

표 5

필드(Field)	설명	폭(Width)
MAC8*	8비트 MAC 동작	8
ALU8*	8비트 ALU 동작	8
OneMem11*	1 메모리 동작	11
MAC16*	16비트 MAC 동작	16
ALU16*	16비트 ALU 동작	16
DMOV16*	조건부 Reg 무브(Conditional Reg Move)/Inport/Outport	16
TwoMem19	2 메모리 동작	19
TwoMov19*	2 메모리/데이타 무브 동작	19
DMOV24	로드(Load)/Store Direct/Load Addr/Jump	24
TreeMem27	3 메모리 동작	27
DL40	듀얼 로드(Dual-Load)	40

* 는 필드가 어떤 비트 패턴을 사용하여 놉 (nop'ed; 무연산) 될 수 있다는 것을 지시한다.

표 5. 명령 프래그먼트

표 6 - 8 은 본 발명의 실시예에 따른 48, 32 및 16 비트 전 명령 내에서 사용될 수 있는 명령 프래그먼트의 다

양한 조합을 제공한다. 명령 프래그먼트의 다른 조합이 본 발명의 사용 및 동작과 양립하지만, 여기에 개시된 조합의 일정한 특징들은 아래 설명하고 있는 바와 같이 바람직하다. 이에 더하여, 전 명령의 전부 또는 일부가 "예약된 (reserved)" 이라고 나타나 있는 곳에서는, 설명된 실시예에서 개별적인 명령 조합이 특정되거나 사용되지 않았지만, 이러한 전 명령 조합의 추후의 사용이 예상된다.

표 6 은 본 발명의 실시예에 따라 수행될 때 16비트 전 명령 포맷을 제공한다. 전 명령은 11개의 명령 비트가 뒤따르는 5개의 비트 헤더로 구성된다.

표 6

11 헤더					11 명령 비트			0
0	0	0	0	0	MAC8	0	0	0
0	0	0	0	0	ALU8	0	0	1
0	0	0	0	0	Reserved	0	1	0
0	0	0	0	0	Reserved	0	1	1
0	0	0	0	0	Reserved	1	0	0
0	0	0	0	0	Reserved	1	0	1
0	0	0	0	0	Reserved	1	1	0
0	0	0	0	0	Reserved	1	1	1
0	0	0	0	1	OneMem11			

16비트 전 명령 포맷

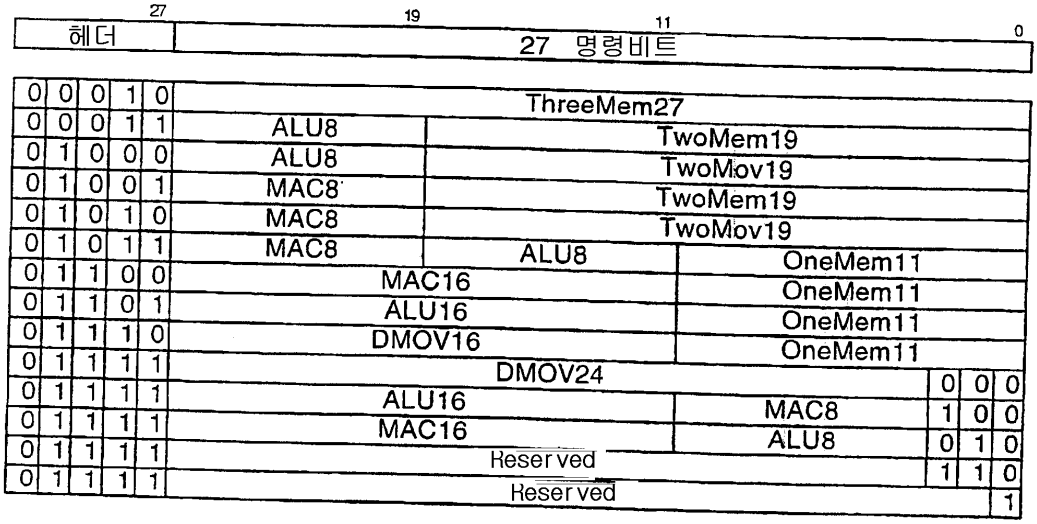
주 : 16비트 NOP 는 MAC8 의 NOP 를 이용하여 얻어진다.

헤더 비트는 명령의 타입에 대한 일부 정보 뿐만 아니라 명령 길이를 나타낸다. 00000 의 헤더용으로, 3개의 최하위 테일 비트는 수행되는 동작을 더 특정화하는 데 사용된다. 특히, 000 의 테일 비트는 나머지 8개 비트가 MAC8 명령 프래그먼트를 포함하는 것을 나타낸다. 001 의 테일 비트는 나머지 8개 비트가 ALU8 명령 프래그먼트를 포함하는 것을 나타낸다. 다른 테일 비트 조합을 위하여, 어떠한 명령도 특정화되지 않는다. 가장 짧은 명령은 저장하기에 최소량의 메모리를 요구하기에, 상기한 16비트 전 명령 사용은 특정한 집합의 동작을 수행하는 데 필요한 명령 메모리 양을 감소시킨다. 따라서, DSP 의 전체 크기 및 이에 따른 비용 및 전력 소모가 또한 감소된다.

16비트 명령은 특히 한 개 또는 감소된 개수의 동작이 수행될 수 있는 상태일 때 사용된다. 특히, 한 개의 동작만을 특정화하기에 필요한 명령 크기가 감소될 수 있고, 따라서 절반 워드 사용, 또는 한 개 동작을 수행하기 위한 16비트 명령이 감소될 수 있다. 또한, 16비트 명령은, 수행될 것으로 예상되는 거의 모든 동작을 포함하는 MAC, ALU, 메모리 이동 또는 프로그램 흐름 동작용으로 사용될 수 있다.

표 7 은 본 발명의 실시예에 따라 구성될 때 명령 프래그먼트 조합 및 32 비트 전 명령의 관련된 포맷을 도시한다.

표 7



32비트 전 명령 포맷

상기한 바와 같이, 5개의 헤더 비트는 명령 프래그먼트의 특정한 조합 뿐만 아니라 전 명령 길이를 나타낸다.

예를 들어, 00010 헤더는 나머지 27 명령 비트가 ThreeMem27 명령 프래그먼트를 포함하는 것을 나타내며, 00011 헤더는 나머지 27 명령 비트가 TwoMem19 명령 프래그먼트가 뒤따르는 ALU8 명령 프래그먼트를 포함하는 것을 나타낸다.

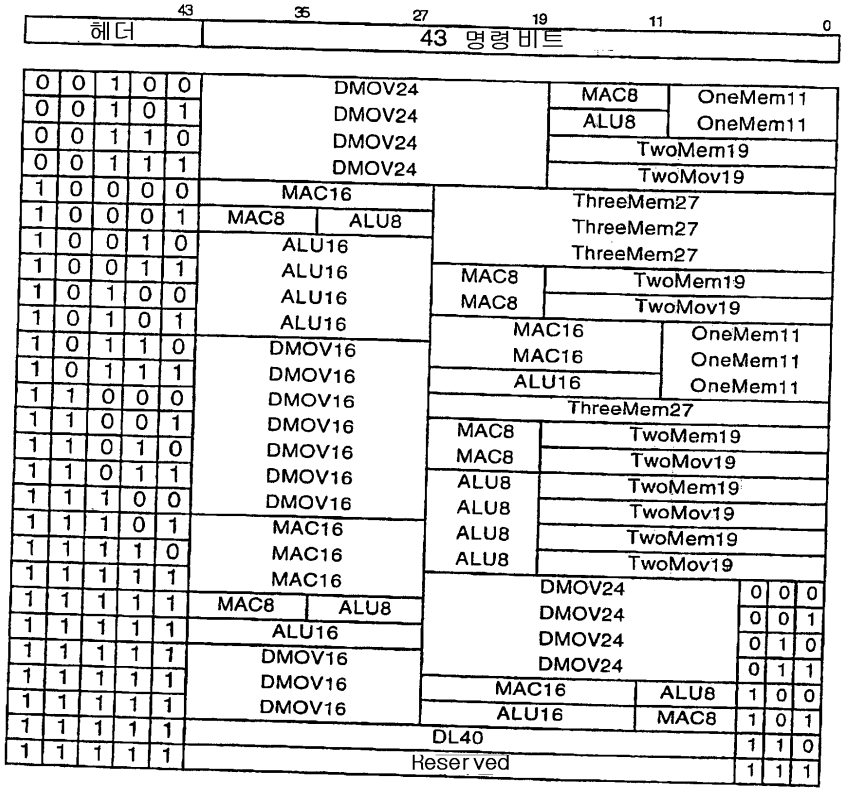
01111 헤더용으로, 최하위 테일 비트는 명령 프래그먼트의 조합을 또한 나타낸다. 예를 들어, 0 의 최하위 테일 비트용으로, 그 다음 2개의 최하위 비트는 나머지 24비트가 DMOV24, MAC8 이 뒤따르는 ALU16, 또는 ALU8 명령 프래그먼트가 뒤따르는 MAC16 를 포함하는지 여부를 나타낸다. 다른 테일 비트 상태, 1의 최하위 테일 비트는 예약된 조합을 특정화한다.

32개의 비트 명령으로 인하여 대부분의 수행 동작이 동시에 수행될 수 있고, 이것은 명령 크기를 줄이는 한편 파이프라인을 용이하게 한다. 예를 들어, 필터링과 같은 응용을 위하여 2개의 인출 동작 및 멀티플라이/어큐뮬레이트 동작을 수행하는 것은 흔한 일이다. 32비트 명령으로 인하여 상기와 같은 동작이 필 (fill) 48 비트 명령 공간을 필요로 하지 않고 파이프라인 형식으로 수행될 수 있다.

또한, 32비트 명령으로 인하여 가장 큰 명령 크기를 사용하지 않고도 MAC 및 ALU 동작이 프로그램 점프 및 콜 (call) 동작뿐만 아니라 동시에 수행될 수 있다.

표 8 은 본 발명의 실시예에 따라 수행될 때 명령 프래그먼트 조합 및 48 비트 전 명령용 포맷을 나타낸다.

표 8



48비트 전 명령 포맷

5개의 헤더 비트는 특정한 명령 프래그먼트 조합뿐만 아니라 명령 길이를 특정화한다. 예를 들어, 00100 헤더 비트는 43개의 나머지 명령 비트가 DMOV24, MAC8, OneMem11 명령 프래그먼트로 구성된 것을 나타낸다. 10011 헤더 비트는 43개의 나머지 비트가 ALU16, MAC8, 및 TwoMem19 명령 프래그먼트로 구성된 것을 나타낸다.

11111 헤더 비트용으로, 3개의 최하위 테일 비트는 나머지 명령 비트에 포함된 명령 프래그먼트를 또한 나타낸다. 000 테일 비트는 나머지 40 명령 비트가 MAC16 및 DMOV24 명령 프래그먼트를 포함하는 것을 나타낸다. 001 테일 비트는 나머지 40 개의 명령 비트가 MAC8, ALU8, 및 DMOV24 명령 프래그먼트를 포함하는 것을 나타낸다. 110 테일 비트는 나머지 40개 명령 비트가 DL40 명령 프래그먼트를 포함하는 것을 나타낸다.

48비트 전 명령내에 제공되는 명령 프래그먼트 조합으로 인하여 많은 동작이 동시에 수행될 수 있고 따라서 직렬로 수행되는 것보다 빠르게 수행될 수 있다. 예를 들어, 여러 개의 48비트 전 명령으로 인하여 ALU 동작, MAC 동작 및 메모리 동작이 동시에 수행될 수 있다. 메모리 동작은 로드, 저장, 및 데이터 이동 동작을 포함하며, 흔히 다중 메모리 위치가 한번에 액세스될 수 있게 한다.

48비트 명령으로 인하여 승산 동작이 ALU 동작, 데이터 인출, 및 프로그램 흐름 동작과 함께 파이프라인 형식으로 수행될 수 있다. 이것은, (시프팅과 같은) ALU 동작이 뒤따르는 MAC 동작을 수행함으로써 흔히 수행되는 스케일링 동작과 조합될 때 필터링에 유용할 수 있다. MAC 및 ALU 를 사용하는 다른 응용에는 3개 이상의 데이터 스트림을 조합하는 것이 포함된다. 특히 3개의 버스 아키텍처를 사용시 48비트 명령은 이러한 경우에 파이프라인 동작을 용이하게 한다.

이것은 단일 48비트 전 명령 내지 5 개 (MAC, ALU, FETCH1, FETCH2, 및 STORE) 로 수행될 수 있는 동작 수를 효율적으로 증가시킨다. DSP 에서 다중 명령을 동시에 수행할 수 있는 기능은, 일반적으로, DSP 내부의 다양한 처리 시스템을 접속하기 위한 다중 내부 버스와 함께 DSP 를 사용함으로써 더 향상된다. 상이한 세트의 데이터가 상이한 버스를 사용하여 동시에 이동 및 액세스될 수도 있다.

수행될 수 있는 동작 수에 의거하여 명령 길이를 변경함으로써 명령 메모리가 사용되는 효율을 더 증가시킨다. 어떤 특정한 태스크는 다중 동작이 동시에 수행될 수 있는 주기, 및 보다 적은, 또는 한개의, 동작이 수행

될 수 있는 다른 주기를 갖는다. 동시에 수행될 수 있는 동작 수에 따라 명령 길이를 조절함으로써, 명령 메모리 양이 감소된다.

상기한 방법이 타이트한 (tight) 명령 패킹과 결합되어 함께 사용될 때, 필요한 명령 메모리가 더 감소된다. 가변 길이 명령 또는 타이트하게 패킹된 명령 또는 두 명령을 함께 사용함으로써 다중 버스 아키텍처 및 다중 액세스 레지스터 뱅크 사용이 용이해지며, 보다 많은 회로 영역이 이용가능해진다. 따라서, 이러한 본 발명의 태양은 향상된 성능 및 향상된 효율을 동시에 제공하기 위해 결합된다.

C. 명령 프래그먼트

상기한 바와 같이, 전 명령은 한 개 또는 소정의 방식으로 그룹화된 그 이상의 명령 프래그먼트의 세트로 구성된다. 본 발명의 실시예에서 이용가능한 명령 프래그먼트 세트가 표 5 에 도시된다. 본 발명의 실시예에서 제공된 전 명령을 사용하여 이용가능한 조합 및 명령 프래그먼트는 동작 세트가 결합되어 함께 수행될 수 있도록 설계되어 주어진 동작을 수행하는 데 필요한 명령 메모리의 양이 감소된다. 본 발명의 실시예에서 사용되는 다양한 명령 프래그먼트의 포맷 및 동작이 아래에 뒤따른다.

C.1 명령 프래그먼트 명명법

다음의 명령 프래그먼트 및 서브프래그먼트에서, 다음의 약어는 표 9 및 10 에 도시된 레지스터를 언급하는 데 사용된다. 또한, 본 발명의 실시예에서 사용되는 특정한 비트 코드 (매핑) 는 좌측에 도시된다.

표 9

	Dreg
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	L0h
1001	L1h
1010	L2h
1011	L3h
1100	L0l
1101	L1l
1110	L2l
1111	L3l

	R0-R7	Lh/Ll	A0-A7	Lreg/Dreg
000	R0	L0h	A0	L0
001	R1	L1h	A1	L1
010	R2	L2h	A2	L2
011	R3	L3h	A3	L3
100	R4	L0l	A4	D0
101	R5	L1l	A5	D1
110	R6	L2l	A6	D2
111	R7	L3l	A7	D3

	AS	AL
0	AS0	AL0
1	AS1	AL1

	R0-R3	L0-L3	D0-D3	C0-C3	Cmod
0 0	R0	L0	D0	C0	++
0 1	R1	L1	D1	C1	--
1 0	R2	L2	D2	C2	++CM0
1 1	R3	L3	D3	C3	++CM1

cond									
00000	LT		01000	L0LT		10000	L1LT		11000	L2LT
00001	LE		01001	L0LE		10001	L1LE		11001	L2LE
00010	EQ		01010	L0EQ		10010	L1EQ		11010	L2EQ
00011	NE		01011	L0NE		10011	L1NE		11011	L2NE
00100	GE		01100	L0GE		10100	L1GE		11100	L2GE
00101	GT		01101	L0GT		10101	L1GT		11101	L2GT
00110	OV		01110	L0OV		10110	L1OV		11110	L2OV
00111	Uncond		01111	Rsvd		10111	Rsvd		11111	Rsvd

명령 프래그먼트 명명법 및 코드

주 : L3 는 조건문을 갖지 않는다.

표 10

	RegA	regB
00000	R0	R0
00001	R1	R1
00010	R2	R2
00011	R3	R3
00100	R4	R4
00101	R5	R5
00110	R6	R6
00111	R7	R7
01000	L0h	L0h
01001	L1h	L1h
01010	L2h	L2h
01011	L3h	L3h
01100	L0l	L0l
01101	L1l	L1l
01110	L2l	L2l
01111	L3l	L3l
10000	A0	B0
10001	A1	B1
10010	A2	B2
10011	A3	B3
10100	A4	B4
10101	A5	B5
10110	A6	B6
10111	A7	B7
11000	AS0	BS0
11001	AS1	BS1
11010	AL0	BL0
11011	AL1	BL1
11100	AM0	BM0
11101	AM1	BM1
11110	Reserved	Reserved
11111	Reserved	Reserved

	regC
0000	L0
0001	L1
0010	L2
0011	L3
0100	D0
0101	D1
0110	D2
0111	D3
1000	C0
1001	C1
1010	C2
1011	C3
1100	CM0
1101	CM1
1110	Reserved
1111	Reserved

명령 프래그먼트 명명법 및 코드

regA 는 A 메모리에 저장될 수 있는/A 메모리로부터 로드될 수 있는 모든 레지스터를 포함한다.

regB 는 B 메모리에 저장될 수 있는/B 메모리로부터 로드될 수 있는 모든 레지스터를 포함한다.

regC 는 C 메모리에 저장될 수 있는/C 메모리로부터 로드될 수 있는 모든 레지스터를 포함한다.

C.2 명령 프래그먼트 설명

일련의 명령 프래그먼트들은 MAC8과 MAC16의 2가지 유형의 MAC 명령 프래그먼트들을 포함한다. MAC8명령 프래그먼트들은 부호화된-부호화되지 않은 것과 부호화-부호화된 곱셈형을 지원하고 그 결과는 어큐뮬레이터 (L0 또는 L1)에 저장된다. MAC8 명령 프래그먼트는 16 비트 전(full) 명령을 이용하는 MAC 동작을 허용하고, MAC 동작들을 요하는 많은 병렬 명령 조합들이 48비트 명령 대신에 32비트 명령으로 인코드되도록 함으로써 명령 RAM을 절약한다. 일반적으로, MAC8 명령들에 의해 수행되는 절차는 다음 수학식을 따른다.

수학식 2

$$\begin{Bmatrix} L0 \\ L1 \end{Bmatrix} = \begin{bmatrix} L0\pm \\ L1\pm \end{bmatrix} \begin{Bmatrix} R0 \\ R2 \\ R4 \\ R6 \end{Bmatrix} * \begin{Bmatrix} R0 \\ R1 \\ R3 \\ R5 \end{Bmatrix} \begin{bmatrix} (SU) \\ (SS) \end{bmatrix}$$

수학식 2 에서 나타난 바와 같이, MAC8 명령 프래그먼트는 레지스터 (L0 는 L1) 의 내용이 레지스터 (R0, R2, R4 및 R6) 와 (R0, R1, R3 및 R5) 의 곱으로 합계가 되도록 하거나 레지스터들의 곱으로 직접 설정되도록 한다.

게다가, 부호화되거나 부호화되지 않은 곱셈들이 명기될 수 있다. MAC 연산이 MAC8 명령을 이용하여 수행될 수 있는 레지스터들의 수를 제한함으로써, 명령의 길이는 8비트로 유지될 수 있어 더 짧은 8비트 명령 프래그먼트를 이용하여 MAC 연산이 수행되도록 한다.

MAC8 명령에 의해 수행된 특정 동작이 표 11 에 설명된 명령을 구성하는 8비트의 값에 의해 구체화된다.

표 11

7	6	5	4	3	2	1	0
MAC Operation			mac8Op1		mac8Op2		SU/SS

MAC8 명령 프래그먼트 포맷

SU/SS 는 부호화된 또는 부호화되지 않은 곱셈을 명기한다. MAC8 명령 프래그먼트 내의 다양한 동작들을 명기하는 코드들이 표 12 에 나열되어 있다.

표 12

MAC Operation	
000	L0=
001	L1=
010	L0=L0+
011	L1=L1+
100	L0=L0-
101	L1=L1-
110	L0=L1+
111	L0=L1-

mac8Op1	
0 0	R0
0 1	R2
1 0	R4
1 1	R6

mac8Op2	
0 0	R0
0 1	R1
1 0	R3
1 1	R5

SU/SS	
0	SU
1	SS

MAC8 명령 프래그먼트 코드

그러므로, 0×99의 MAC8 명령은 레지스터 R0 와 R3 의 부호화되지 않은 곱과 함께 레지스터 (L0) 의 내용의 합을 레지스터 L0 로 이동시킨다.

MAC16 명령 프래그먼트는 추가의 레지스터들이 멀티플라이-어큐뮬레이트 동작에서 이용되도록 함으로써 부가적인 유연성을 제공한다. 수학식 3 은 MAC16 명령 프래그먼트를 이용하여 수행될 수 있는 동작들을 설명한다.

수학식 3

$$\begin{Bmatrix} L0 \\ L1 \\ L2 \\ L3 \end{Bmatrix} = \begin{bmatrix} L0 & [>>16] & \pm \\ L1 & [>>16] & \pm \\ L2 & [>>16] & \pm \\ L3 & [>>16] & \pm \end{bmatrix} \; macOp1 * macOp2 \;\; [(mtype)] \;\; [<< mshift] \;\; [:CPS]$$

예컨대, 모든 어큐뮬레이터 (L0-L3) 는 비록 어큐뮬레이터들의 모든 조합들이 멀티플라이-어큐뮬레이터 명령들에서 허용되지 않을지라도 수신지로 이용될 수 있다. CPS 필드는 코프로세서가 특정 동작을 병렬로 수행해야 함을 신호한다. MAC16 명령에 의해 수행된 특정 동작은 표 13 에 나타낸 명령을 구성하는 16비트의 값들에 의해 특정된다.

표 13

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAC Operation				macOp1			macOp2				mtype		mshift		CPS

MAC16 명령 프래그먼트 포맷

MAC16 명령 프래그먼트 내의 다양한 동작들을 명기하는 코드들이 표 14 에 나열되어 있다.

표 14

	MAC Operation	macOp2
0000	L0=	R0
0001	L1=	R1
0010	L0=L0 [>>16] +	R2
0011	L1=L1 [>>16] +	R3
0100	L0=L0 [>>16] -	R4
0101	L1=L1 [>>16] -	R5
0110	L0=L1 [>>16] +	R6
0111	L0=L1 [>>16] -	R7
1000	L2=	L0h
1001	L3=	L1h
1010	L2=L2 [>>16] +	L2h
1011	L3=L3 [>>16] +	L3h
1100	L2=L2 [>>16] -	L0l
1101	L3=L3 [>>16] -	L1l
1110	L2=L3 [>>16] +	L2l
1111	L2=L3 [>>16] -	L3l

macOp1	
0 0 0	R0
0 0 1	R2
0 1 0	R4
0 1 1	R6
1 0 0	L0h
1 0 1	L1h
1 1 0	L2h
1 1 1	L3h

mtype	
0 0	(SU)
0 1	(UU)
1 0	(SS)
1 1	주 참조

mshift	
0 0	<< 0
0 1	<< 1
1 0	<< 2
1 1	<< 3

CPS	
0	OFF
1	ON

MAC16 명령 프래그먼트 코드

주 : 명령 L0=R0*R0(SU)<<0 은 NOP로 디코드된다.

mtype 11 은 부호화-부호화 멀티플라이/어큐뮬레이트 명령에 대해 스트레이트(straight) 곱셈과 16만큼 오른쪽으로 어큐뮬레이터를 시프트하는 것에 대한 RND로 이용된다.

mtype SU 및 macOp1=macOp2 를 갖는 MAC 명령은 허용되지 않는다.

MAC16 명령 프래그먼트는 3까지의 좌측 시프트를 허용하여 스트레이트 곱셈(누산 없음) 동안에 라운드 동작을 수행할 수 있는데, 이 라운드는 시프트 후에 일어난다. 누산이 수행될 때, 가산될 어큐뮬레이터는 부호화-부호화된 곱셈과 병렬로 16만큼 시프트 다운될 수 있다. CPS 비트는 MAC 동작에서 이용된 데이터가 코프로세서에 전송되어야함을 가리키기 위한 코프로세서 스트로브 비트이다.

MAC8 명령 프래그먼트는 MAC16에 의해 수행될 수 있는 동작들의 서브세트를 수행한다는 점을 주목한다. MAC8 명령 프래그먼트에 대해 선택된 특정 집합의 명령들은 MAC16 명령 프래그먼트를 이용하여 수행될 수 있는 상기 집합의 동작들로부터 가장 흔히 수행된다. 이는 MAC8 명령 프래그먼트를 이용하여 MAC 연산들의 대부분이 수행될 수 있도록 허용함으로써 프로그램 메모리를 절약한다.

8비트 ALU8 명령 프래그먼트는 MAC 연산 (MAC8과 MAC16)과 병행하여 가장 흔하며 즉시 데이터를 포함하지 않는 ALU 동작들로 구성된다. 모든 ALU8 시프트 동작들은 명령 인코딩 비트들을 절약하기 위해 내부 시프트 레지스터 (SR) 를 이용하는 산술 시프트이다. ALU8 명령 프래그먼트를 이용하여 수행된 동작들은 표 15 에 나타나 있다.

표 15

NOP;	NOP (병렬 조합을 위해 필요함)
LD = DETNORM(LS);	블록 정규화 인자를 결정함
LD = SET(LS);	Accumulator를 복사(포화는 아님)
LD = LS << SR;	Accumulator시프트
LD = RND(LS << SR);	Accumulator를 시프트하고 라운드함
LD = LD±(LS << SR);	시프트된 Accumulator를 누산함
LD = LS ± LT;	Accumulator를 가산 또는 감산
LS ± LT;	Accumulator의 가산/감산 결과를 0으로(플래그를 세트)

ALU8 명령 프래그먼트 동작

LS 는 로드 소스(L0-L3)이고 LD는 로드 수신지(L0-L3)이다.

ALU8 명령 프래그먼트에 의해 수행된 특정 동작들이 표 16 에 설명되는 명령 프래그먼트로 이루어지는 8비트의 값들로 명기되어 있다.

표 16

7	6	5	4	3	2	1	0
0	ALUOp			LS		LD	
0	1	1	Sign	LS		LT	
1	LD		Sign	LS		LT	

ALU8 명령 프래그먼트 동작

ALU8 명령 프래그먼트를 이용하여 수행된 동작들을 명기하기 위해 이용된 특정 코드들이 표 17 에 설명되어 있다.

표 17

ALUOp	
000	LD=DETNORM(LS)
001	LD=SET(LS)
010	LD=LS<<SR
011	LD=RND(LS<<SR)
100	LD=LD + (LS<<SR)
101	LD=LD - (LS<<SR)
110	LS+LT
111	LS-LT

LD / LS / LT	
00	L0
01	L1
10	L2
11	L3

Sign	
0	[LD=] LS+LT
1	[LD=] LS-LT

ALU8 명령 프래그먼트 동작

주: 올(all)-제로 명령 L0=DETNORM(L0) 은 NOP로 디코드된다.

LD=DETNORM(LD)인 명령은 허용되지 않는다.

ALU8 클리어(clear) 어큐뮬레이터 명령은 LD=LD-LD이다.

ALU16 명령 프래그먼트는 산술과 논리 시프트를 둘다 허용한다. ALU16 명령 프래그먼트에 의해 수행된 특정 동작들이 표 18 에 설명되어 있다.

<< 표시는 산술 시프트를 나타내고, <<< 는 논리 시프트를 나타낸다.

표 18

(a) *IF cond NOP*;	조건 NOP(병렬 조합용)
(b) *IF cond LD = SET(LS)*;	조건부로 Accumulator를 복사
(c) *IF cond LD = LS ± LT*;	조건부로 Accumulator를 가산/감산
(d) *LD = NORM(LS,SR)*;	Accumulator를 정규화
(e) *LD = ABS(LS)*;	Accumulator의 절대값
(f) *LD = −LS*;	Accumulator를 부정으로 함
(g) *LD=~LS*;	Accumulator를 인버트시킴(1의 보수)
(h) *LD = BIT(immediate5)*;	비트 마스크 신설 (LD = 0x1<<imm5)
(i) *LD=~BIT(immediate5)*;	인버트된 비트 마스크를 신설 (LD=~(0x1<<imm5))
(j) $LD = LS\left\{\begin{matrix}\& \\ \mid \\ \wedge\end{matrix}\right\}\left\{\begin{matrix}LT \\ BIT(immediate5) \\ \sim BIT(immediate5)\end{matrix}\right\}$;	비트와이즈 AND, OR, XOR
(k) $LD = [RND]\left(LS\left\{\begin{matrix}<< \\ <<<\end{matrix}\right\}\left\{\begin{matrix}R0-R3 \\ immediate6\end{matrix}\right\}\right)$;	Accumulator를 시프트(및 라운드)
(l) $LD = LS \pm (LT\left\{\begin{matrix}<< \\ <<<\end{matrix}\right\} immediate6)$;	Accumulator를 시프트하고 가산함
(m) $\left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} = \left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} \pm \{R0-R7\}$;	레지스터 가산
(n) $\left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} = \left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} + immediate6$	Immediate 가산
(o) $\left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} = SET(immediate6)$;	Immediate 로드
(p) $\left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} = \left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\}\left\{\begin{matrix}\& \\ \mid \\ \wedge\end{matrix}\right\}\{R0-R7$	16비트 논리
(q) $\{R0-R3\} = SR \pm \{R0-R7\}$;	레지스터를 SR에 가산
(r) $\{R0-R3\} = SR + immediate6$;	SR에 Immediate 가산
(s) $SR = \{R0-R3\} + immediate6$;	Immediate 합으로 SR을 로드
(t) $SR = SET(immediate6)$;	SR을 Immediate 로드

ALU16 명령 프래그먼트 동작

명령 LO=SET(LO) 은 NOP로 디코드된다.

ALU16 명령 프래그먼트의 포맷이 표 19 에 개시된다.

표 19

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(l)	0	0	+/-	AL	LT		LS		LD		immediate6					
(c)	0	1	0	0	LT		LS		LD		Cop1	cond				
(k)	0	1	0	1	Sop2	AL	LS		LD		immediate6					
(j)	0	1	1	BitOp		Inv	LS		LD		immediate6					
(p)	0	1	1	1	1	BitOp		dregh(dst)			dregh(src)			R0-R7 (src)		
(o)	0	1	1	1	1	1	1	dregh(dst)			immediate6					
(n)	1	0	0	0	dregh(src)			dregh(dst)			immediate6					
(m)	1	0	0	1	0	0	+/-	dregh (dst)			dregh(src)			R0-R7 (src)		
	1	0	0	1	0	1	Reserved (all zeros)									
	1	0	0	1	1	Reserved (all zeros)										
	1	0	1	Reserved (all zeros)												
(b)	1	1	0	0	0	0	LS		LD		Cop2	cond				
(h-i)	1	1	0	0	0	1	0	Inv	LD		immediate6					
(r)	1	1	0	0	0	1	1	0	R0-R3 (d)		immediate6					
(s)	1	1	0	0	0	1	1	1	R0-R3 (s)		immediate6					
(k)	1	1	0	0	1	0	0	0	R0-R3 (s)		Sop1	AL	LS		LD	
(j)	1	1	0	0	1	0	0	1	BitOp		LT		LS		LD	
(d-g)	1	1	0	0	1	0	0	1	1	1	AccOp		LS		LD	
(t)	1	1	0	0	1	0	1	0	0	0	immediate6					
(q)	1	1	0	0	1	0	1	0	0	1	+/-	R0-R3 (d)		R0-R7 (src)		
	1	1	0	0	1	0	1	0	1	Reserved (all zeros)						
	1	1	0	0	1	0	1	1	Reserved (all zeros)							
	1	1	0	0	1	1	Reserved (all zeros)									
	1	1	0	1	Reserved (all zeros)											
	1	1	1	Reserved (all zeros)												

16 비트 ALU 명령 프래그먼트 포맷 및 코드

ALU16 명령 프래그먼트에 의해 수행되는 특별한 동작들이, 표 20 에 개시된 것과 같이 명령 프래그먼트를 구성하는 비트들의 값들로서 구체화된다.

표 20

Dregh	
000	R0
001	R1
010	R2
011	R3
100	L0h
101	L1h
110	L2h
111	L3h

BitOp	
0 0	AND
0 1	OR
1 0	XOR

Sop1	
0	LD=LS<<R0-R3
1	LD=RND(LS<<R0-R3)

Sop2	
0	LD=LS<<imm6
1	LD=RND(LS<<imm6)

AL	
0	Arithmetic Shift
1	Logical Shift

AccOp	
0 0	LD=NORM(LS,SR)
0 1	LD=ABS(LS)
1 0	LD=-LS
1 1	LD=~LS

Cop1	
0	LD=LS+LT
1	LD=LS-LT

Cop2	
0	LD=SET(LS)
1	Reserved

+/-	
0	+
1	-

Inv	
0	Normal bitmask
1	Inverse bitmask

주 : 올 제로 (all zeros) 명령 L0=SET(L0) 이 NOP로서 디코드된다.

BIT 명령 (h,i,j) 에 있어, 어셈블러는 제로 신호를 immediate5 에 부가함으로써 immediate6 을 인코드한다(이는 디코딩을 단순화시킨다).

표 20. ALU16 명령 프래그먼트 코드

DMOV16 명령 프래그먼트는 표 21에서 개시된 바와 같은 데이터 입력포트 및 데이터 출력포트 동작들의 상이한 데이터 무브 (move)를 수행하기 위한 16 비트 명령 프래그먼트이다.

표 21

(a)	*NOP*;	병렬 명령들을 위한 NOP
(b)	$LC = immediate9$;	루프 카운터 Immediate의 로드
(c)	$\left\{\begin{matrix} AM0-AM1 \\ BM0-BM1 \\ CM0-CM1 \end{matrix}\right\} = immediate10$;	수정 레지스터 Immediate의 로드
(d)	$\left\{\begin{matrix} AL0-AL1 \\ BL0-BL1 \end{matrix}\right\} = immediate11$;	순환길이 레지스터 Immediated의 로드
(e)	$\left\{\begin{matrix} L0-L3 \\ D0-D3 \\ L0h-L3h \end{matrix}\right\} = INPORT(port\_addr)$;	입력부 동작
(f)	$OUTPORT(port\_addr) = \left\{\begin{matrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{matrix}\right\}$	출력부 동작
(g)	*OUTPORTA*(*port_addr*);	버스 A상에서의 출력부 값
(h)	*OUTPORTB*(*port_addr*);	버스 B상에서의 출력부 값

DMOV16 명령 프래그먼트를 유용하게 이용하는 동작들을 수행하는 데 사용되는 포맷 및 코드들이 표 22 에 개시된다.

표 22

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	Reserved (all zeros)												
(d)	0	0	1	0	AL	immediate11 (im1)										
(d)	0	0	1	1	BL	immediate11 (im2)										
(c)	0	1	0	0	0	AM	immediate10 (im1)									
(c)	0	1	0	0	1	BM	immediate10 (im2)									
(c)	0	1	0	1	0	CM	immediate10 (im3)									
	0	1	0	1	1	Reserved (all zeros)										
(b)	0	1	1	0	0	0	0	immediate9 (im1)								
	0	1	1	0	0	0	1	Reserved (all zeros)								
	0	1	1001-1111					Reserved (all zeros)								
(e)	1	0	0	0	0	0	L0-L3	Inport address (PI3)								
(e)	1	0	0	0	0	1	D0-D3	Inport address (PI3)								
(e)	1	0	0	0	1	0	L0h-L3h	Inport address (PI3)								
	1	0	0011-1111					Reserved (all zeros)								
(f)	1	1	0	0	dreg			Outport address (Abus, PO1)								
(f)	1	1	0	1	dreg			Outport address (Bbus, PO2)								
(g)	1	1	1	0	0	0	0	0	Outport address (Reads A bus)							
(h)	1	1	1	0	0	0	0	1	Outport address (Reads B bus)							
	1	1	10001-11111						Reserved (all zeros)							

주 : 명령 LC=0 은 NOP로서 디코드된다.

immediate10 이 표시되고 immediate11 이 표시되지 않는다.

표 22. DMOV16 명령 프래그먼트 포맷

명령 OUTPORTA(port_addr) 는 Abus 상의 값을 판독하고, 그 값을 지정된 포트로 출력시킨다. 동시에 메모리 A 로부터 값을 판독함으로써, 명령은 값을 직접적으로 메모리 A에서 포트로 보내는데 사용될 수 있다. OTUPORTB(port_addr) 도 유사하게 동작한다.

DMOV24 명령 프래그먼트는 표 23에 개시된 것과 같은 상이한 로드/저장 레지스터 다이렉트 (load/store register direct) 또는 로드 레지스터 이미디어트 (load register immediate) 동작들을 수행하기 위한 24 비트 명령 프래그먼트이다.

표 23

(a)	$\{regA\} = memA(address14);$	
(b)	$memA(address14) = \{regA\};$	L/S direct memory A
(c)	$\{regB\} = memB(address14);$	
(d)	$memB(address14) = \{regB\};$	L/S direct memory B
(e)	$\{regC\} = memC(address14);$	
(f)	$memC(address14) = \{regC\};$	L/S direct memory C
(g)	$\left\{\begin{matrix} A0-A7 \\ B0-B7 \\ C0-C3 \end{matrix}\right\} = address14;$	Address 레지스터 Immediate의 로드
(h)	$\left\{\begin{matrix} AS0-AS1 \\ BS0-BS1 \end{matrix}\right\} = address14;$	순환시작 레지스터 Immediate의 로드
(i)	$\left\{\begin{matrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{matrix}\right\} = immediate16;$	데이터 레지스터 Immediate의 로드
(j)	$LOOP\ UNTIL\ address17;$	Loop until end address.
(k)	$CALL\ address17;$	Function Call.
(l)	$[IF\ cond]\left\{\begin{matrix} JUMP \\ JUMPD \end{matrix}\right\} address17;$	[Conditional] [Delayed] Jump.

시작 레지스터들은 AGU 장치에 위치된다.

표 23. DMOV24 명령 프래그먼트 동작

표 24 는 본 발명의 바람직한 실시예에 따르는 DMOV24 명령을 유용하게 이용하는 다양한 동작을 수행하는데 사용되는 포맷 및 몇몇 코드들을 제공한다.

표 24

	23	22	21	20	19	18	17	16	15	14	13-0
(l)	0	0	cond					address17 (JUMP)			
(l)	0	1	cond					address17 (JUMPD)			
(j)	1	0	0	0	0	0	0	address17 (LOOP)			
(k)	1	0	0	0	0	0	1	address17 (CALL)			
(g)	1	0	0	0	0	1	0	A0-A7			address14 (im1)
(g)	1	0	0	0	0	1	1	B0-B7			address14 (im2)
(e)	1	0	0	0	1	0	regC (dst)				address14 (im3, Cbus+PI3)
(f)	1	0	0	0	1	1	regC (src)				address14 (im3, Cbus+PO3)
(a)	1	0	0	1	0	regA (dst)					address14 (im1, Abus+PI1)
(b)	1	0	0	1	1	regA (src)					address14 (im1, Abus+PO1)
(c)	1	0	1	0	0	regB (dst)					address14 (im2, Bbus+PI2)
(d)	1	0	1	0	1	regB (src)					address14 (im2, Bbus+PO2)
(i)	1	0	1	1	dreg			immediate16 (im1+PI1)			
(i)	1	1	0	0	dreg			immediate16 (im2+PI2)			
(g)	1	1	0	1	0	0	0	0	C0-C3		address14 (im3)
(h)	1	1	0	1	0	0	0	1	0	AS	address14 (im1)
(h)	1	1	0	1	0	0	0	1	1	BS	address14 (im2)
	1	1	0	1001-1111				Reserved (all zeros)			
	1	1	1	Reserved (all zeros)							

주 : Address14 및 Address17 은 부화화되지 않고 (unsigned), immediate16 은 부호화된다 (signed).

표 24. DMOV24 명령 프래그먼트 포맷 및 코드

다른 명령 프래그먼트들 뿐만 아니라 DMOV24에 있어서, 어떤 동작들은 두 번 인코드된다는 것에 주목하여야 한다. 예를 들어, 행 (i) 및 (j)에서 지정된 형태들은 동일한 동작을 인코드하고, 하나는 immediate

bus (Im1) 의 사용을 특정하고 다른 하나는 immediate bus (Im2) 의 사용을 특정한다. 두 번의 인코딩은 명령 프래그먼트가 매우 다양한 다른 명령 프래그먼트들로 결합되게 하고, 이는 Immediate bus 1 은 물론 Immediate bus 2 의 사용을 요할 수 있다.

40-bit 듀얼 로드 명령 프래그먼트 (DL40) 은 immediate load 또는 address load 동작들을 수행하기 위한 40 비트 명령 프래그먼트이다. 본 발명의 바람직한 실시예에서 수행되는 특별한 동작들이 표 25 에 도시된다.

표 25

(a)	$\{A0-A3\} = address14, \{B0-B3\} = address14;$
(b)	$\{A0-A3\} = address14, \{C0-C3\} = address14;$
(c)	$\{B0-B3\} = address14, \{C0-C3\} = address14;$
(d)	$\left\{\begin{matrix} A0-A3 \\ B0-B3 \\ C0-C3 \end{matrix}\right\} = address14, \left\{\begin{matrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{matrix}\right\} = immediate16;$
(e)	$\left\{\begin{matrix} L0-L3 \\ D0-D3 \end{matrix}\right\} = immediate32;$
(f)	$\{A0-A3\} = address14, regB = memB(address14);$
(g)	$\{B0-B3\} = address14, regA = memA(address14);$
(h)	$\{A0-A3\} = address14, regC = memC(address14);$
(i)	$\{C0-C3\} = address14, regA = memA(address14);$
(j)	$\{B0-B3\} = address14, regC = memC(address14);$
(k)	$\{C0-C3\} = address14, regB = memB(address14);$
(l)	$regA = memA(address14), regB = memB(address14);$
(m)	$regA = memA(address14), regC = memC(address14);$
(n)	$regB = memB(address14), regC = memC(address14);$

표 25. DL40 명령 프래그먼트 포맷

각 동작을 위한 DL40 명령 프래그먼트의 형태는 표 26에서 제공된다.

표 26

	39	38	37	36	35	34	33	32	31	30	29-16	15-14	13-0
	0	0	0	0	0	Reserved							
(e)	0	0	0	0	1	Lreg/Dreg			immediate32				
(d)	0	0	0	1	A0-A3		dreg		address14			immediate16	
(d)	0	0	1	0	B0-B3		dreg		address14			immediate16	
(d)	0	0	1	1	C0-C3		dreg		address14			immediate16	
(b)	0	1	0	0	0	0	0	0	C0-C3		Caddress14	A0-A3	Aaddress14
(c)	0	1	0	0	0	0	0	1	C0-C3		Caddress14	B0-B3	Baddress14
(a)	0	1	0	0	0	0	1	0	A0-A3		Aaddress14	B0-B3	Baddress14
	0	1	0	0	0	0	1	1	Reserved (all zeros)				
	0	1	0	0	0	1	Reserved (all zeros)						
(h)	0	1	0	0	1	0	regC				Caddress14	A0-A3	Aaddress14
(j)	0	1	0	0	1	1	regC				Caddress14	B0-B3	Baddress14
(g)	0	1	0	1	0	regA					Aaddress14	B0-B3	Baddress14
(i)	0	1	0	1	1	regA					Aaddress14	C0-C3	Caddress14
(f)	0	1	1	0	0	regB					Baddress14	A0-A3	Aaddress14
(k)	0	1	1	0	1	regB					Baddress14	C0-C3	Caddress14
	0	1	1	1	Reserved								
(m)	1	0	0	regC			re...				Caddress14	..gA	Aaddress14
(n)	1	0	1	regC			re...				Caddress14	..gB	Baddress14
(l)	1	1	regA				re...				Aaddress14	..gB	Baddress14

주 : Address14 는 부호화되지 않고 (unsigned), immediate16 및 immediate32 는 부호화된다 (signed).

표 26. DL40 명령 프래그먼트 포맷 및 코드

표 5 에서도 도시된 바와 같이, 메모리 무브 및 프로그램 플로우 프래그먼트들의 4가지 유형들이 본 발명의 바람직한 실시예에서 제공되며, 그 리스트는 표 27에서 제공된다.

표 27

OneMem11
TwoMem19
TwoMov19
ThreeMem27

표 27. 메모리 무브 및 프로그램 플로우 명령 프래그먼트

각 메모리 무브 및 프로그램 플로우 명령 (MMPF) 프래그먼트는 표 28 에 열거된 MMPF 서브프래그먼트들의 집합으로 구성된다.

표 28

	명령 서브프래그먼트들	
LD(A)	메모리 A 인다이렉트의 로드	8
ST(A)	메모리 A 인다이렉트의 저장	8
LD(B)	메모리 B 인다이렉트의 로드	8
ST(B)	메모리 B 인다이렉트의 저장	8
LS(C)*	메모리 C 인다이렉트의 로드/저장	8
DMOVA*	버스 A 레지스터 이동	8
DMOVB*	버스 B 레지스터 이동	8
PF8	8-비트 프로그램 플로우	8

표 28. 조합 데이터 무브 및 프로그램 플로우 명령 서브프래그먼트

MMPF 명령 프래그먼트들의 포맷 및 동작이 우선 설명되고, MMPF 서브프래그먼트들의 포맷 및 동작의 보다 상세한 설명이 뒤따를 것이다.

OneMem11 MMPF 명령 프래그먼트가 단일 메모리 로드 및 저장 동작들, 데이터 무브 동작들 및 프로그램 플로우 동작들을 수행하는데 사용된다. 여기에 제공된 바람직한 실시예로서, 8가지의 상이한 동작들이 표 29에 도시된 것과 같은 11 비트 프래그먼트 중에서 제 1에서 제 3 비트들에 의해 지시된 특정한 동작과 함께, OneMem11 MMPF 명령 프래그먼트를 사용하여 수행되며, 표 29 는 OneMem11 데이터 무브 명령 프래그먼트를 사용하여 수행될 수 있는 동작들을 열거한다.

표 29

10	9	8	7	6	5	4	3	2	1	0
0	0	0	LD(A)							
0	0	1	ST(A)							
0	1	0	LD(B)							
0	1	1	ST(B)							
1	0	0	LS(C)							
1	0	1	DMOVA							
1	1	0	DMOVB							
1	1	1	PF8							

표 29. OneMem11 명령 프래그먼트 포맷

TwoMem19 MMPF 명령 프래그먼트는 표 30 에 개시된 것과 같이 수행되는 메모리 로드 및 저장 동작들의 8개의 상이한 조합들을 가능케하는 19 비트 명령 프래그먼트이다.

표 30

18	17	16	15-8	7-0
0	0	0	LD(A)	LD(B)
0	0	1	LD(A)	ST(B)
0	1	0	LD(A)	LS(C)
0	1	1	ST(A)	LD(B)
1	0	0	ST(A)	ST(B)
1	0	1	ST(A)	LS(C)
1	1	0	LS(C)	LD(B)
1	1	1	LS(C)	ST(B)

TwoMem19 명령 프래그먼트 포맷

TwoMov19 MMPF 명령 프래그먼트는 표 31에 도시된 바와 같이 데이터 무브 동작들에 따른 메모리 로드 및 기억 동작들의 8 개의 상이한 결합을 허용하는 19 비트 명령 프래그먼트이다.

표 31

18	17	16	15-8	7-0
0	0	0	LD(A)	DMOVB
0	0	1	ST(A)	DMOVB
0	1	0	DMOVA	LD(B)
0	1	1	DMOVA	ST(B)
1	0	0	DMOVA	LS(C)
1	0	1	LS(C)	DMOVB
1	1	0	DMOVA	DMOVB
1	1	1	보존	

TwoMev19 명령 프래그먼트 포맷

ThreeMem27 MMPF 명령 프래그먼트는 표 32에서 도시된 바와 같이 수행되는 데이터 동작들, 메모리 기억, 메모리 로드에 대한 8 개의 상이한 결합을 허용하는 27 비트 명령 프래그먼트이다.

표 32

26	25	24	23-16	15-8	7-0
0	0	0	LS(C)	LD(A)	LD(B)
0	0	1	LS(C)	LD(A)	ST(B)
0	1	0	LS(C)	ST(A)	LD(B)
0	1	1	LS(C)	ST(A)	ST(B)
1	0	0	LS(C)	DMOVA	LD(B)
1	0	1	LS(C)	DMOVA	ST(B)
1	1	0	LS(C)	LD(A)	DMOVB
1	1	1	LS(C)	ST(A)	DMOVB

ThreeMem27 명령 프래그먼트 포맷

수학식 4 는 LD(A) 명령 서브프래그먼트에 의해 수행되는 동작들을 제공한다.

수학식 4

$$\begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix} = *AX \begin{bmatrix} ++ \\ -- \\ ++AM0 \\ ++AM1 \end{bmatrix}$$

표 33은 본 발명의 예시적인 실시예에 따른 LD(A) 명령 서브프래그먼트의 포맷을 제공한다.

표 33

7	6	5	4	3	2	1	0
dreg				A0-A3		Amod	

LD(A) 명령 서브프래그먼트 포맷

수학식 5는 LD(B) 명령 서브프래그먼트에 의해 수행되는 동작들을 제공한다.

수학식 5

$$\begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix} = *BX \begin{bmatrix} ++ \\ -- \\ ++BM0 \\ ++BM1 \end{bmatrix}$$

표 34는 본 발명의 예시적인 실시예에 따른 LD(B) 명령 서브프래그먼트의 포맷을 제공한다.

표 34

7	6	5	4	3	2	1	0
dreg				B0-B3		Bmod	

LD(B) 명령 서브프래그먼트 포맷

수학식 6 은 ST(A) 명령 서브프래그먼트에 의해 수행되는 동작들을 제공한다.

수학식 6

$$*AX \begin{bmatrix} ++ \\ -- \\ ++AM0 \\ ++AM1 \end{bmatrix} = \begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix}$$

표 35는 본 발명의 예시적인 실시예에 따른 ST(A) 명령 서브프래그먼트의 포맷을 제공한다.

표 35

7	6	5	4	3	2	1	0
dreg				A0-A3		Amod	

ST(A) 명령 서브프래그먼트 포맷

수학식 7은 ST(B) 명령 서브프래그먼트에 의해 수행되는 동작들을 제공한다.

수학식 7

$$*BX\begin{bmatrix} ++ \\ -- \\ ++BM0 \\ ++BM1 \end{bmatrix} = \begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix}$$

표 36 은 본 발명의 예시적인 실시예에 따른 ST(B) 명령 서브프래그먼트의 포맷을 제공한다.

표 36

7	6	5	4	3	2	1	0
dreg				B0-B3		Bmod	

ST(A) 명령 서브프래그먼트 포맷

표 37 은 DMOVA 명령 서브프래그먼트에 의해 수행되는 동작들을 나타낸다.

표 37

(a)	*NOP*;	병렬 명령에 대한 NOP
(b)	*TLOOP*;	Tight 루프(단일명령 루프)
(c)	$\{R0-R7\} = \begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix}$;	데이터 레지스터 이동
(d)	$\{R0-R7\} = \{A0-A3\}$;	어드레스 레지스터를 데이터 레지스터로 이동
(e)	$\begin{Bmatrix} A0-A3 \\ AM0-AM1 \end{Bmatrix} = \{R0-R3\}$;	데이터 레지스터를 AGU 레지스터로 이동
(f)	$\{A0-A3\} = \{A0-A3\}$; $\{B0-B3\} = \{B0-B3\}$; $\{C0-C3\} = \{C0-C3\}$;	어드레스 레지스터 이동
(g)	*IF cond*	병렬 명령의 상태

DMOVA 명령 서브프래그먼트 포맷

표 38은 본 발명의 예시적인 실시예에 따른 DMOVA 명령 서브프래그먼트의 포맷을 제공한다.

표 38

	7	6	5	4	3	2	1	0
(a,b,c)	0	R0-R7 (dst)			dreg (src)			
(d)	1	0	0	0	A0-A3 (src)		R0-R3 (dst)	
(e)	1	0	0	1	A0-A3 (dst)		R0-R3 (src)	
(f)	1	0	1	0	A0-A3 (dst)		A0-A3 (src)	
(f)	1	0	1	1	B0-B3 (dst)		B0-B3 (src)	
(f)	1	1	0	0	C0-C3 (dst)		C0-C3 (src)	
(d)	1	1	0	1	A0-A3 (src)		R4-R7 (dst)	
(g)	1	1	1	cond				

주해 : 명령 R0=R0는 NOP로 복호화
명령 R1=R1은 TLOOP로 복호화
명령 A0=A0는 AM0=R0로 복호화
명령 A1=A1은 AM0=R1으로 복호화
명령 A2=A2는 AM0=R2로 복호화
명령 A3=A3는 AM0=R3로 복호화
명령 B0=B0는 AM1=R0로 복호화
명령 B1=B1은 AM1=R1으로 복호화
명령 B2=B2는 AM1=R2로 복호화
명령 B3=B3는 AM1=R3로 복호화

DMOVA 명령 서브프래그먼트 포맷

이와 같이, 하나 이상의 명령 서브프래그먼트를 포함할 수 있는 MMPF 명령 프래그먼트들을 제공함에 의해, 전 명령을 사용하여 수행될 수 있는 동작들의 수는 더 증가된다. 예를 들어, 전 명령은 산술 및 MAC 동작들이 일군의 3개의 메모리 무브 및 프로그램 플로우 동작들에 따라 수행되게 할 수 있다. 단일 명령을 사용하여 이러한 많은 동작들을 수행하는 능력은 소정의 동작을 수행할 필요가 있는 전 명령들의 수를 더 감소시켜 DSP 상에서 요청되는 전 명령 메모리를 감소시킨다. 명령 메모리를 감소시키면 다이(die)크기, DSP의 비용 및 소비 전력이 감소되어, 이러한 DSP를 이동 무선 전화를 포함하는 많은 다양한 애플리케이션에 더 적합하게 한다.

이와 같이, 고도의 평행 변수 길이 명령 집합을 사용하여 DSP를 제어하는 시스템 및 방법이 기술되었다. 임의의 상기 기술에 숙련된 자들에게 본 발명을 제작 또는 사용할 수 있는 바람직한 실시예에 대한 설명이 이전에 제공되었다. 이러한 실시예들에 대한 다양한 변경들은 용이하게도 상기 기술에 숙련된 자들에게 명백하며, 본원에서 규정된 일반적인 원리들은 본 발명의 기능을 사용하지 않고 다른 실시예들에 적용될 수 있다. 예를 들어, 상기 시스템 및 방법이 DSP의 내용에 기술되지만, 그것의 다양한 특성들은 일반적인 컴퓨팅 시스템 및 장치들에 적용가능하다.

이와 같이 바람직한 실시예들를 참조하여 본 발명이 기술되었으므로, 당면된 실시예들은 단지 예시적이며 적절한 지식 및 기술을 가진 자들에게 일어날 수 있는 이와 같은 그러한 변경 및 수정이 첨부된 청구범위 및 그와 동등한 것에 기술된 바와 같이 본 발명의 사상 및 범위를 이탈하지 않고 행해질 수 있다.

전 명령 포맷 (full instruction format)

이에 더하여, 로테이터 (308) 의 구성은, 표 2 에 표시된 바와 같이 코딩된 제어 (320) 을 이루는 2 개의 선택 비트 (S1 및 S0) 에 의해서 제어된다.

표 2

S1 S0	로테이터 출력
0 0	ABC
0 1	BCD
1 0	CDA
1 1	DAB

로테이터 선택 제어 비트 및 출력

분명하게 알 수 있는 바와 같이, S1 및 S0 의 상태가 증가함에 따라, 로테이터 (308) 의 출력은 좌측-회전 (left-rotated), 또는 배럴 시프트된다 (barrel- shift). 좌측-회전은 각각의 입력 그룹 (A, B, C 및 D) 가 출력 상에 좌측으로 시프트되도록 하는 것이다. 출력의 맨 좌측 위치에 있는 입력 그룹은 제거된다. 이전

에 출력에서 표명되지 않은 입력 그룹은 다음에 맨 우측 위치에서 출력된다.

S1 및 S1 의 상태와 그에 따른 로테이터 (308) 의 구성은 다양한 길이의 명령에 응답하여 변하는 양에 의해서 갱신되거나 회전된다. 특히, 처리되는 명령의 길이를 나타내는 값 (I1 I0) 은 제어 비트 (S1 및 S0) 에 더해지고, 어떤 실행 값 (carryout value) 도 버려진다 (discard). 즉,

수학식 1

S1(t+1),S0 (t+1) = S1(t),S0 (t) + I0, I1

브랜치 또는 리세트 조건에 대해서, S1 및 S0 의 값은 프로세싱이 브랜치되거나 리세트되는 특정 명령에 기초하여 리세트되어, 수학식 1 이 이용되지 않는다. 브랜치, 리세트 및 스톨 명령을 프로세싱하기 위한 다양한 방법들이 본 기술분야에서 잘 알려져 있고, 이 프로세싱은 본 발명과 특별히 관련되지 않기때문에 더이상 설명하지 않겠다.

예시적인 프로세싱에서, 로테이터 (308) 는 ABC 의 출력과 00 에서의 신호 비트 (S1 및 S0) 로 시작된다. 16 비트 명령이 수신되면, I1 및 I0 의 해당하는 명령 길이 비트가 S1 및 S0 에 더해져서, 01 의 S1 및 S0 을 산출하고, 이것은 BCD 의 로테이터 (308) 로부터의 출력 (324) 에 해당한다. BCD 의 출력은 처음 16 비트의 명령 데이타 (입력 A) 가 처리된 후에 다음 인 라인 세트의 명령이 된다.

다음 명령이 32 비트 명령이면, 명령 길이 (I1 및 I0) 는 01 인 현재 S1 및 S2 상태에 더해져서, 11 을 산출한다. 결과적인 출력은 DAB 가 되고, 이것은 처리되지 않은 다음 48 비트의 명령 데이타에 해당하여, 입력 D 상에 수신된 다음 인 라인 명령 데이타가 가장 중요한, 또는 맨 좌측의 위치에 위치하도록 구성된다. 그 이전에, 입력 (B 및 C) 로부터 명령 데이타가 이미 처리된 상태이다. 이전의 32 비트 명령의 처리동안에 새로운 데이타가 전술한 명령 데이타를 로딩하는 프로세스에 따라서 명령 레지스터 (307) 로 로드된다는 것에 유의하여야 한다.

48 비트 명령이 처리되면, 선택 비트 (S1 및 S0) 의 상태는 11 의 명령 길이 (I0 및 I1) 에 의해서 증가되어, 10 의 S1 및 S0 을 산출하고, 이것은 로테이터 (308) 를 출력 (CDA) 로 구성한다. CDA 의 출력은 처리될 다음 48 비트의 명령 데이타에 해당하고 다음 인 라인 비트는 가장 중요한 위치로 배치된다. 그 전에, 입력 (D, A 및 B) 상의 명령 데이타가 처리되었다. 이전의 명령이 처리되는 동안에 새로운 명령 데이타가 레지스터 (306) 으로 판독된다는 것에 또한번 다시 유의하여야 한다. 관련된 명령 데이타 프로세싱을 수행하는 전술한 논리는 실시예의 목적으로 제공된 것이며, 전술한 명령 데이타 프로세싱을 수행하는 다른 논리의 사용은 분명하고 본 발명의 사용과 양립한다.

따라서, 실시예에서, 제어 논리는 로테이터 (308) 를 명령 레지스터 (306 및 307) 로부터 수신된 다음 48 비트의 명령 데이타를 출력하도록 구성되어, 처리될 다음 명령 비트가 맨 좌측 위치에 위치하게 된다. 처리될 다음 세트의 명령 데이타가 확인될 수 있는 소정의 배치로 명령 데이타를 명령 프로세싱 부 (158) 에 제공하는 것은 또한 본 발명의 사용과 양립한다는 사실을 본 기술 분야의 당업자들은 알수 있다.

전술한 바와 같이, 처리되지 않은 다음 48 비트의 명령 데이타를 출력하는 것에 더하여, 로테이터 (308) 는 또한 처리될 다음 명령의 사이즈를 제어 논리 (304) 에 지시한다. 특히, 로테이터 (308) 는 처리될 다음 5 비트의 명령 데이타의 부가적인 복사를 논리 제어 (304) 에 출력한다. 본 발명의 바람직한 실시예에서, 명령의 길이는 명령의 처음 5 비트에 의해서 특정된다.

도 8 은 본 발명의 일 실시예에 따라 구성될 때의 MAC 부 (128) 의 블록도이다. 시프트 라이트 (shift right; 900) 는 누산될 40 비트 입력을 수신하고 0 또는 16 비트만큼 값을 시프트하여, 출력이 멀티플렉서 (901) 의 하나의 입력으로 인가된다. 멀티플렉서 (901) 의 다른 입력은 0 ×800 값을 수신한다. 곱셈기 (멀티플라이어; 902) 는 각 입력에 대한 총 17 비트에 대해 명령 디코더 (158) 로부터의 신호 비트와 함께 곱하여질 2 개의 16 비트 값을 수신한다.

곱셈기 (902) 의 출력은 시프트 레프트 (shift left; 904) 에 의해 수신되는데, 이것은 명령 디코더 (158) 에 의해서 특정되는 바와 같이, 0, 1, 2 또는 3 비트만큼 출력을 시프트한다. 가산회로/감산회로 (906) 는 명령 디코더 (158) 에 의해 지시되는 대로 2 개의 입력 값의 가산 또는 감산을 수행하고 그 결과를 출력하며, 이것은 본 발명의 일례에서의 레지스터 뱅크 입력 포트 (PI4) 로 인가된다.

본 발명의 실시예에서 MAC 부 (128) 내의 시프트 라이트 16 유닛 (900) 의 사용은 다은 형태의 MAC 부에 대한

추가적인 이용을 제공한다. 더 자세하게 말해서, 시프트 라이트 16 유닛 (900) 의 사용은 감소된 수의 클럭 사이클 에서의 2 배 정밀도 동작을 수행하는 것을 용이하게 한다. 예를 들어, 32 비트 수 (A) 가 16 비트 수 (B) 와 곱해지는 2 배 정밀도 동작을 수행하기 위해서, 32 비트 수의 아래 16 비트 (Al) 가 먼저 제 1 클럭 사이클 동안에 16 비트 수 (B) 와 곱해져서, 레지스터 뱅크 (120) 내에 저장된 중간값 (I) 을 산출한다. 제 2 클럭 사이클 동안에, 중간값 (I) 은 시프트 라이트 16 유닛 (900) 으로 입력되고 16 비트만큼 우측으로 시프트된다. 이에 더하여, 16 비트 수 (B) 와 32 비트 수 (A) 의 위 16 비트 (Ah) 가 곱해지고, 그 결과는 시프트 라이트 16 유닛 (900) 으로부터온 우측으로 시프트된 중간값 (I) 과 가산된다. 따라서, 2 배 정밀도 곱하기는 3 이라기 보다 2 개의 클럭 사이클에서 수행된다. 일반적으로, 많은 2 배 정밀도 동작은 하나 또는 그 이상의 변수가 다른 변수에 대해 시프트되도록 요구하며, 멀티플라이 또는 어큐뮬레이트 동작의 하나로서 동일한 클럭 사이클동안에 수행되는 시프팅 단계를 허용하는 것은 2 배 정밀도 동작을 수행하는 데 필요한 사이클의 수를 감소시킨다.

보통, 3 개 클럭 사이클이 요구되는데, 이는 제 1 곱하기 동작, 시프트 동작, 및 제 2 곱하기 동작 각각은 전형적으로 클럭 사이클이 요구되기 때문이다. 따라서, 시프트 회로의 사용은 2 배 정밀도 곱하기를 수행하는 데 요구되는 클럭의 수를 감소시킨다. 다른 크기의 피연산자 (operands) 에 관련된 다른 2 배 정밀도 동작은 또한, 시프트 라이트 부 (900) 의 사용에 의해서 용이하게 된다.

Ⅱ. 명령 구성 (INSTRUCTION CONFIGURATION)

A. 개관.

도 9 는 본 발명의 실시예에서 사용된 명령 체계를 설명하는 블록도이다. 블록 (402) 은 DSP 의 동작을 제어하는 16, 32 또는 48 비트로 이루어진 가변 길이 전 명령 (variable length full instruction)을 나타낸다. 가변 길이 명령은 블록 (403) 에 나타난 바와 같이, 메모리 무브 및 프로그램 플로우 (memory move and program flow) MMPE 와 일반 명령 프래그먼트를 포함하는 명령 프래그먼트로 구성된다. 본 발명의 실시예에서 사용된 일반 명령 프래그먼트는 MAC8, MAC16, ALU8, ALU16, DMOV16, DMOV24 및 DL40 명령 프래그먼트를 포함한다. MMPF 명령 프래그먼트는 OneMem11, TwoMem19, TwoMov19 및 TreeMem24 명령 프로그래먼트를 포함한다. MMPF 명령 프래그먼트는 블록 (406) 에 나타난 MMPF 명령 서브프래그먼트로 구성된다. MMPF 명령 서브프래그먼트는 LD(A), LD(B), ST(A), ST(B), LS(C), DMOVA, DMOVB, 및 PF8 을 포함한다. 다양한 전 명령, 명령 프래그먼트 및 명령 서브프래그먼트는 아래에서 더 자세히 설명되어있다.

B. 전 명령 (Full Instruction)

본 발명의 바람직한 실시예에서, DSP 는 16, 32 및 48 비트의 길이를 갖는 전 명령을 사용하여 제어된다. 전 명령은 하나 또는 그 이상의 명령 프래그먼트를 조합하여 형성된다. 전 명령은 명령 메모리 (152) 내에서의 연속적인 저장 및 DSP 에 의한 프로세싱을 감안하도록 구성된다. 전 명령의 포맷 (format) 및 구성은 아래 설명되어있고, 그 다음에 명령 프래그먼트의 포맷과 구성이 뒤따른다. 동작동안, DSP 는 전 명령의 각각의 클럭 사이클을 처리한다. 따라서, 곱하기 동작은, 개별적인 동작이 개별적인 세트의 선택된 명령 프래그먼트에 의해 결정되면서, 각각의 전 명령의 프로세싱동안에 수행될 수 있다.

본 발명의 실시예에서 사용된 3 개의 전 명령의 포맷은 표 3 에 나타나 있다.

표 3

전 명령 포맷

각각의 전 명령에 사용된 5 비트 헤더는 전 명령의 길이를 지시하고, 어떤 추가적인 포맷은 전체 포맷의 내용에 관한 것이다. 본 발명의 실시예에서 사용된 헤더의 포맷은 표 4 에 제공되어 있다.

표 4

5비트 헤더					명령 길이
0	0	0	0	X	16비트 명령 (2 타입;2 types)
0	0	0	1	X	32비트 명령 (2 타입)
0	0	1	X	X	48비트 명령 (4 타입)
0	1	X	X	X	32비트 명령 (8 타입)
1	X	X	X	X	48비트 명령 (16 타입)

전 명령 헤더 포맷

각각의 전 명령 (16, 32 및 48 비트 길이) 은 하나 또는 그 이상의 명령 프래그먼트를 포함한다. 표 5 는 본 발명의 실시예에서의 이용 가능한 명령 프래그먼트의 목록을 제공한다. 명령 프래그먼트의 포맷 및 동작의 더 자세한 설명은 전 명령에 대한 논의 후에 제공된다.

표 5

필드(Field)	설명	폭(Width)
MAC8*	8비트 MAC 동작	8
ALU8*	8비트 ALU 동작	8
OneMem11*	1 메모리 동작	11
MAC16*	16비트 MAC 동작	16
ALU16*	16비트 ALU 동작	16
DMOV16*	조건부 Reg 무브(Conditional Reg Move)/Inport/Outport	16
TwoMem19	2 메모리 동작	19
TwoMov19*	2 메모리/데이타 무브 동작	19
DMOV24	로드(Load)/Store Direct/Load Addr/Jump	24
TreeMem27	3 메모리 동작	27
DL40	듀얼 로드(Dual-Load)	40

* 는 필드가 어떤 비트 패턴을 사용하여 놉 (nop'ed; 무연산) 될 수 있다는 것을 지시한다.

표 5. 명령 프래그먼트

표 6 - 8 은 본 발명의 실시예에 따른 48, 32 및 16 비트 전 명령 내에서 사용될 수 있는 명령 프래그먼트의 다양한 조합을 제공한다. 명령 프래그먼트의 다른 조합이 본 발명의 사용 및 동작과 양립하지만, 여기에 개시된 조합의 일정한 특징들은 아래 설명하고 있는 바와 같이 바람직하다. 이에 더하여, 전 명령의 전부 또는 일부가 "예약된 (reserved)" 이라고 나타나 있는 곳에서는, 설명된 실시예에서 개별적인 명령 조합이 특정되거나 사용되지 않았지만, 이러한 전 명령 조합의 추후의 사용이 예상된다.

표 6 은 본 발명의 실시예에 따라 수행될 때 16비트 전 명령 포맷을 제공한다. 전 명령은 11개의 명령 비트가 뒤따르는 5개의 비트 헤더로 구성된다.

표 6

11											0
헤더					11 명령 비트						
0	0	0	0	0	MAC8			0	0	0	
0	0	0	0	0	ALU8			0	0	1	
0	0	0	0	0	Reserved			0	1	0	
0	0	0	0	0	Reserved			0	1	1	
0	0	0	0	0	Reserved			1	0	0	
0	0	0	0	0	Reserved			1	0	1	
0	0	0	0	0	Reserved			1	1	0	
0	0	0	0	0	Reserved			1	1	1	
0	0	0	0	1	OneMem11						

16비트 전 명령 포맷

주 : 16비트 NOP 는 MAC8 의 NOP 를 이용하여 얻어진다.

헤더 비트는 명령의 타입에 대한 일부 정보 뿐만 아니라 명령 길이를 나타낸다. 00000 의 헤더용으로, 3 개의 최하위 테일 비트는 수행되는 동작을 더 특정화하는 데 사용된다. 특히, 000 의 테일 비트는 나머지 8개 비트가 MAC8 명령 프래그먼트를 포함하는 것을 나타낸다. 001 의 테일 비트는 나머지 8개 비트가 ALU8 명령 프래그먼트를 포함하는 것을 나타낸다. 다른 테일 비트 조합을 위하여, 어떠한 명령도 특정화되지 않는다. 가장 짧은 명령은 저장하기에 최소량의 메모리를 요구하기에, 상기한 16비트 전 명령 사용은 특정한 집합의 동작을 수행하는 데 필요한 명령 메모리 양을 감소시킨다. 따라서, DSP 의 전체 크기 및 이에 따른 비용 및 전력 소모가 또한 감소된다.

16비트 명령은 특히 한 개 또는 감소된 개수의 동작이 수행될 수 있는 상태일 때 사용된다. 특히, 한 개의 동작만을 특정화하기에 필요한 명령 크기가 감소될 수 있고, 따라서 절반 워드 사용, 또는 한 개 동작을 수행하기 위한 16비트 명령이 감소될 수 있다. 또한, 16비트 명령은, 수행될 것으로 예상되는 거의 모든 동작을 포함하는 MAC, ALU, 메모리 이동 또는 프로그램 흐름 동작용으로 사용될 수 있다.

표 7 은 본 발명의 실시예에 따라 구성될 때 명령 프래그먼트 조합 및 32 비트 전 명령의 관련된 포맷을 도시한다.

표 7

27					19								11								0
헤더					27 명령비트																
0	0	0	1	0	ThreeMem27																
0	0	0	1	1	ALU8					TwoMem19											
0	1	0	0	0	ALU8					TwoMov19											
0	1	0	0	1	MAC8					TwoMem19											
0	1	0	1	0	MAC8					TwoMov19											
0	1	0	1	1	MAC8				ALU8					OneMem11							
0	1	1	0	0	MAC16									OneMem11							
0	1	1	0	1	ALU16									OneMem11							
0	1	1	1	0	DMOV16									OneMem11							
0	1	1	1	1	DMOV24													0	0	0	
0	1	1	1	1	ALU16								MAC8					1	0	0	
0	1	1	1	1	MAC16								ALU8					0	1	0	
0	1	1	1	1	Reserved													1	1	0	
0	1	1	1	1	Reserved															1	

32비트 전 명령 포맷

상기한 바와 같이, 5개의 헤더 비트는 명령 프래그먼트의 특정한 조합 뿐만 아니라 전 명령 길이를 나타낸다.

예를 들어, 00010 헤더는 나머지 27 명령 비트가 ThreeMem27 명령 프래그먼트를 포함하는 것을 나타내며, 00011 헤더는 나머지 27 명령 비트가 TwoMem19 명령 프래그먼트가 뒤따르는 ALU8 명령 프래그먼트를 포함하는 것을 나타낸다.

01111 헤더용으로, 최하위 테일 비트는 명령 프래그먼트의 조합을 또한 나타낸다. 예를 들어, 0 의 최하

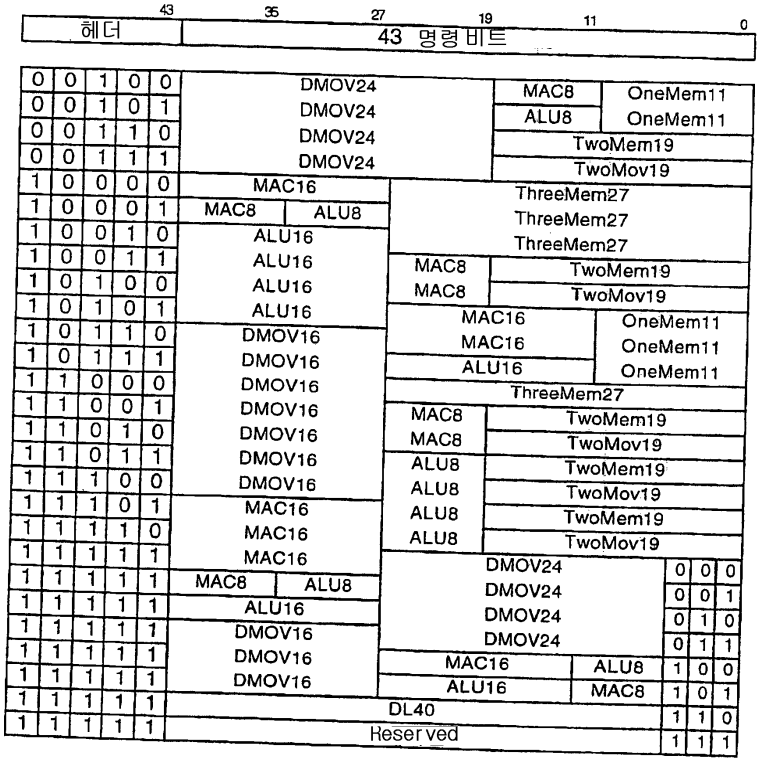
위 테일 비트용으로, 그 다음 2개의 최하위 비트는 나머지 24비트가 DMOV24, MAC8 이 뒤따르는 ALU16, 또는 ALU8 명령 프래그먼트가 뒤따르는 MAC16 를 포함하는지 여부를 나타낸다. 다른 테일 비트 상태, 1의 최하위 테일 비트는 예약된 조합을 특정화한다.

32개의 비트 명령으로 인하여 대부분의 수행 동작이 동시에 수행될 수 있고, 이것은 명령 크기를 줄이는 한편 파이프라인을 용이하게 한다. 예를 들어, 필터링과 같은 응용을 위하여 2개의 인출 동작 및 멀티플라이/어큐뮬레이트 동작을 수행하는 것은 흔한 일이다. 32비트 명령으로 인하여 상기와 같은 동작이 필 (fill) 48 비트 명령 공간을 필요로 하지 않고 파이프라인 형식으로 수행될 수 있다.

또한, 32비트 명령으로 인하여 가장 큰 명령 크기를 사용하지 않고도 MAC 및 ALU 동작이 프로그램 점프 및 콜 (call) 동작뿐만 아니라 동시에 수행될 수 있다.

표 8 은 본 발명의 실시예에 따라 수행될 때 명령 프래그먼트 조합 및 48 비트 전 명령용 포맷을 나타낸다.

표 8



48비트 전 명령 포맷

5개의 헤더 비트는 특정한 명령 프래그먼트 조합뿐만 아니라 명령 길이를 특정화한다. 예를 들어, 00100 헤더 비트는 43개의 나머지 명령 비트가 DMOV24, MAC8, OneMem11 명령 프래그먼트로 구성된 것을 나타낸다. 10011 헤더 비트는 43개의 나머지 비트가 ALU16, MAC8, 및 TwoMem19 명령 프래그먼트로 구성된 것을 나타낸다.

11111 헤더 비트용으로, 3개의 최하위 테일 비트는 나머지 명령 비트에 포함된 명령 프래그먼트를 또한 나타낸다. 000 테일 비트는 나머지 40 명령 비트가 MAC16 및 DMOV24 명령 프래그먼트를 포함하는 것을 나타낸다. 001 테일 비트는 나머지 40 개의 명령 비트가 MAC8, ALU8, 및 DMOV24 명령 프래그먼트를 포함하는 것을 나타낸다. 110 테일 비트는 나머지 40개 명령 비트가 DL40 명령 프래그먼트를 포함하는 것을 나타낸다.

48비트 전 명령내에 제공되는 명령 프래그먼트 조합으로 인하여 많은 동작이 동시에 수행될 수 있고 따라서 직렬로 수행되는 것보다 빠르게 수행될 수 있다. 예를 들어, 여러 개의 48비트 전 명령으로 인하여 ALU 동작, MAC 동작 및 메모리 동작이 동시에 수행될 수 있다. 메모리 동작은 로드, 저장, 및 데이터 이동 동작을 포함하며, 흔히 다중 메모리 위치가 한번에 액세스될 수 있게 한다.

48비트 명령으로 인하여 승산 동작이 ALU 동작, 데이터 인출, 및 프로그램 흐름 동작과 함께 파이프라인 형식으로 수행될 수 있다. 이것은, (시프팅과 같은) ALU 동작이 뒤따르는 MAC 동작을 수행함으로써 흔히 수행되

는 스케일링 동작과 조합될 때 필터링에 유용할 수 있다. MAC 및 ALU 를 사용하는 다른 응용에는 3개 이상의 데이터 스트림을 조합하는 것이 포함된다. 특히 3개의 버스 아키텍처를 사용시 48비트 명령은 이러한 경우에 파이프라인 동작을 용이하게 한다.

이것은 단일 48비트 전 명령 내지 5 개 (MAC, ALU, FETCH1, FETCH2, 및 STORE) 로 수행될 수 있는 동작 수를 효율적으로 증가시킨다. DSP 에서 다중 명령을 동시에 수행할 수 있는 기능은, 일반적으로, DSP 내부의 다양한 처리 시스템을 접속하기 위한 다중 내부 버스와 함께 DSP 를 사용함으로써 더 향상된다. 상이한 세트의 데이터가 상이한 버스를 사용하여 동시에 이동 및 액세스될 수도 있다.

수행될 수 있는 동작 수에 의거하여 명령 길이를 변경함으로써 명령 메모리가 사용되는 효율을 더 증가시킨다. 어떤 특정한 태스크는 다중 동작이 동시에 수행될 수 있는 주기, 및 보다 적은, 또는 한개의, 동작이 수행될 수 있는 다른 주기를 갖는다. 동시에 수행될 수 있는 동작 수에 따라 명령 길이를 조절함으로써, 명령 메모리 양이 감소된다.

상기한 방법이 타이트한 (tight) 명령 패킹과 결합되어 함께 사용될 때, 필요한 명령 메모리가 더 감소된다. 가변 길이 명령 또는 타이트하게 패킹된 명령 또는 두 명령을 함께 사용함으로써 다중 버스 아키텍처 및 다중 액세스 레지스터 뱅크 사용이 용이해지며, 보다 많은 회로 영역이 이용가능해진다. 따라서, 이러한 본 발명의 태양은 향상된 성능 및 향상된 효율을 동시에 제공하기 위해 결합된다.

C. 명령 프래그먼트

상기한 바와 같이, 전 명령은 한 개 또는 소정의 방식으로 그룹화된 그 이상의 명령 프래그먼트의 세트로 구성된다. 본 발명의 실시예에서 이용가능한 명령 프래그먼트 세트가 표 5 에 도시된다. 본 발명의 실시예에서 제공된 전 명령을 사용하여 이용가능한 조합 및 명령 프래그먼트는 동작 세트가 결합되어 함께 수행될 수 있도록 설계되어 주어진 동작을 수행하는 데 필요한 명령 메모리의 양이 감소된다. 본 발명의 실시예에서 사용되는 다양한 명령 프래그먼트의 포맷 및 동작이 아래에 뒤따른다.

C.1 명령 프래그먼트 명명법

다음의 명령 프래그먼트 및 서브프래그먼트에서, 다음의 약어는 표 9 및 10 에 도시된 레지스터를 언급하는 데 사용된다. 또한, 본 발명의 실시예에서 사용되는 특정한 비트 코드 (매핑) 는 좌측에 도시된다.

표 9

	Dreg
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	L0h
1001	L1h
1010	L2h
1011	L3h
1100	L0l
1101	L1l
1110	L2l
1111	L3l

	R0-R7	Lh/Ll	A0-A7	Lreg/Dreg
000	R0	L0h	A0	L0
001	R1	L1h	A1	L1
010	R2	L2h	A2	L2
011	R3	L3h	A3	L3
100	R4	L0l	A4	D0
101	R5	L1l	A5	D1
110	R6	L2l	A6	D2
111	R7	L3l	A7	D3

	AS	AL
0	AS0	AL0
1	AS1	AL1

	R0-R3	L0-L3	D0-D3	C0-C3	Cmod
0 0	R0	L0	D0	C0	++
0 1	R1	L1	D1	C1	--
1 0	R2	L2	D2	C2	++CM0
1 1	R3	L3	D3	C3	++CM1

cond									
00000	LT		01000	L0LT		10000	L1LT		11000	L2LT
00001	LE		01001	L0LE		10001	L1LE		11001	L2LE
00010	EQ		01010	L0EQ		10010	L1EQ		11010	L2EQ
00011	NE		01011	L0NE		10011	L1NE		11011	L2NE
00100	GE		01100	L0GE		10100	L1GE		11100	L2GE
00101	GT		01101	L0GT		10101	L1GT		11101	L2GT
00110	OV		01110	L0OV		10110	L1OV		11110	L2OV
00111	Uncond		01111	Rsvd		10111	Rsvd		11111	Rsvd

명령 프래그먼트 명명법 및 코드

주 : L3 는 조건문을 갖지 않는다.

표 10

	RegA	regB
00000	R0	R0
00001	R1	R1
00010	R2	R2
00011	R3	R3
00100	R4	R4
00101	R5	R5
00110	R6	R6
00111	R7	R7
01000	L0h	L0h
01001	L1h	L1h
01010	L2h	L2h
01011	L3h	L3h
01100	L0l	L0l
01101	L1l	L1l
01110	L2l	L2l
01111	L3l	L3l
10000	A0	B0
10001	A1	B1
10010	A2	B2
10011	A3	B3
10100	A4	B4
10101	A5	B5
10110	A6	B6
10111	A7	B7
11000	AS0	BS0
11001	AS1	BS1
11010	AL0	BL0
11011	AL1	BL1
11100	AM0	BM0
11101	AM1	BM1
11110	Reserved	Reserved
11111	Reserved	Reserved

	regC
0000	L0
0001	L1
0010	L2
0011	L3
0100	D0
0101	D1
0110	D2
0111	D3
1000	C0
1001	C1
1010	C2
1011	C3
1100	CM0
1101	CM1
1110	Reserved
1111	Reserved

명령 프래그먼트 명명법 및 코드

regA 는 A 메모리에 저장될 수 있는/A 메모리로부터 로드될 수 있는 모든 레지스터를 포함한다.

regB 는 B 메모리에 저장될 수 있는/B 메모리로부터 로드될 수 있는 모든 레지스터를 포함한다.

regC 는 C 메모리에 저장될 수 있는/C 메모리로부터 로드될 수 있는 모든 레지스터를 포함한다.

C.2 명령 프래그먼트 설명

일련의 명령 프래그먼트들은 MAC8과 MAC16의 2가지 유형의 MAC 명령 프래그먼트들을 포함한다. MAC8명령 프래그먼트들은 부호화된-부호화되지 않은 것과 부호화-부호화된 곱셈형을 지원하고 그 결과는 어큐뮬레이터 (L0 또는 L1)에 저장된다. MAC8 명령 프래그먼트는 16 비트 전(full) 명령을 이용하는 MAC 동작을 허용하고, MAC 동작들을 요하는 많은 병렬 명령 조합들이 48비트 명령 대신에 32비트 명령으로 인코드되도록 함으로써 명령 RAM을 절약한다. 일반적으로, MAC8 명령들에 의해 수행되는 절차는 다음 수학식을 따른다.

수학식 2

$$\begin{Bmatrix} L0 \\ L1 \end{Bmatrix} = \begin{bmatrix} L0\pm \\ L1\pm \end{bmatrix} \begin{Bmatrix} R0 \\ R2 \\ R4 \\ R6 \end{Bmatrix} * \begin{Bmatrix} R0 \\ R1 \\ R3 \\ R5 \end{Bmatrix} \begin{bmatrix} (SU) \\ (SS) \end{bmatrix}$$

수학식 2 에서 나타난 바와 같이, MAC8 명령 프래그먼트는 레지스터 (L0 는 L1) 의 내용이 레지스터 (R0, R2, R4 및 R6) 와 (R0, R1, R3 및 R5) 의 곱으로 합계가 되도록 하거나 레지스터들의 곱으로 직접 설정되도록 한다.

게다가, 부호화되거나 부호화되지 않은 곱셈들이 명기될 수 있다. MAC 연산이 MAC8 명령을 이용하여 수행될 수 있는 레지스터들의 수를 제한함으로써, 명령의 길이는 8비트로 유지될 수 있어 더 짧은 8비트 명령 프래그먼트를 이용하여 MAC 연산이 수행되도록 한다.

MAC8 명령에 의해 수행된 특정 동작이 표 11 에 설명된 명령을 구성하는 8비트의 값에 의해 구체화된다.

표 11

7	6	5	4	3	2	1	0
MAC Operation			mac8Op1		mac8Op2		SU/SS

MAC8 명령 프래그먼트 포맷

SU/SS 는 부호화된 또는 부호화되지 않은 곱셈을 명기한다. MAC8 명령 프래그먼트 내의 다양한 동작들을 명기하는 코드들이 표 12 에 나열되어 있다.

표 12

MAC Operation	
000	L0=
001	L1=
010	L0=L0+
011	L1=L1+
100	L0=L0-
101	L1=L1-
110	L0=L1+
111	L0=L1-

mac8Op1	
0 0	R0
0 1	R2
1 0	R4
1 1	R6

mac8Op2	
0 0	R0
0 1	R1
1 0	R3
1 1	R5

SU/SS	
0	SU
1	SS

MAC8 명령 프래그먼트 코드

그러므로, 0×99의 MAC8 명령은 레지스터 R0 와 R3 의 부호화되지 않은 곱과 함께 레지스터 (L0) 의 내용의 합을 레지스터 L0 로 이동시킨다.

MAC16 명령 프래그먼트는 추가의 레지스터들이 멀티플라이-어큐뮬레이트 동작에서 이용되도록 함으로써 부가적인 유연성을 제공한다. 수학식 3 은 MAC16 명령 프래그먼트를 이용하여 수행될 수 있는 동작들을 설명한다.

수학식 3

$$\begin{Bmatrix} L0 \\ L1 \\ L2 \\ L3 \end{Bmatrix} = \begin{bmatrix} L0 & [>>16] & \pm \\ L1 & [>>16] & \pm \\ L2 & [>>16] & \pm \\ L3 & [>>16] & \pm \end{bmatrix} \; macOp1 * macOp2 \;\; [(mtype)] \;\; [<< mshift] \;\; [:CPS]$$

예컨대, 모든 어큐뮬레이터 (L0-L3) 는 비록 어큐뮬레이터들의 모든 조합들이 멀티플라이-어큐뮬레이터 명령들에서 허용되지 않을지라도 수신지로 이용될 수 있다. CPS 필드는 코프로세서가 특정 동작을 병렬로 수행해야 함을 신호한다. MAC16 명령에 의해 수행된 특정 동작은 표 13 에 나타낸 명령을 구성하는 16비트의 값들에 의해 특정된다.

표 13

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAC Operation				macOp1			macOp2				mtype		mshift		CPS

MAC16 명령 프래그먼트 포맷

MAC16 명령 프래그먼트 내의 다양한 동작들을 명기하는 코드들이 표 14 에 나열되어 있다.

표 14

	MAC Operation	macOp2
0000	L0=	R0
0001	L1=	R1
0010	L0=L0 [>>16] +	R2
0011	L1=L1 [>>16] +	R3
0100	L0=L0 [>>16] -	R4
0101	L1=L1 [>>16] -	R5
0110	L0=L1 [>>16] +	R6
0111	L0=L1 [>>16] -	R7
1000	L2=	L0h
1001	L3=	L1h
1010	L2=L2 [>>16] +	L2h
1011	L3=L3 [>>16] +	L3h
1100	L2=L2 [>>16] -	L0l
1101	L3=L3 [>>16] -	L1l
1110	L2=L3 [>>16] +	L2l
1111	L2=L3 [>>16] -	L3l

macOp1	
000	R0
001	R2
010	R4
011	R6
100	L0h
101	L1h
110	L2h
111	L3h

mtype	
00	(SU)
01	(UU)
10	(SS)
11	주 참조

mshift	
00	<< 0
01	<< 1
10	<< 2
11	<< 3

CPS	
0	OFF
1	ON

MAC16 명령 프래그먼트 코드

주 : 명령 L0=R0*R0(SU)<<0 은 NOP로 디코드된다.

mtype 11 은 부호화-부호화 멀티플라이/어큐뮬레이트 명령에 대해 스트레이트(straight) 곱셈과 16만큼 오른쪽으로 어큐뮬레이터를 시프트하는 것에 대한 RND로 이용된다.

mtype SU 및 macOp1=macOp2 를 갖는 MAC 명령은 허용되지 않는다.

MAC16 명령 프래그먼트는 3까지의 좌측 시프트를 허용하여 스트레이트 곱셈(누산 없음) 동안에 라운드 동작을 수행할 수 있는데, 이 라운드는 시프트 후에 일어난다. 누산이 수행될 때, 가산될 어큐뮬레이터는 부호화-부호화된 곱셈과 병렬로 16만큼 시프트 다운될 수 있다. CPS 비트는 MAC 동작에서 이용된 데이터가 코프로세서에 전송되어야함을 가리키기 위한 코프로세서 스트로브 비트이다.

MAC8 명령 프래그먼트는 MAC16에 의해 수행될 수 있는 동작들의 서브세트를 수행한다는 점을 주목한다. MAC8 명령 프래그먼트에 대해 선택된 특정 집합의 명령들은 MAC16 명령 프래그먼트를 이용하여 수행될 수 있는 상기 집합의 동작들로부터 가장 흔히 수행된다. 이는 MAC8 명령 프래그먼트를 이용하여 MAC 연산들의 대부분이 수행될 수 있도록 허용함으로써 프로그램 메모리를 절약한다.

8비트 ALU8 명령 프래그먼트는 MAC 연산 (MAC8과 MAC16)과 병행하여 가장 흔하며 즉시 데이터를 포함하지 않는 ALU 동작들로 구성된다. 모든 ALU8 시프트 동작들은 명령 인코딩 비트들을 절약하기 위해 내부 시프트 레지스터 (SR) 를 이용하는 산술 시프트이다. ALU8 명령 프래그먼트를 이용하여 수행된 동작들은 표 15 에 나타나 있다.

표 15

NOP;	NOP (병렬 조합을 위해 필요함)
LD = DETNORM(LS);	블록 정규화 인자를 결정함
LD = SET(LS);	Accumulator를 복사(포화는 아님)
LD = LS << SR;	Accumulator시프트
LD = RND(LS << SR);	Accumulator를 시프트하고 라운드함
LD = LD ± (LS << SR);	시프트된 Accumulator를 누산함
LD = LS ± LT;	Accumulator를 가산 또는 감산
LS ± LT;	Accumulator의 가산/감산 결과를 0으로(플래그를 세트)

ALU8 명령 프래그먼트 동작

LS 는 로드 소스(L0-L3)이고 LD는 로드 수신지(L0-L3)이다.

ALU8 명령 프래그먼트에 의해 수행된 특정 동작들이 표 16 에 설명되는 명령 프래그먼트로 이루어지는 8비트의 값들로 명기되어 있다.

표 16

7	6	5	4	3	2	1	0
0	ALUOp			LS		LD	
0	1	1	Sign	LS		LT	
1	LD		Sign	LS		LT	

ALU8 명령 프래그먼트 동작

ALU8 명령 프래그먼트를 이용하여 수행된 동작들을 명기하기 위해 이용된 특정 코드들이 표 17 에 설명되어 있다.

표 17

ALUOp	
000	LD=DETNORM(LS)
001	LD=SET(LS)
010	LD=LS<<SR
011	LD=RND(LS<<SR)
100	LD=LD + (LS<<SR)
101	LD=LD - (LS<<SR)
110	LS+LT
111	LS-LT

LD / LS / LT	
00	L0
01	L1
10	L2
11	L3

Sign	
0	[LD=] LS+LT
1	[LD=] LS-LT

ALU8 명령 프래그먼트 동작

주: 올(all)-제로 명령 L0=DETNORM(L0) 은 NOP로 디코드된다.

LD=DETNORM(LD)인 명령은 허용되지 않는다.

ALU8 클리어(clear) 어큐뮬레이터 명령은 LD=LD-LD이다.

ALU16 명령 프래그먼트는 산술과 논리 시프트를 둘다 허용한다. ALU16 명령 프래그먼트에 의해 수행된 특정 동작들이 표 18 에 설명되어 있다.

<< 표시는 산술 시프트를 나타내고, <<< 는 논리 시프트를 나타낸다.

표 18

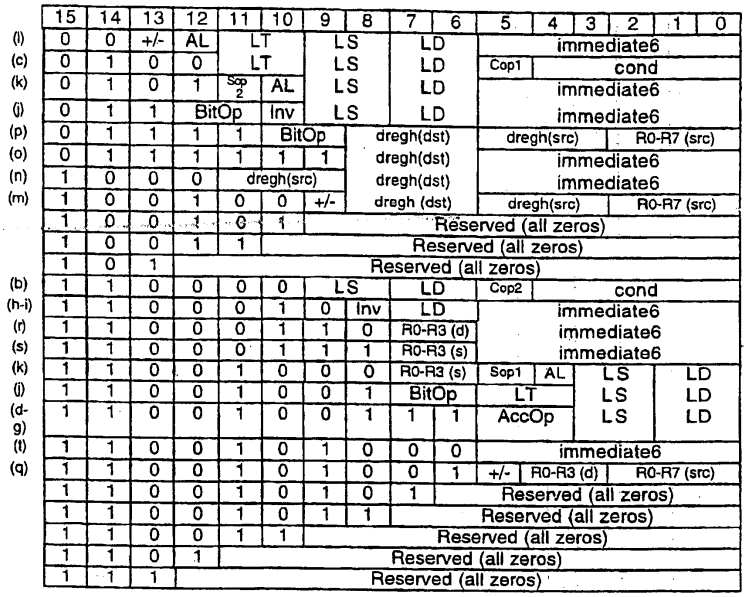
(a) *IF cond NOP*;	조건 NOP(병렬 조합용)
(b) *IF cond LD = SET(LS)*;	조건부로 Accumulator를 복사
(c) *IF cond LD = LS ± LT*;	조건부로 Accumulator를 가산/감산
(d) *LD = NORM(LS,SR)*;	Accumulator를 정규화
(e) *LD = ABS(LS)*;	Accumulator의 절대값
(f) *LD = -LS*;	Accumulator를 부정으로 함
(g) *LD=~LS*;	Accumulator를 인버트시킴(1의 보수)
(h) *LD = BIT(immediate5)*;	비트 마스크 신설 (LD = 0x1<<imm5)
(i) *LD=~BIT(immediate5)*;	인버트된 비트 마스크를 신설 (LD=~(0x1<<imm5))
(j) $LD = LS\left\{\begin{matrix}\& \\ \mid \\ \wedge\end{matrix}\right\}\left\{\begin{matrix}LT \\ BIT(immediate5) \\ \sim BIT(immediate5)\end{matrix}\right\}$;	비트와이즈 AND, OR, XOR
(k) $LD = [RND]\left(LS\left\{\begin{matrix}<< \\ <<<\end{matrix}\right\}\left\{\begin{matrix}R0-R3 \\ immediate6\end{matrix}\right\}\right)$;	Accumulator를 시프트(및 라운드)
(l) $LD = LS \pm (LT\left\{\begin{matrix}<< \\ <<<\end{matrix}\right\} immediate6)$;	Accumulator를 시프트하고 가산함
(m) $\left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} = \left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} \pm \{R0-R7\}$;	레지스터 가산
(n) $\left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} = \left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} + immediate6$	Immediate 가산
(o) $\left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} = SET(immediate6)$;	Immediate 로드
(p) $\left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\} = \left\{\begin{matrix}L0h-L3h \\ R0-R3\end{matrix}\right\}\left\{\begin{matrix}\& \\ \mid \\ \wedge\end{matrix}\right\}\{R0-R7\}$	16비트 논리
(q) $\{R0-R3\} = SR \pm \{R0-R7\}$;	레지스터를 SR에 가산
(r) $\{R0-R3\} = SR + immediate6$;	SR에 Immediate 가산
(s) $SR = \{R0-R3\} + immediate6$;	Immediate 합으로 SR을 로드
(t) $SR = SET(immediate6)$;	SR을 Immediate 로드

ALU16 명령 프래그먼트 동작

명령 L0=SET(L0) 은 NOP로 디코드된다.

ALU16 명령 프래그먼트의 포맷이 표 19 에 개시된다.

표 19



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(l)	0	0	+/-	AL	LT		LS		LD		immediate6					
(c)	0	1	0	0	LT		LS		LD		Cop1	cond				
(k)	0	1	0	1	Sop2	AL	LS		LD		immediate6					
(j)	0	1	1	BitOp		Inv	LS		LD		immediate6					
(p)	0	1	1	1	1	BitOp		dregh(dst)			dregh(src)			R0-R7 (src)		
(o)	0	1	1	1	1	1	1	dregh(dst)			immediate6					
(n)	1	0	0	0	dregh(src)			dregh(dst)			immediate6					
(m)	1	0	0	1	0	0	+/-	dregh (dst)			dregh(src)			R0-R7 (src)		
	1	0	0	1	0	1	Reserved (all zeros)									
	1	0	0	1	1	Reserved (all zeros)										
	1	0	1	Reserved (all zeros)												
(b)	1	1	0	0	0	0	LS		LD		Cop2	cond				
(h-i)	1	1	0	0	0	1	0	Inv	LD		immediate6					
(f)	1	1	0	0	0	1	1	0	R0-R3 (d)		immediate6					
(s)	1	1	0	0	0	1	1	1	R0-R3 (s)		immediate6					
(k)	1	1	0	0	1	0	0	0	R0-R3 (s)		Sop1	AL	LS		LD	
(j)	1	1	0	0	1	0	0	1	BitOp		LT		LS		LD	
(d-g)	1	1	0	0	1	0	0	1	1	1	AccOp		LS		LD	
(t)	1	1	0	0	1	0	1	0	0	0	immediate6					
(q)	1	1	0	0	1	0	1	0	0	1	+/-	R0-R3 (d)		R0-R7 (src)		
	1	1	0	0	1	0	1	0	1		Reserved (all zeros)					
	1	1	0	0	1	0	1	1	Reserved (all zeros)							
	1	1	0	0	1	1	Reserved (all zeros)									
	1	1	0	1	Reserved (all zeros)											
	1	1	1	Reserved (all zeros)												

16 비트 ALU 명령 프래그먼트 포맷 및 코드

ALU16 명령 프래그먼트에 의해 수행되는 특별한 동작들이, 표 20 에 개시된 것과 같이 명령 프래그먼트를 구성하는 비트들의 값들로서 구체화된다.

표 20

Dregh	
000	R0
001	R1
010	R2
011	R3
100	L0h
101	L1h
110	L2h
111	L3h

BitOp	
0 0	AND
0 1	OR
1 0	XOR

Sop1	
0	LD=LS<<R0-R3
1	LD=RND(LS<<R0-R3)

Sop2	
0	LD=LS<<imm6
1	LD=RND(LS<<imm6)

AL	
0	Arithmetic Shift
1	Logical Shift

AccOp	
0 0	LD=NORM(LS,SR)
0 1	LD=ABS(LS)
1 0	LD=-LS
1 1	LD=~LS

Cop1	
0	LD=LS+LT
1	LD=LS-LT

Cop2	
0	LD=SET(LS)
1	Reserved

+/-	
0	+
1	-

Inv	
0	Normal bitmask
1	Inverse bitmask

주 : 올 제로 (all zeros) 명령 L0=SET(L0) 이 NOP로서 디코드된다.

BIT 명령 (h,i,j) 에 있어, 어셈블러는 제로 신호를 immediate5 에 부가함으로써 immediate6 을 인코드한다(이는 디코딩을 단순화시킨다).

표 20. ALU16 명령 프래그먼트 코드

DMOV16 명령 프래그먼트는 표 21에서 개시된 바와 같은 데이터 입력포트 및 데이터 출력포트 동작들의 상이한 데이터 무브 (move)를 수행하기 위한 16 비트 명령 프래그먼트이다.

표 21

(a)	*NOP*;	병렬 명령들을 위한 NOP
(b)	*LC = immediate9*;	루프 카운터 Immediate의 로드
(c)	$\left\{\begin{matrix} AM0-AM1 \\ BM0-BM1 \\ CM0-CM1 \end{matrix}\right\}$ = *immediate10*;	수정 레지스터 Immediate의 로드
(d)	$\left\{\begin{matrix} AL0-AL1 \\ BL0-BL1 \end{matrix}\right\}$ = *immediate11*;	순환길이 레지스터 Immediated의 로드
(e)	$\left\{\begin{matrix} L0-L3 \\ D0-D3 \\ L0h-L3h \end{matrix}\right\}$ = *INPORT(port_addr)*;	입력부 동작
(f)	*OUTPORT(port_addr)* = $\left\{\begin{matrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{matrix}\right\}$	출력부 동작
(g)	*OUTPORTA(port_addr)*;	버스 A상에서의 출력부 값
(h)	*OUTPORTB(port_addr)*;	버스 B상에서의 출력부 값

DMOV16 명령 프래그먼트를 유용하게 이용하는 동작들을 수행하는 데 사용되는 포맷 및 코드들이 표 22 에 개시된다.

표 22

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	Reserved (all zeros)												
(d)	0	0	1	0	AL	immediate11 (im1)										
(d)	0	0	1	1	BL	immediate11 (im2)										
(c)	0	1	0	0	0	AM	immediate10 (im1)									
(c)	0	1	0	0	1	BM	immediate10 (im2)									
(c)	0	1	0	1	0	CM	immediate10 (im3)									
	0	1	0	1	1	Reserved (all zeros)										
(b)	0	1	1	0	0	0	0	immediate9 (im1)								
	0	1	1	0	0	0	1	Reserved (all zeros)								
	0	1	1001-1111					Reserved (all zeros)								
(e)	1	0	0	0	0	0	L0-L3	Inport address (PI3)								
(e)	1	0	0	0	0	1	D0-D3	Inport address (PI3)								
(e)	1	0	0	0	1	0	L0h-L3h	Inport address (PI3)								
	1	0	0011-1111					Reserved (all zeros)								
(f)	1	1	0	0	dreg			Outport address (Abus, PO1)								
(f)	1	1	0	1	dreg			Outport address (Bbus, PO2)								
(g)	1	1	1	0	0	0	0	0	Outport address (Reads A bus)							
(h)	1	1	1	0	0	0	0	1	Outport address (Reads B bus)							
	1	1	10001-11111					Reserved (all zeros)								

주 : 명령 LC=0 은 NOP로서 디코드된다.

immediate10 이 표시되고 immediate11 이 표시되지 않는다.

표 22. DMOV16 명령 프래그먼트 포맷

명령 OUTPORTA(port_addr) 는 Abus 상의 값을 판독하고, 그 값을 지정된 포트로 출력시킨다. 동시에 메모리 A 로부터 값을 판독함으로써, 명령은 값을 직접적으로 메모리 A에서 포트로 보내는데 사용될 수 있다. OTUPORTB(port_addr) 도 유사하게 동작한다.

DMOV24 명령 프래그먼트는 표 23에 개시된 것과 같은 상이한 로드/저장 레지스터 다이렉트 (load/store register direct) 또는 로드 레지스터 이미디어트 (load register immediate) 동작들을 수행하기 위한 24 비트 명령 프래그먼트이다.

표 23

(a)	$\{regA\} = memA(address14);$	
(b)	$memA(address14) = \{regA\};$	L/S direct memory A
(c)	$\{regB\} = memB(address14);$	
(d)	$memB(address14) = \{regB\};$	L/S direct memory B
(e)	$\{regC\} = memC(address14);$	
(f)	$memC(address14) = \{regC\};$	L/S direct memory C
(g)	$\left\{\begin{matrix} A0-A7 \\ B0-B7 \\ C0-C3 \end{matrix}\right\} = address14;$	Address 레지스터 Immediate의 로드
(h)	$\left\{\begin{matrix} AS0-AS1 \\ BS0-BS1 \end{matrix}\right\} = address14;$	순환시작 레지스터 Immediate의 로드
(i)	$\left\{\begin{matrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{matrix}\right\} = immediate16;$	데이터 레지스터 Immediate의 로드
(j)	*LOOP UNTIL address17;*	Loop until end address.
(k)	*CALL address17;*	Function Call.
(l)	$[IF\ cond]\left\{\begin{matrix} JUMP \\ JUMPD \end{matrix}\right\} address17;$	[Conditional] [Delayed] Jump.

시작 레지스터들은 AGU 장치에 위치된다.

표 23. DMOV24 명령 프래그먼트 동작

표 24 는 본 발명의 바람직한 실시예에 따르는 DMOV24 명령을 유용하게 이용하는 다양한 동작을 수행하는데 사용되는 포맷 및 몇몇 코드들을 제공한다.

표 24

	23	22	21	20	19	18	17	16	15	14	13-0
(l)	0	0	cond					address17 (JUMP)			
(l)	0	1	cond					address17 (JUMPD)			
(j)	1	0	0	0	0	0	0	address17 (LOOP)			
(k)	1	0	0	0	0	0	1	address17 (CALL)			
(g)	1	0	0	0	0	1	0	A0-A7			address14 (im1)
(g)	1	0	0	0	0	1	1	B0-B7			address14 (im2)
(e)	1	0	0	0	1	0	regC (dst)				address14 (im3, Cbus+PI3)
(f)	1	0	0	0	1	1	regC (src)				address14 (im3, Cbus+PO3)
(a)	1	0	0	1	0	regA (dst)					address14 (im1, Abus+PI1)
(b)	1	0	0	1	1	regA (src)					address14 (im1, Abus+PO1)
(c)	1	0	1	0	0	regB (dst)					address14 (im2, Bbus+PI2)
(d)	1	0	1	0	1	regB (src)					address14 (im2, Bbus+PO2)
(i)	1	0	1	1	dreg			immediate16 (im1+PI1)			
(i)	1	1	0	0	dreg			immediate16 (im2+PI2)			
(g)	1	1	0	1	0	0	0	0	C0-C3		address14 (im3)
(h)	1	1	0	1	0	0	0	1	0	AS	address14 (im1)
(h)	1	1	0	1	0	0	0	1	1	BS	address14 (im2)
	1	1	0	1001-1111				Reserved (all zeros)			
	1	1	1	Reserved (all zeros)							

주 : Address14 및 Address17 은 부화화되지 않고 (unsigned), immediate16 은 부호화된다 (signed).

표 24. DMOV24 명령 프래그먼트 포맷 및 코드

다른 명령 프래그먼트들 뿐만 아니라 DMOV24에 있어서, 어떤 동작들은 두 번 인코드된다는 것에 주목하여야 한다. 예를 들어, 행 (i) 및 (j)에서 지정된 형태들은 동일한 동작을 인코드하고, 하나는 immediate bus (Im1) 의 사용을 특정하고 다른 하나는 immediate bus (Im2) 의 사용을 특정한다. 두 번의 인코딩은 명령 프래그먼트가 매우 다양한 다른 명령 프래그먼트들로 결합되게 하고, 이는 Immediate bus 1 은 물론 Immediate bus 2 의 사용을 요할 수 있다.

40-bit 듀얼 로드 명령 프래그먼트 (DL40) 은 immediate load 또는 address load 동작들을 수행하기 위한 40 비트 명령 프래그먼트이다. 본 발명의 바람직한 실시예에서 수행되는 특별한 동작들이 표 25 에 도시된다.

표 25

(a)	$\{A0-A3\} = address14, \{B0-B3\} = address14;$
(b)	$\{A0-A3\} = address14, \{C0-C3\} = address14;$
(c)	$\{B0-B3\} = address14, \{C0-C3\} = address14;$
(d)	$\left\{\begin{matrix}A0-A3\\B0-B3\\C0-C3\end{matrix}\right\} = address14, \left\{\begin{matrix}R0-R7\\L0h-L3h\\L0l-L3l\end{matrix}\right\} = immediate16;$
(e)	$\left\{\begin{matrix}L0-L3\\D0-D3\end{matrix}\right\} = immediate32;$
(f)	$\{A0-A3\} = address14, regB = memB(address14);$
(g)	$\{B0-B3\} = address14, regA = memA(address14);$
(h)	$\{A0-A3\} = address14, regC = memC(address14);$
(i)	$\{C0-C3\} = address14, regA = memA(address14);$
(j)	$\{B0-B3\} = address14, regC = memC(address14);$
(k)	$\{C0-C3\} = address14, regB = memB(address14);$
(l)	$regA = memA(address14), regB = memB(address14);$
(m)	$regA = memA(address14), regC = memC(address14);$
(n)	$regB = memB(address14), regC = memC(address14);$

표 25. DL40 명령 프래그먼트 포맷

각 동작을 위한 DL40 명령 프래그먼트의 형태는 표 26에서 제공된다.

표 26

	39	38	37	36	35	34	33	32	31	30	29-16	15-14	13-0
	0	0	0	0	0	Reserved							
(e)	0	0	0	0	1	Lreg/Dreg			immediate32				
(d)	0	0	0	1	A0-A3		dreg				address14	immediate16	
(d)	0	0	1	0	B0-B3		dreg				address14	immediate16	
(d)	0	0	1	1	C0-C3		dreg				address14	immediate16	
(b)	0	1	0	0	0	0	0	0	C0-C3		Caddress14	A0-A3	Aaddress14
(c)	0	1	0	0	0	0	0	1	C0-C3		Caddress14	B0-B3	Baddress14
(a)	0	1	0	0	0	0	1	0	A0-A3		Aaddress14	B0-B3	Baddress14
	0	1	0	0	0	0	1	1	Reserved (all zeros)				
	0	1	0	0	0	1	Reserved (all zeros)						
(h)	0	1	0	0	1	0	regC				Caddress14	A0-A3	Aaddress14
(j)	0	1	0	0	1	1	regC				Caddress14	B0-B3	Baddress14
(g)	0	1	0	1	0	regA					Aaddress14	B0-B3	Baddress14
(i)	0	1	0	1	1	regA					Aaddress14	C0-C3	Caddress14
(f)	0	1	1	0	0	regB					Baddress14	A0-A3	Aaddress14
(k)	0	1	1	0	1	regB					Baddress14	C0-C3	Caddress14
	0	1	1	1	Reserved								
(m)	1	0	0	regC			re...				Caddress14	..gA	Aaddress14
(n)	1	0	1	regC			re...				Caddress14	..gB	Baddress14
(l)	1	1	regA				re...				Aaddress14	..gB	Baddress14

주 : Address14 는 부호화되지 않고 (unsigned), immediate16 및 immediate32 는 부호화된다 (signed).

표 26. DL40 명령 프래그먼트 포맷 및 코드

표 5 에서도 도시된 바와 같이, 메모리 무브 및 프로그램 플로우 프래그먼트들의 4가지 유형들이 본 발

명의 바람직한 실시예에서 제공되며, 그 리스트는 표 27에서 제공된다.

표 27

OneMem11
TwoMem19
TwoMov19
ThreeMem27

표 27. 메모리 무브 및 프로그램 플로우 명령 프래그먼트

각 메모리 무브 및 프로그램 플로우 명령 (MMPF) 프래그먼트는 표 28 에 열거된 MMPF 서브프래그먼트들의 집합으로 구성된다.

표 28

	명령 서브프래그먼트들	
LD(A)	메모리 A 인다이렉트의 로드	8
ST(A)	메모리 A 인다이렉트의 저장	8
LD(B)	메모리 B 인다이렉트의 로드	8
ST(B)	메모리 B 인다이렉트의 저장	8
LS(C)*	메모리 C 인다이렉트의 로드/저장	8
DMOVA*	버스 A 레지스터 이동	8
DMOVB*	버스 B 레지스터 이동	8
PF8	8-비트 프로그램 플로우	8

표 28. 조합 데이터 무브 및 프로그램 플로우 명령 서브프래그먼트

MMPF 명령 프래그먼트들의 포맷 및 동작이 우선 설명되고, MMPF 서브프래그먼트들의 포맷 및 동작의 보다 상세한 설명이 뒤따를 것이다.

OneMem11 MMPF 명령 프래그먼트가 단일 메모리 로드 및 저장 동작들, 데이터 무브 동작들 및 프로그램 플로우 동작들을 수행하는데 사용된다. 여기에 제공된 바람직한 실시예로서, 8가지의 상이한 동작들이 표 29 에 도시된 것과 같은 11 비트 프래그먼트 중에서 제 1에서 제 3 비트들에 의해 지시된 특정한 동작과 함께, OneMem11 MMPF 명령 프래그먼트를 사용하여 수행되며, 표 29 는 OneMem11 데이터 무브 명령 프래그먼트를 사용하여 수행될 수 있는 동작들을 열거한다.

표 29

10	9	8	7	6	5	4	3	2	1	0
0	0	0	LD(A)							
0	0	1	ST(A)							
0	1	0	LD(B)							
0	1	1	ST(B)							
1	0	0	LS(C)							
1	0	1	DMOVA							
1	1	0	DMOVB							
1	1	1	PF8							

표 29. OneMem11 명령 프래그먼트 포맷

TwoMem19 MMPF 명령 프래그먼트는 표 30 에 개시된 것과 같이 수행되는 메모리 로드 및 저장 동작들의 8개의 상이한 조합들을 가능케하는 19 비트 명령 프래그먼트이다.

표 30

18	17	16	15-8	7-0
0	0	0	LD(A)	LD(B)
0	0	1	LD(A)	ST(B)
0	1	0	LD(A)	LS(C)
0	1	1	ST(A)	LD(B)
1	0	0	ST(A)	ST(B)
1	0	1	ST(A)	LS(C)
1	1	0	LS(C)	LD(B)
1	1	1	LS(C)	ST(B)

TwoMem19 명령 프래그먼트 포맷

TwoMov19 MMPF 명령 프래그먼트는 표 31에 도시된 바와 같이 데이터 무브 동작들에 따른 메모리 로드 및 기억 동작들의 8 개의 상이한 결합을 허용하는 19 비트 명령 프래그먼트이다.

표 31

18	17	16	15-8	7-0
0	0	0	LD(A)	DMOVB
0	0	1	ST(A)	DMOVB
0	1	0	DMOVA	LD(B)
0	1	1	DMOVA	ST(B)
1	0	0	DMOVA	LS(C)
1	0	1	LS(C)	DMOVB
1	1	0	DMOVA	DMOVB
1	1	1	보존	

TwoMev19 명령 프래그먼트 포맷

ThreeMem27 MMPF 명령 프래그먼트는 표 32에서 도시된 바와 같이 수행되는 데이터 동작들, 메모리 기억, 메모리 로드에 대한 8 개의 상이한 결합을 허용하는 27 비트 명령 프래그먼트이다.

표 32

26	25	24	23-16	15-8	7-0
0	0	0	LS(C)	LD(A)	LD(B)
0	0	1	LS(C)	LD(A)	ST(B)
0	1	0	LS(C)	ST(A)	LD(B)
0	1	1	LS(C)	ST(A)	ST(B)
1	0	0	LS(C)	DMOVA	LD(B)
1	0	1	LS(C)	DMOVA	ST(B)
1	1	0	LS(C)	LD(A)	DMOVB
1	1	1	LS(C)	ST(A)	DMOVB

ThreeMem27 명령 프래그먼트 포맷

수학식 4 는 LD(A) 명령 서브프래그먼트에 의해 수행되는 동작들을 제공한다.

수학식 4

$$\begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix} = *AX \begin{bmatrix} ++ \\ -- \\ ++AM0 \\ ++AM1 \end{bmatrix}$$

표 33은 본 발명의 예시적인 실시예에 따른 LD(A) 명령 서브프래그먼트의 포맷을 제공한다.

표 33

7	6	5	4	3	2	1	0
dreg				A0-A3		Amod	

LD(A) 명령 서브프래그먼트 포맷

수학식 5는 LD(B) 명령 서브프래그먼트에 의해 수행되는 동작들을 제공한다.

수학식 5

$$\begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix} = *BX\begin{bmatrix} ++ \\ -- \\ ++BM0 \\ ++BM1 \end{bmatrix}$$

표 34는 본 발명의 예시적인 실시예에 따른 LD(B) 명령 서브프래그먼트의 포맷을 제공한다.

표 34

7	6	5	4	3	2	1	0
dreg				B0-B3		Bmod	

LD(B) 명령 서브프래그먼트 포맷

수학식 6 은 ST(A) 명령 서브프래그먼트에 의해 수행되는 동작들을 제공한다.

수학식 6

$$*AX\begin{bmatrix} ++ \\ -- \\ ++AM0 \\ ++AM1 \end{bmatrix} = \begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix}$$

표 35는 본 발명의 예시적인 실시예에 따른 ST(A) 명령 서브프래그먼트의 포맷을 제공한다.

표 35

7	6	5	4	3	2	1	0
dreg				A0-A3		Amod	

ST(A) 명령 서브프래그먼트 포맷

수학식 7은 ST(B) 명령 서브프래그먼트에 의해 수행되는 동작들을 제공한다.

수학식 7

$$*BX\begin{bmatrix} ++ \\ -- \\ ++BM0 \\ ++BM1 \end{bmatrix} = \begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix}$$

표 36 은 본 발명의 예시적인 실시예에 따른 ST(B) 명령 서브프래그먼트의 포맷을 제공한다.

표 36

7	6	5	4	3	2	1	0
dreg				B0-B3		Bmod	

ST(A) 명령 서브프래그먼트 포맷

표 37 은 DMOVA 명령 서브프래그먼트에 의해 수행되는 동작들을 나타낸다.

표 37

(a)	*NOP*;	병렬 명령에 대한 NOP
(b)	*TLOOP*;	Tight 루프(단일명령 루프)
(c)	$\{R0-R7\}=\begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix}$;	데이터 레지스터 이동
(d)	$\{R0-R7\}=\{A0-A3\}$;	어드레스 레지스터를 데이터 레지스터로 이동
(e)	$\begin{Bmatrix} A0-A3 \\ AM0-AM1 \end{Bmatrix}=\{R0-R3\}$;	데이터 레지스터를 AGU 레지스터로 이동
(f)	$\{A0-A3\}=\{A0-A3\}$; $\{B0-B3\}=\{B0-B3\}$; $\{C0-C3\}=\{C0-C3\}$;	어드레스 레지스터 이동
(g)	*IF cond*	병렬 명령의 상태

DMOVA 명령 서브프래그먼트 포맷

표 38은 본 발명의 예시적인 실시예에 따른 DMOVA 명령 서브프래그먼트의 포맷을 제공한다.

표 38

	7	6	5	4	3	2	1	0
(a,b,c)	0	R0-R7 (dst)			dreg (src)			
(d)	1	0	0	0	A0-A3 (src)		R0-R3 (dst)	
(e)	1	0	0	1	A0-A3 (dst)		R0-R3 (src)	
(f)	1	0	1	0	A0-A3 (dst)		A0-A3 (src)	
(f)	1	0	1	1	B0-B3 (dst)		B0-B3 (src)	
(f)	1	1	0	0	C0-C3 (dst)		C0-C3 (src)	
(d)	1	1	0	1	A0-A3 (src)		R4-R7 (dst)	
(g)	1	1	1	cond				

주해: 명령 R0=R0는 NOP로 복호화
명령 R1=R1은 TLOOP로 복호화
명령 A0=A0는 AM0=R0로 복호화
명령 A1=A1은 AM0=R1으로 복호화
명령 A2=A2는 AM0=R2로 복호화
명령 A3=A3는 AM0=R3로 복호화
명령 B0=B0는 AM1=R0로 복호화
명령 B1=B1은 AM1=R1으로 복호화
명령 B2=B2는 AM1=R2로 복호화
명령 B3=B3는 AM1=R3로 복호화

DMOVA 명령 서브프래그먼트 포맷

이와 같이, 하나 이상의 명령 서브프래그먼트를 포함할 수 있는 MMPF 명령 프래그먼트들을 제공함에 의해, 전 명령을 사용하여 수행될 수 있는 동작들의 수는 더 증가된다. 예를 들어, 전 명령은 산술 및 MAC 동작들이 일군의 3개의 메모리 무브 및 프로그램 플로우 동작들에 따라 수행되게 할 수 있다. 단일 명령을 사용하여 이러한 많은 동작들을 수행하는 능력은 소정의 동작을 수행할 필요가 있는 전 명령들의 수를 더 감소시켜 DSP 상에서 요청되는 전 명령 메모리를 감소시킨다. 명령 메모리를 감소시키면 다이(die)크기, DSP의 비용 및 소비 전력이 감소되어, 이러한 DSP를 이동 무선 전화를 포함하는 많은 다양한 애플리케이션에 더 적합하게 한다.

이와 같이, 고도의 평행 변수 길이 명령 집합을 사용하여 DSP를 제어하는 시스템 및 방법이 기술되었다. 임의의 상기 기술에 숙련된 자들에게 본 발명을 제작 또는 사용할 수 있는 바람직한 실시예에 대한 설명이 이전에 제공되었다. 이러한 실시예들에 대한 다양한 변경들은 용이하게도 상기 기술에 숙련된 자들에게 명백하며, 본원에서 규정된 일반적인 원리들은 본 발명의 기능을 사용하지 않고 다른 실시예들에 적용될 수 있다. 예를 들어, 상기 시스템 및 방법이 DSP의 내용에 기술되지만, 그것의 다양한 특성들은 일반적인 컴퓨팅 시스템 및 장치들에 적용가능하다.

이와 같이 바람직한 실시예들를 참조하여 본 발명이 기술되었으므로, 당면된 실시예들은 단지 예시적이며 적절한 지식 및 기술을 가진 자들에게 일어날 수 있는 이와 같은 그러한 변경 및 수정이 첨부된 청구범위 및 그와 동등한 것에 기술된 바와 같이 본 발명의 사상 및 범위를 이탈하지 않고 행해질 수 있다.

도면의 간단한 설명

[0019] 본 발명의 특징, 목적 및 장점은 동일한 참조번호가 전체를 통하여 대응하여 동일시되는 도면과 함께 주어진 하

기에서 설명되는 본 발명의 예시적인 실시예의 상세한 설명으로부터 더욱더 명백해질 것이다.

[0020] 도 1은 종래 기술에 따라 구성된 디지털 신호 처리기의 블록도;

[0021] 도 2는 본 발명을 실시하는 디지털 신호 처리기의 블록도;

[0022] 도 3은 레지스터 뱅크의 레지스터와 입력 포트 사이의 접속에 대한 블록도;

[0023] 도 4는 레지스터 뱅크의 출력 포트와 레지스터 사이의 접속에 대한 블록도;

[0024] 도 5는 본 발명의 실시예에 따른 메모리 공간에 저장된 가변 길이 명령 집합의 다이어그램;

[0025] 도 6은 명령 인출부의 동작을 설명하는 흐름도;

[0026] 도 7은 본 발명의 실시예에 따라 구성된 경우 명령 인출부의 블록도;

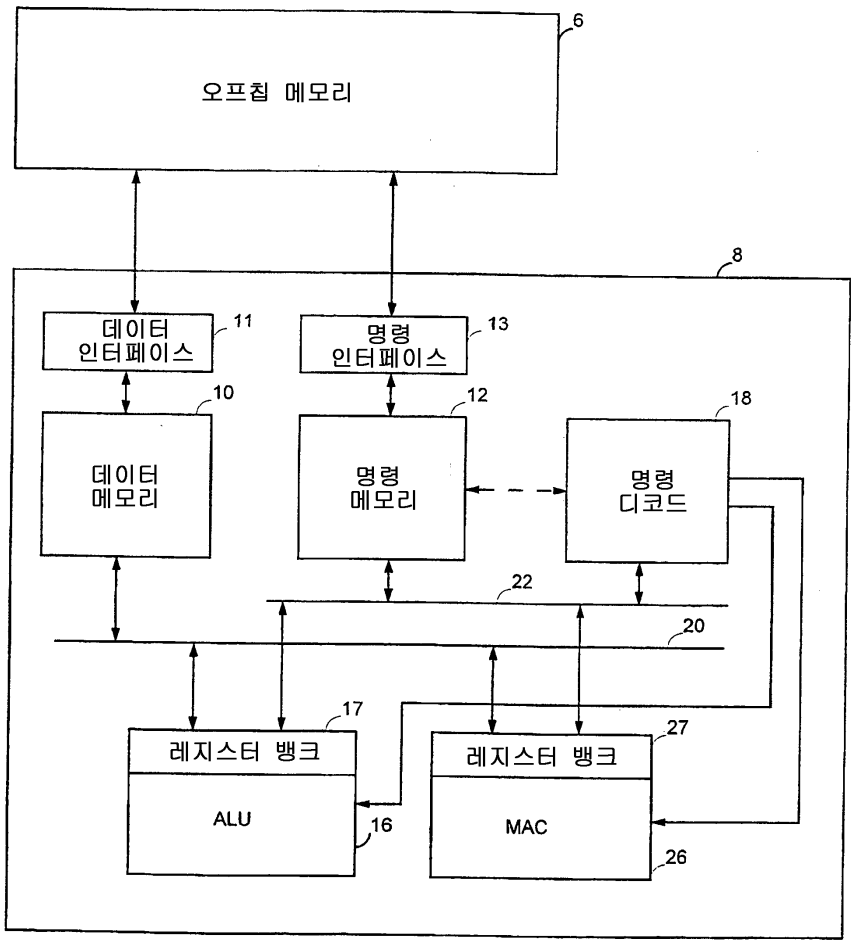
[0027] 도 8은 본 발명의 실시예에 따라 구성된 경우 MAC 부의 블록도;

[0028] 도 9는 본 발명의 실시예에 사용된 명령 계층구성의 블록도.

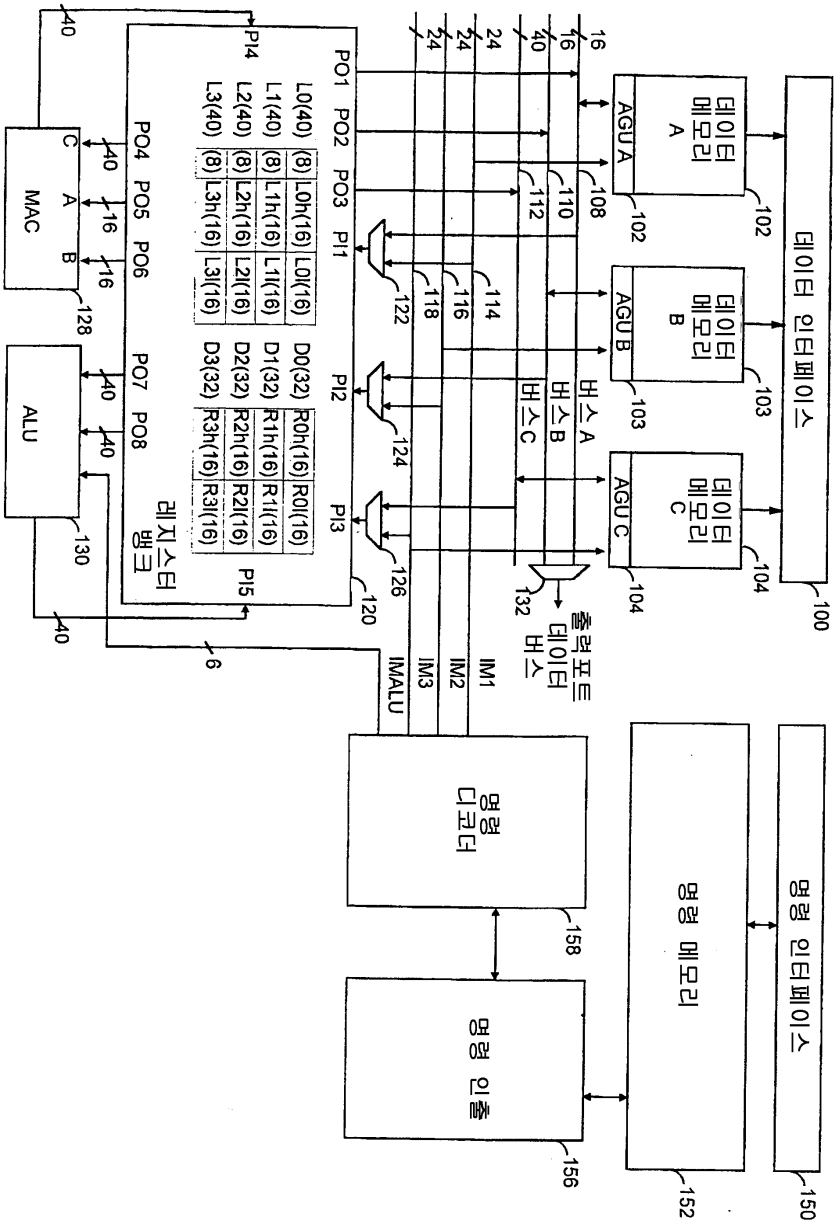
도면

도면1

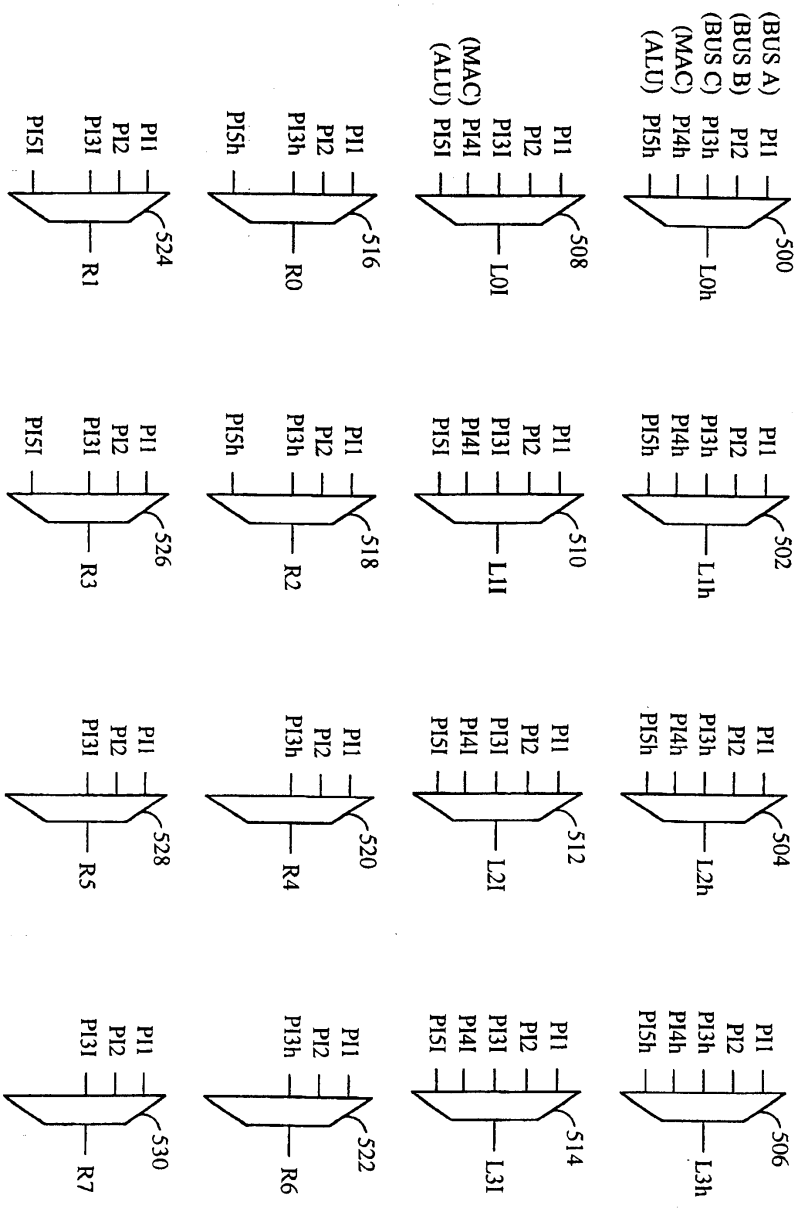
종래 기술



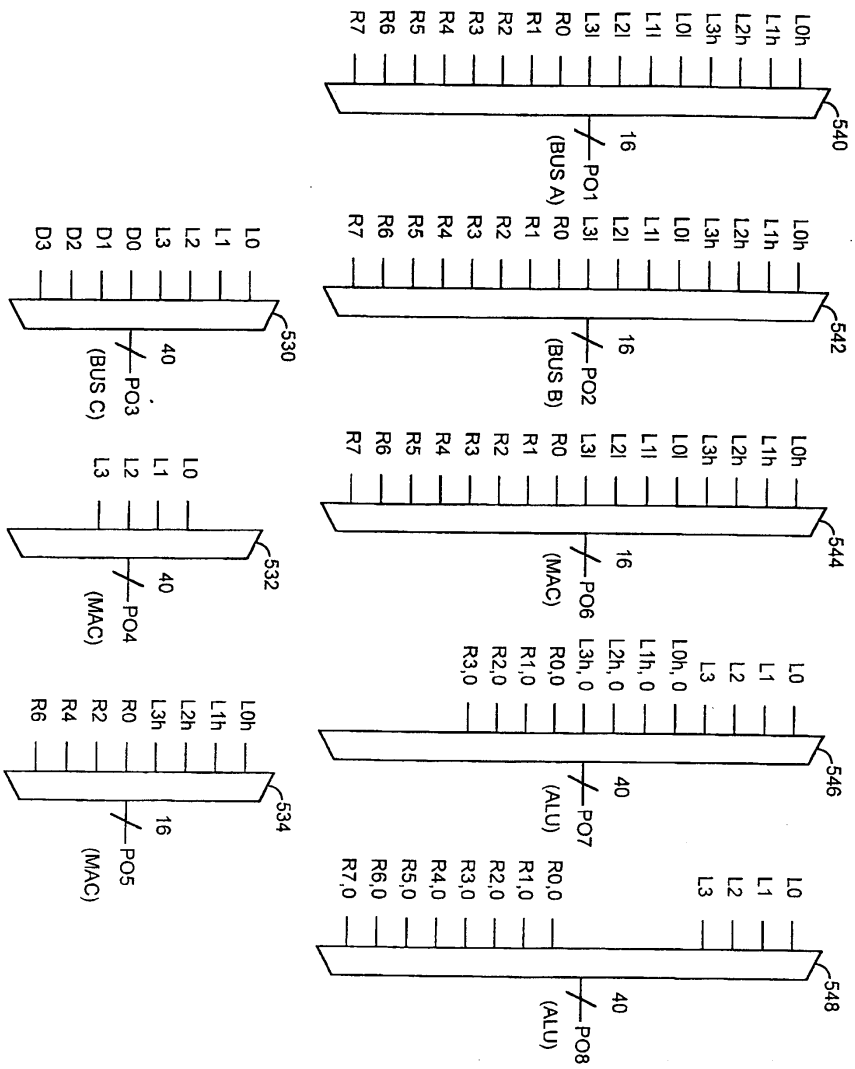
도면2


데이터 인터페이스
100
데이터 메모리 A
AGU A
102
데이터 메모리 B
AGU B
103
데이터 메모리 C
AGU C
104
버스 A
버스 B
버스 C
출력포트 데이터 버스
132
레지스터 뱅크
PO1 PO2 PO3 PO4 PO5 PO6 PO7 PO8
PI1 PI2 PI3 PI4 PI5
L0(40) L1(40) L2(40) L3(40)
R0(16) R1(16) R2(16) R3(16)
D0(32) D1(32) D2(32) D3(32)
IM1 IM2 IM3 IMALU
MAC
ALU
명령 디코더
158
명령 인출
156
명령 메모리
152
명령 인터페이스
150

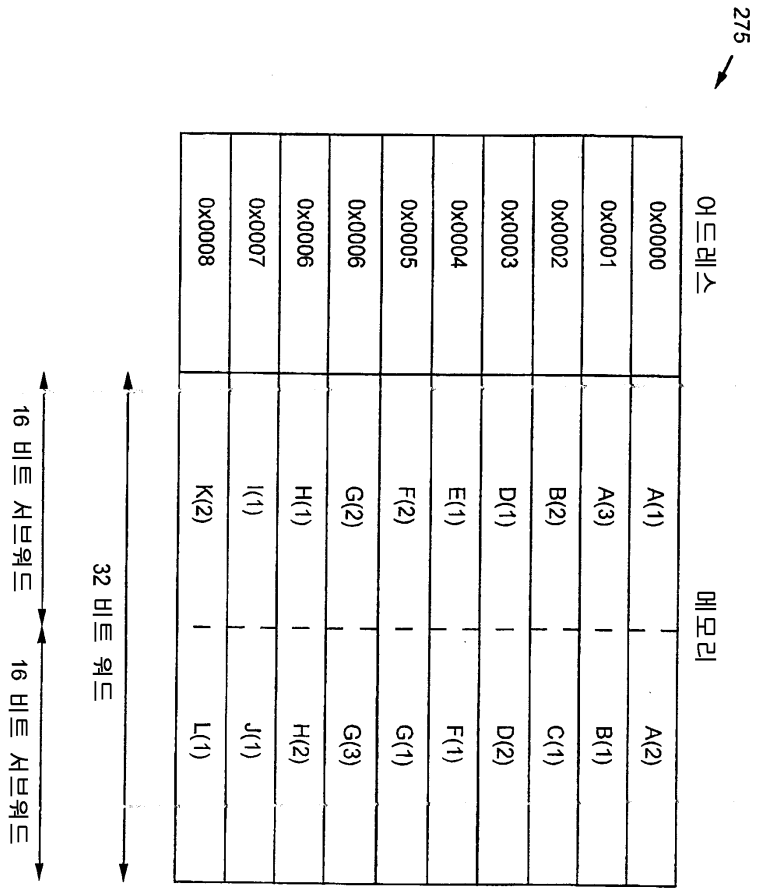
도면3


(BUS A) PI1
(BUS B) PI2
(BUS C) PI3h
(MAC) PI4h
(ALU) PI5h
L0h
500
PI1
PI2
PI3h
PI4h
PI5h
L1h
502
PI1
PI2
PI3h
PI4h
PI5h
L2h
504
PI1
PI2
PI3h
PI4h
PI5h
L3h
506
PI1
PI2
PI3I
(MAC) PI4I
(ALU) PI5I
L0I
508
PI1
PI2
PI3I
PI4I
PI5I
L1I
510
PI1
PI2
PI3I
PI4I
PI5I
L2I
512
PI1
PI2
PI3I
PI4I
PI5I
L3I
514
PI1
PI2
PI3h
PI5h
R0
516
PI1
PI2
PI3h
PI5h
R2
518
PI1
PI2
PI3h
PI5h
R4
520
PI1
PI2
PI3h
PI5h
R6
522
PI1
PI2
PI3I
PI5I
R1
524
PI1
PI2
PI3I
PI5I
R3
526
PI1
PI2
PI3I
PI5I
R5
528
PI1
PI2
PI3I
PI5I
R7
530

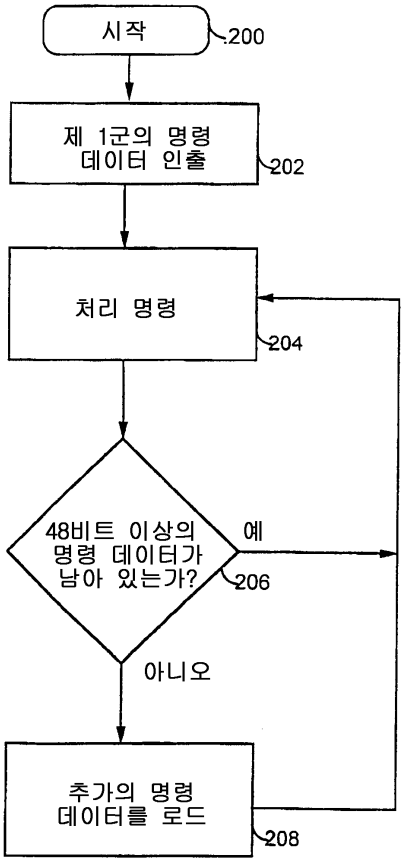
도면4


L0h L1h L2h L3h L0l L1l L2l L3l R0 R1 R2 R3 R4 R5 R6 R7
540
16
PO1
(BUS A)
542
PO2
(BUS B)
544
PO6
(MAC)
546
L0 L1 L2 L3 L0h, 0 L1h, 0 L2h, 0 L3h, 0 R0,0 R1,0 R2,0 R3,0
40
PO7
(ALU)
548
L0 L1 L2 L3 R0,0 R1,0 R2,0 R3,0 R4,0 R5,0 R6,0 R7,0
40
PO8
(ALU)
530
L0 L1 L2 L3 D0 D1 D2 D3
40
PO3
(BUS C)
532
L0 L1 L2 L3
40
PO4
(MAC)
534
L0h L1h L2h L3h R0 R2 R4 R6
16
PO5
(MAC)

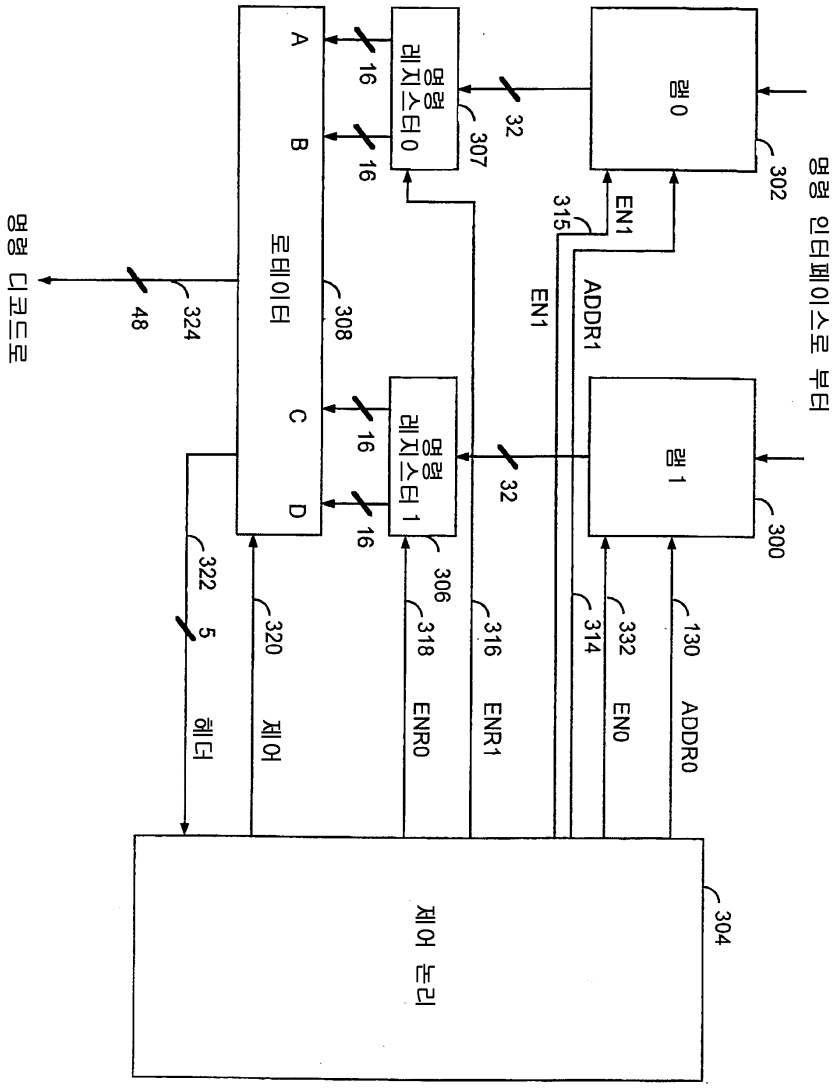
도면5


275
어드레스
메모리
0x0000
0x0001
0x0002
0x0003
0x0004
0x0005
0x0006
0x0006
0x0007
0x0008
A(1)
A(2)
A(3)
B(1)
B(2)
C(1)
D(1)
D(2)
E(1)
F(1)
F(2)
G(1)
G(2)
G(3)
H(1)
H(2)
I(1)
J(1)
K(2)
L(1)
32 비트 워드
16 비트 서브워드
16 비트 서브워드

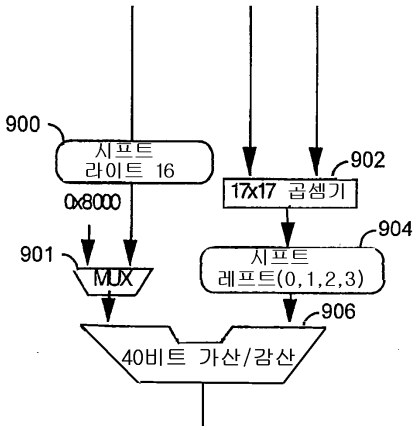
도면6


시작
200
제 1군의 명령 데이터 인출
202
처리 명령
204
48비트 이상의 명령 데이터가 남아 있는가?
206
예
아니오
추가의 명령 데이터를 로드
208

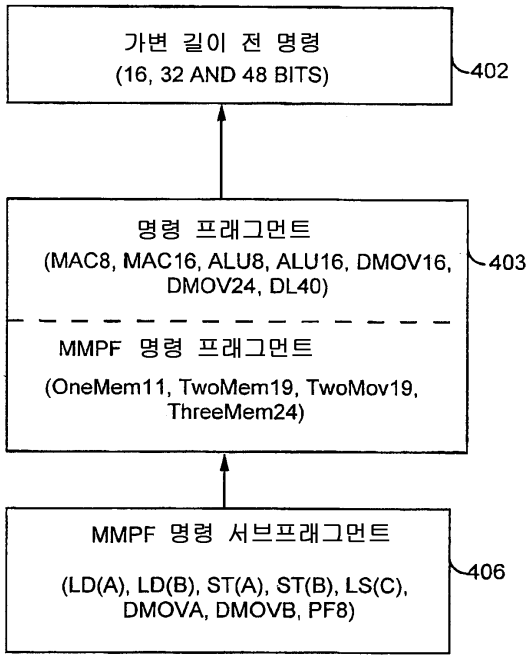
도면7


램 0
302
램 1
300
명령 인터페이스로 부터
명령 레지스터 0
307
명령 레지스터 1
306
32
16
A
B
C
D
로테이터
308
명령 디코드로
48
324
EN1
315
ADDR1
322
5
헤더
320
제어
318
ENR0
316
ENR1
314
332
EN0
130
ADDR0
제어 논리
304

도면8


900
시프트 라이트 16
0x8000
901
MUX
902
17x17 곱셈기
904
시프트 레프트(0,1,2,3)
906
40비트 가산/감산

도면9


가변 길이 전 명령
(16, 32 AND 48 BITS)
402
명령 프래그먼트
(MAC8, MAC16, ALU8, ALU16, DMOV16, DMOV24, DL40)
403
MMPF 명령 프래그먼트
(OneMem11, TwoMem19, TwoMov19, ThreeMem24)
MMPF 명령 서브프래그먼트
(LD(A), LD(B), ST(A), ST(B), LS(C), DMOVA, DMOVB, PF8)
406