(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl. [6]
G06T 1/00

(45) 공고일자 2003년01월08일
(11) 등록번호 10-0348950
(24) 등록일자 2002년08월02일

(21) 출원번호 10-1994-0032081
(22) 출원일자 1994년11월30일

(65) 공개번호 특1995-0015137
(43) 공개일자 1995년06월16일

(30) 우선권주장 08/159285 1993년11월30일 미국(US)

(73) 특허권자
텍사스 인스트루먼츠 인코포레이티드
미국 텍사스주 75265 달라스 노스센트럴 익스프레스웨이 13500

(72) 발명자
카알엠.거태그
미합중국 77459 텍사스주 미조리 시티 스우스 샌디 코트 4015
리차드심프슨
영국 잉글랜드 엠케이43 7제이유 베츠포드 칼론 카우티 더 마쉬 21
브렌다왈시
영국잉글랜드 엠케이41 비에이치티 베드포드 칠링햄 그린 14

(74) 대리인
장수길
주성민

심사관 : 전상현

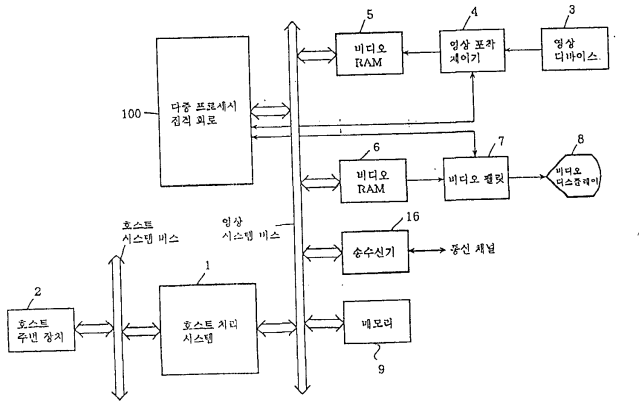
(54) 데이타처리장치

요약

3 입력 산술 논리 유닛(230)은 3 다중비트 입력 신호의 혼합된 산술 및 부울 조합을 형성한다. 상기 산술 논리 유닛(230)은 먼저 부울 조합을 형성한 후 산술 조합을 형성한다. 현재의 명령은 형성된 조합을 제어하는 함수 신호 F0-F7를 생성하는 명령 디코더 (250, 245)를 구동시킨다. 3 입력 산술 논리 유닛(230)은 바람직하게 비트 회로(400)의 세트를 이용하는데, 각 비트 회로는 캐리 전파, 생성 및 킬 신호를 형성한다. 이들 신호는 다단 논리 트리 회로 및 합성 결과를 만들기 위해 캐리 입력 및 다음 비트 회로를 만들기 위해 캐리 출력과 함께 이용될 수 있다. 이와 같은 구조로써 현재 명령에 기초하여 3 입력 신호의 선택된 산술, 부울 또는 혼합된 산술 및 부울 함수를 형성할 수 있게 된다. 함수 신호의 선택은 입력 신호중 한개의 입력 신호가 조합에 반응하지 않게 함으로써, 나머지 입력 신호중 두 입력 함수를 수행한다. 명령 그 자체는 함수 신호 및 함수 수정 비트를 포함할 수 있거나, 또는 함수 신호 및 함수 수정 신호는 특수 데이타 레지스터에 저장될 수 있다. 함수 수정 신호는 사용하기 전에 함수 신호를 수정시키게 된다. 3 입력 산술 논리 입력(203)은 최하위 비트로 캐리 입력을 공급하는 최하위 비트 캐리-인 생성기(246)를 포함한다. 이 캐리 입력은 형성되는 조합에 의해 결정되고, 일반적으로 감산 동안만은 " 1" 이다. 이 캐리 입력은 임의의 명령에 대해 특수 목적의 데

이타 레지스터(D0)에 지정될 수 있다. 형성된 조합은 입력중 한 입력의 부호 비트에 따라 선택적으로 수정될 수 있다.

대표도



명세서

도면의 간단한 설명

제1도는 본 발명에 사용되는 영상 처리 시스템의 시스템 아키택쳐를 나타내는 도면.

제2도는 본 발명의 양호한 실시예를 형성하는 하나의 집적 회로 멀티프로세서의 아키택쳐를 나타내는 도면.

제3도는 제2도에 도시한 디지탈 영상/그래픽 프로세서 중 하나를 블럭 다이어그램 형태로 나타낸 도면.

제4도는 제2도에 도시한 디지탈 영상/그래픽 프로세서의 연산의 파이프라인 단(stages)을 개략적인 형태로 나타낸 도면.

제5도는 제3도에 도시한 디지탈 영상/그래픽 프로세서의 데이타 유닛을 블럭 다이어그램 형태로 나타낸 도면.

제6도는 제5도에 도시한 데이타 유닛의 상태 레지스터의 필드 정의를 개략적인 형태로 나타낸 도면.

제7도는 제5도에 도시한 데이타 유닛의 산술 논리 유닛을 분할하는 방법을 블럭 다이어그램 형태로 나타낸 도면.

제8도는 회전 레지스터로서 제5도에 도시한 데이타 유닛의 데이타 레지스터를 어드레싱하는 방법을 블럭 다이어그램 형태로 나타낸 도면.

제9도는 제5도에 도시한 데이타 유닛의 제1 데이타 레지스터의 필드 정의를 개략적인 형태로 나타낸 도면.

제10도는 제5도에 도시한 데이타 유닛의 산술 논리 유닛의 1비트 회로의 구성을 블럭 다이어그램 형태로 나타낸 도면.

제11도는 제10도에 도시한 비트 회로의 결과 논리(resultant logic) 및 캐리-아웃 논리(carry out logic)의 구성을 개략적인 형태로 나타낸 도면.

제12도는 제10도에 도시한 비트 회로의 부울(Boolean) 함수 발생기의 구성을 개략적인 형태로 나타낸 도면.

제13도는 제5도에 도시한 데이타 유닛의 함수 신호 발생기의 함수 신호 셀렉터를 블럭 다이어그램 형태로 나타낸 도면.

제14도는 제5도에 도시한 데이타 유닛의 함수 신호 발생기의 함수 신호 수정 부분을 블럭 다이어그램 형태로 나타낸 도면.

제15도는 제5도에 도시한 데이타 유닛의 비트 제로(0) 캐리-인 발생기를 블럭 다이어그램 형태로 나타낸 도면.

제16도는 제10도 및 제11도에 도시한 산술 논리 유닛의 개념도를 블럭 다이어그램 형태로 나타낸 도면.

제17도는 산술 논리 유닛의 다른 실시예의 개념도를 블럭 다이어그램 형태로 니타낸 도면.

제18도는 제3도에 도시한 디지탈 영상/그래픽 프로세서의 연산을 제어하는 명령 워드를 개략적인 형태로 나타낸 도면.

제19도는 컬러 팩시밀리 시스템 내에 하나의 디지탈 영상/그래픽 프로세서를 갖는 본 발명의 멀티프로세서 집적 회로를 개략적인 형태로 나타낸 도면.

도면의 주요 부분에 대한 부호의 설명

1 : 호스트 처리 시스템 2 : 호스트 주변 장치

3 : 영상화 장치 4 : 영상 포착 제어기

5, 6 : 비디오 RAM 7 : 비디오 팰릿

8 : 비디오 디스플레이 9 : 메모리

16 : 송수신기 60 : 마스터 프로세서

71, 71, 73, 74 : 디지탈 영상/그래픽 프로세서

80 : 전송 제어기 90 : 프레임 제어기

100 : 멀티프로세서 집적 회로

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명의 기술 분야는 디지탈 데이타 처리, 특히 디지탈 데이타 처리 특히 디지탈 영상/그래픽 처리를 위한 마이크로프로세서 회로, 아키텍처 및 그 방법에 관한 것이다.

본 발명은 컴퓨터 그래픽 분야에 관한 것으로, 구체적으로는 비트맵 그래픽에 관한 것이다. 비트맵 그래픽에서는 연산 메모리가 영상의 각 개개의 픽셀 또는 픽셀용 데이타를 영상 내에 상기 픽셀의 위치에 대응하는 메모리 위치에 저장한다. 이러한 영상은 영상으로서 표시되거나 또는 포착된(captured) 영상으로서 조정, 저장, 표시 또는 재전송되게 된다. 비트맵 컴퓨터 그래픽의 분야는 동적 등속 호출 메모리(DRAM)의 비용 절감 및 유량 증가 그리고 마이크로프로세서의

비용 절감 및 처리 능력 면에서 아주 유리하다. 구성 부분의 비용 및 성능에 있어서 이러한 유리한 변화로 인해 크고 보다 복잡한 컴퓨터 영상 시스템을 경제적으로 구현할 수 있다.

비트맵 그래픽 분야는 영상 데이타 조정을 위한 처리 형태에 있어서 여러 단계의 발전이 있었다. 초기에 비트맵 그래픽을 지원하는 컴퓨터 시스템은 모든 비트맵 연산을 위해 시스템 프로세서를 사용했다. 그러나, 이런 형태의 시스템은 여러 가지 결점이 있었다. 먼저, 컴퓨터 시스템 프로세서는 비트맵 그래픽을 처리하기 위해 특별히 설계된 것이 아니었다. 범용의 연산을 위해 매우 합리적으로 선택한 디자인은 비트맵 그래픽 시스템에는 잘 들어맞지 않았다. 결국 몇몇의 루틴(routine)한 그래픽 임무에 대한 처리 속도가 늦어지게 되었다. 또한, 비트맵 그래픽의 영상 조정에 필요한 처리가 시스템 프로세서의 연산 능력에 부담이 되어 다른 연산이 또한 늦게되는 것이 발견되었다.

비트맵 그래픽 처리의 발전의 다음 단계는 전용 하드웨어 그래픽 제어기였다. 이들 장치는 시스템 프로세서의 제어 하에서, 선, 타원 및 원과 같은 간단한 그림을 그릴 수 있다. 또한 이들 장치의 대다수는 픽셀 블럭 전송(PixBlt)을 할 수 있다. 픽셀 블럭 전송은 메모리의 한 부분으로부터 다른 부분으로 영상 데이타의 메모리 이송 연산이다. 픽셀 블럭 전송은 특정한 형태의 폰트의 영숫자와 같은 표준 영상 요소를, 비표시된 메모리로부터 비트맵 디스플레이 메모리로 전송시켜 디스플레이 내에 표시하는데 유용하다. 이러한 기능은 동일한 작은 영상을 비트맵 디스플레이 메모리의 전체에까지 전송함으로써 타일링(tiling)하는데 사용될 수 있다. 가장 자주 사용되는 그래픽 기능의 몇몇을 수행하기 위한 내장(built-in) 알고리즘은 시스템 성능을 개선하는 방법을 제공한다. 그러나, 종종 유용한 그래픽 컴퓨터 시스템은 상기한 하드웨어 그래픽 제어기에 구성되는 이들 몇몇의 기능 외에 많은 기능을 필요로 한다. 이들 추가의 기능은 시스템 프로세서에 의해 소프트웨어로 구성되어야 한다. 통상적으로 이들 하드웨어 그래픽 제어기는 시스템 프로세서로 하여금 비트맵 메모리에 대한 제한적인 접근만을 할 수 있도록 함으로써, 시스템 소프트웨어가 하드웨어 그래픽 제어기의 일정한 기능 세트를 증가(augment)시킬 수 있을 정도로 제한한다.

그래픽 시스템 프로세서는 비트맵 그래픽 처리에 있어서 또 다른 발전 단계를 거쳤다. 그래픽 시스템 프로세서는 프로세서의 모든 속성을 갖는 프로그래머블 장치이며, 비트맵 그래픽용의 특수 기능을 포함한다. 텍사스 인스트루먼츠사에서 제조한 TMS34010 및 TMS34020 그래픽 시스템 프로세서는 장치의 이러한 분류를 나타낸다. 이들 그래픽 시스템 프로세서는 마이크로프로세서와 동일한 방식으로 저장된 프로그램에 응답하며, 산술 논리 유닛을 통해 데이타를 조정할 수 있는 능력, 레지스터 파일에의 데이타 저장 그리고 프로그램 흐름 및 외부 데이타 메모리의 제어를 포함한다. 아울러, 이들 장치는 프로그램 제어하에 연산하는 특수 목적의 그래픽 조정을 포함한다. 이들 그래픽 시스템 프로세서의 명령 세트내의 추가의 명령이 특수 목적의 그래픽 하드웨어를 제어한다. 이들 명령 및 이들 명령을 지원하는 하드웨어는 많은 문맥(context)에 유용한 기본 레벨의 그래픽 기능을 수행하도록 선택된다. 이와 같은 방식으로 그래픽 시스템 프로세서는 특정한 문제용으로 선택된 알고리즘을 사용하여 많은 다른 그래픽 응용을 위해 프로그램될 수 있다. 이것은 하드웨어 제어기로부터 프로그램된 마이크로프로세서로 변경함으로써 제공되는 유용성과 유사한 유용성의 향상을 제공한다. 상기한 그래픽 시스템 프로세서는 마이크로프로세서와 동일한 방식의 프로그래머블 장치이므로, 독립형(stand alone) 그래픽 프로세서, 즉 시스템 프로세서에 종속된 그래픽 코-프로세서(co-processor) 또는 밀착 결합된 그래픽 제어기로서 연산할 수 있다.

새로운 장치는 보다 강력한 그래픽 기능을 제공할 수 있는 소망을 추구하고있다. 몇몇의 분야는 경제적으로 실행가능한 비용면에서 효과적인 그래픽 연산을 필요로 한다. 이것은 전동 비디오(full motion vidio), 고품위 텔레비젼, 컬러 팩시밀리 및 디지탈 포토그래피와 데이타의 수수를 행하는 비디오 영상 멀티 매체를 포함한다. 이들 분야의 각각에는 독특한 문제점들이 존재하는데, 영상 데이타의 압축 및 분해가 공통된 사항이다. 전송 밴드폭의 양 그리고 영상 및 특정한 전동 비디오에 요구되는 저장 용량의 양은 엄청나다. 최종적으로 수용가능한 최종의 영상의 질을 얻을 수 있는 효과적인 비디오 압축 및 분해가 없으면, 이들 장치는 전달 밴드폭 및 저장 용량과 관련된 비용에 의해 한계에 이르게 된다.

또한, 단일 시스템에 대한 종래 기술에서는 영상 인식과 같은 영상 처리 기능 및 디스플레이 제어와 같은 그래픽 기능을 지원할 필요성이 있다.

3 입력 산술 논리 입력 유닛은 3 다중비트 입력 신호의 혼합된 산술 및 부울 조합을 형성한다. 먼저 이 산술 논리 유닛은 부울 조합을 형성한 후, 바람직하게는 식 A& F1(B,C) + F2(B,C)에 따라 산술 조합을 형성한다. 부울 조합 F1(B,C)와 F2(B,C)는 현재의 명령에 의해 두 입력의 가능한 모든 부울 조합의 세트로부터 독립적으로 선택된다.

현재의 명령은 함수 코드 비트 F0-F7를 생성하는 명령어 디코더를 구동시킨다. 상기 함수 코드 비트 F0-F7은 형성된 조합을 제어한다. 바람직하게는 3입력 산술 논리 유닛은 한 세트의 비트 회로를 이용한다. 부울 함수 생성기는 캐리 전파, 생성 및 킬 신호(carry propagate, generate and kill signal)를 형성한다. 상기 부울 함수 생성기는 다단 논리 트리 회로로 구현될 수 있다. 이들 신호는 캐리 입력과 결합되어 비트 결과 및 캐리를 만들어 다음 비트 회로로 출력한다. 캐리 생성 및 캐리 킬은 산술 인에이블 신호가 존재할 때만 인에이블될 수 있다. 이와 같은 구조는 현재 명령에 기초하여 3 입력 신호의 선택된 산술, 부울 또는 혼합된 산술 및 부울 함수의 형성을 가능케 한다. 가능한 조합에는 A+F(B,C) ; A-F(B,C) ; F(C, A+B, A-B) 및 F1(A,B) ± F2(A,C)가 있다. 함수 코드 비트의 선택으로 입력 신호 중 한 입력 신호에 조합이 응답하지 않음으로써, 나머지 입력 신호의 2 입력 함수를 수행하게 된다. 명령어 그 자체는 함수 신호 및 함수 수정 비트를 포함할 수 있거나, 또는 함수 신호 및 함수 수정 신호는 특수 데이타 레지스터에 저장될 수 있다. 함수 수정 신호는 사용하기 전에 함수 신호의 수정을 초래한다. 이들은 입력 신호 중 한 입력 신호를 효과적으로 수정할 수 있다. 특수한 유형의 명령은 사용시 특수 레지스터에 저장된 함수 신호를 반전시킬 수 있다.

3 입력 산술 논리 유닛은 캐리 입력을 최하위 비트에 공급하는 최하위 비트 캐리-인 생성기를 포함한다. 이 캐리 입력은 형성되는 조합에 의해 결정된다.

만약 F2(B,C)가, 일반적으로 감산하는 동안 B 또는 C의 반전, 또는 B와 C의 부울 조합의 반전을 포함하면, 이때 캐리 입력은 "1"이다. 그렇지 않으면, 캐리 입력은 "0"이다. 캐리 입력은 임의의 명령을 위해 특수 목적의 데이타 레지스터에 지정될 수 있다.

형성된 조합은 입력 중 한 입력의 부호 비트에 따를 수 있다. 함수 신호는 부호에 따른 명령을 실행할 때 바람직하게 서브셋의 반전에 의해 수행된다.

본 발명의 바람직한 실시예에서, 3 입력 산술 논리 유닛은 영상 처리에 사용되는 단일 집적 회로에 형성된 다중프로세서의 일부로서 적어도 한개의 디지탈 영상/그래픽 프로세서에 구현된다.

제1도는 본 발명에 따른 영상 및 그래픽 처리용으로 구성되는 멀티프로세서 집적 회로를 포함하는 영상 데이타 처리 시스템의 블럭도이다. 이 데이타 처리 시스템은 호스트 처리 시스템(1)을 포함한다. 호스트 처리 시스템(1)은 제1도의 데이타 처리 시스템의 호스트 시스템을 위한 데이타 처리를 제공한다. 호스트 처리 시스템(1)내에는, 프로세서, 적어도 하나의 입력 장치, 장기간 저장 장치, 판독 전용 메모리, 등속 호출 메모리 및 호스트 시스템 버스에 결합되는 적어도 하나의 호스트 주변 장치(2)가 포함된다. 호스트 처리 시스템의 구성 및 연산은 종래 구성으로 간주된다. 그 처리 기능으로 인해 호스트 처리 시스템(1)은 영상 데이타 처리 시스템의 기능을 제어한다.

멀티프로세서 집적 회로(100)는 데이타 조정 및 제1도의 영상 데이타 처리 시스템의 영상 조작을 위한 연산을 포함하는 대부분의 데이타 처리를 제공한다. 멀티프로세서 집적 회로(100)는 영상 시스템 버스에 양방향 접속되며 영상 시스템 버스를 통해 호스트 처리 시스템과 교신한다. 제1도의 구성에 있어서, 멀티프로세서 집적 회로(100)는 호스트 처리 시스템(1)과 독립적으로 연산한다. 그러나, 멀티프로세서 집적 회로(100)는 호스트 처리 시스템(1)에 응답할 수 있다.

제1도는 2개의 영상 시스템을 나타낸다. 영상화 장치(3)는 문서 스캐너, 전하 결합 소자 스캐너 또는 영상화 입력 장치로서 작용하는 비디오 카메라를 나타낸다. 영상화 장치(3)는 영상을 디지탈화하여 그것을 라스터 주사 프레임(raster scan frame)속에 형성하는 영상 포착 제어기(4)로 상기 영상을 공급한다. 이 프레임 포착 공정은 멀티프로세서 집접 회로(100)로부터의 신호에 의해 제어된다. 이렇게 형성된 영상 프레임은 비디오 등속 호출 메모리(5)에 저장된다. 비디오 등속 호출 메모리(5)는 멀티프로세서 집적 회로(100)에 의한 영상 처리를 위한 데이타 전송을 가능하게 하는 영상 시스템 버스를 통해 액세스될 수 있다.

제2 영상 시스템은 비디오 디스플레이를 구동한다. 멀티프로세서 집적 회로(100)는 픽셀 맵을 통해 표시된 영상의 사양에 대해 비디오 등속 호출 메모리(6)와 교신한다. 멀티프로세서 집적 회로(100)는 영상 시스템 버스를 통해 비디오 등속 호출 메모리에 저장된 영상 데이타를 제어한다. 이 영상에 대응하는 데이타는 비디오 등속 호출 메모리(6)로부터 재현되어 비디오 팰릿(7)에 공급된다. 비디오 팰릿(7)은 이렇게 재현된 데이타를 다른 컬러 공간으로 수정시킬 수 있으며, 픽셀 당 비트수 등을 확장시킬 수 있다. 이러한 변환은 룩-업 테이블을 통해 달성될 수 있다. 또한 비디오 팰릿(7)은 비디오 디스플레이(8)를 구동하기 위하여 적절한 비디오 신호를 발생시킨다. 이들 비디오 신호가 아날로그 신호이면, 비디오 팰릿(7)은 적절한 디지탈 대 아날로그 변환을 포함한다. 비디오 팰릿(7)으로부터 출력된 비디오 레벨 신호는 컬러, 채도 및 명도 정보를 포함할 수 있다. 멀티프로세서 집적 회로(100)는 비디오 팰릿(7) 내에 저장된 데이타를 제어함으로, 데이타 변환 처리 및 영상 프레임의 타이밍을 제어하게 된다, 멀티프로세서 집적 회로(100)는 비디오 팰릿(7)의 제어를 통해 라인 길이 및 비디오 디스플레이 영상의 프레임당 라인 수, 동기화, 리트레이스(retrace) 및 블랭킹 신호를 제어한다. 특히, 멀티프로세서 집적 회로(100)는 그래픽 표시 정보가 비디오 등속 호출 메모리(6)에 저장되어 있는 장소를 판단하고 제어한다. 이어서, 비디오 등속 호출 메모리(6)로부터의 판독 동안, 멀티프로세서 집적 회로(100)는 비디오 등속 호출 메모리(6)로부터 판독 순서, 액세스 될 어드레스, 그리고 비디오 디스플레이(8)상에 소망의 그래픽 영상를 발생시키는데 필요한 제어 정보를 결정한다.

비디오 디스플레이(8)는 유저가 볼 수 있는 특정한 비디오 표시를 발생시킨다. 흔히 사용되는 2가지 기술이 있다. 첫번째 기술은 각 픽셀에 대한 컬러, 색조, 명도 그리고 채도와 관련하여 비디오 데이타를 특정하는 것이다. 두번째 기술은 적색, 청색, 녹색의 컬러 레벨을 각 픽셀에 대해 특정하는 것이다. 비디오 팰릿(7) 및 비디오 디스플레이(8)는 선택된 기술과 호환될 수 있도록 설계 및 제작되었다.

제1도는 영상 시스템 버스 결합된 부가 메모리(9)를 나타낸다. 이 부가 메모리는 부가 비디오 등속 호출 메모리, 동적 등속 호출 메모리, 정적 등속 호출 메모리 또는 판독 전용 메모리를 포함할 수 있다. 멀티프로세서 집적 회로(100)는 메모리(9)에 저장된 프로그램에 의해 전체적으로 또는 부분적으로 제어될 수 있다. 이 메모리(9)는 그래픽 영상 데이타의 여러 형태를 저장할 수 있다. 또, 멀티프로세서 집적 회로(100)는 비디오 등속 호출 메모리, 동적 등속 호출 메모리 및 정적 등속 호출 메모리용 메모리 인터페이스 회로를 포함한다. 이와 같이 시스템은 임의의 비디오 등속 호출 메모리(5 또는 6)없이 멀티프로세서 집적 회로(100)에 의해 구성될 수 있다.

제1도는 송수신기(16)를 나타낸다. 이 송수신기(16)는 영상 시스템 버스와 통신 채널간의 변환 및 양방향 통신을 제공한다. 송수신기(16)를 사용하는 시스템의 일예는 비디오 회의이다. 제1도에 도시한 영상 데이타 처리 시스템은 제1위치에서 사람의 비디오 영상을 형성하도록 영상화 장치(3) 및 영상 포착 제어기(4)를 사용한다. 멀티프로세서 집적 회로(100)는 비디오 압축을 제공하며, 송수신기(16) 및 통신 채널을 통해 다른 위치의 유사한 영상 데이타 처리 시스템으로 압축된 비디오 신호를 전달한다. 이 송수신기(16)는 통신 채널을 통해 원거리 영상 데이타 처리 시스템으로부터 유사한 압축된 비디오 신호를 수신한다. 멀티프로세서 집적 회로(100)는 이렇게 수신된 신호를 분해하며 비디오 디스플레이(8) 상에 대응하는 분해된 비디오 신호를 표시하도록 비디오 등속 호출 메모리(6) 및 비디오 팰릿(7)을 제어한다. 영상 데이타 처리 시스템이 송수신기(16)을 채용하는 것이 유일한 예는 아니다. 또한, 양방향 통신에서는 동일한 형태의 신호일 필요는 없다. 예를 들면, 대화형 케이블 텔레비젼 신호에서 케이블 시스템 헤드 인(head in)은 통신 채널을 통해 영상 데이타 처리 시스템으로 압축된 비디오 신호를 전달한다. 영상 데이타 처리 시스템은 송수신기(16) 및

통신 채널을 통해 케이블 시스템 헤드 인으로 다시 제어 및 데이타 신호를 전달한다.

제1도는 호스트 처리 시스템(1)을 포함하는 시스템에 구성된 멀티프로세서 집적 회로(100)를 나타낸다. 본 기술 분야에 숙련된 자는 멀티프로세서 집적 회로(100)가 유용한 시스템의 유일한 프로세서로서 사용될 수 있다는 것을 본 발명의 다음의 상세한 설명으로부터 알 수 있을 것이다. 상기한 시스템에서 멀티프로세서 집적 회로(100)는 시스템의 모든 기능을 행할 수 있도록 프로그램되어 있다.

특히, 본 발명은 영상 처리용으로서 사용되는 프로세서에 유용하다. 양호한 실시예에 따르면, 본 발명은 멀티프로세서 집적 회로(100)에 구현된다. 이러한 양호한 실시예는 본 발명을 구성하는 복수의 동일한 프로세서를 포함한다. 이들 프로세서 각각은 디지탈 영상/그래픽 프로세서로 불리우게 된다. 이러한 설명은 단지 편의를 위한 것이다. 본 발명을 구성하는 프로세서는 하나의 집적 회로 또는 복수의 집적 회로 상에 개별적으로 제작되는 프로세서일 수도 있다. 하나의 집적 회로 상에 구현되는 경우, 이러한 하나의 집적 회로는 연산 가능하게 디지탈 영상/그래픽 프로세서에 의해 사용되는 판독 전용 메모리 및 등속 호출 메모리를 포함한다.

제2도는 본 발명의 양호한 실시예의 멀티프로세서 집적 회로(100)의 아키택쳐를 나타낸다. 멀티프로세서 집적 회로(100)는, 복수 섹션으로 각각 분할되어 있는 2개의 등속 호출 메모리(10, 20); 크로스바(50); 마스터 프로세서(60); 디지탈 영상/그래픽 프로세서(71, 72, 73, 74); 시스템 메모리에 대한 액세스를 중재하는 전송 제어기(80); 그리고 독립된 제1 및 제2 영상 메모리에 대한 액세스를 제어하는 프레임 제어기(90)를 포함한다. 멀티프로세서 집적 회로(100)는 멀티-매체의 연산에서와 같이 영상 처리 및 그래픽 연산에 유용하게 되는 고도의 병렬 연산을 제공한다.

멀티프로세서 집적 회로(100)는 2개의 등속 호출 메모리를 포함한다. 등속 호출 메모리(10)는 주로 마스터 프로세서(60)에 전용된다. 이것은 2개의 명령 캐시 메모리(11, 12), 2개의 데이타 캐시 메모리(13, 14) 및 파라메타 메모리(15)를 포함한다. 이들 메모리 섹션은 물리적으로 동일하지만 다르게 접속 및 사용될 수 있다. 등속 호출 메모리(20)는 마스터 프로세서(60) 및 디지탈 영상/그래픽 프로세서(71, 72, 73, 74) 각각에 의해 액세스될 수 있다. 각 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)는 5개의 대응하는 메모리 섹션을 갖는다. 이들은 명령 캐시 메모리, 3개의 데이타 메모리 및 하나의 파라메타 메모리를 포함한다. 이와 같이 디지탈 영상/그래픽 프로세서(71)는 대응하는 명령 캐시 메모리(21), 데이타 메모리(22, 23, 24) 및 파라메타 메모리(25)를 갖고; 디지탈 영상/그래픽 프로세서(72)는 대응하는 명령 캐시 메모리(26), 데이타 메모리(27, 28, 29) 그리고 파라메타 메모리(30)를 가지며; 디지탈 영상/그래픽 프로세서(73)는 대응하는 명령 캐시 메모리(31), 데이타 메모리(32, 33, 34) 그리고 파라메타 메모리(35)를 갖고; 그리고 디지탈 영상/그래픽 프로세서(74)는 대응하는 명령 캐시 메모리(36), 데이타 메모리(37, 38, 39) 그리고 파라메타 메모리(40)를 갖는다. 등속 호출 메모리(10)의 섹션과 같이, 이들 메모리 섹션들은 물리적으로는 동일하지만 다르게 접속 및 사용될 수 있다. 메모리(10, 20)의 이들 메모리 섹션 각각은 2K 바이트를 포함하며, 멀티프로세서 집적 회로(100) 내의 총 메모리는 50K 바이트이다.

멀티프로세서 집적 회로(100)는 프로세서와 복수의 독립적인 병렬 데이타 전송을 이용하는 메모리간에 높은 비율의 데이타 전송을 제공하도록 구성된다. 각 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)는 각 사이클마다 동시에 동작될 수 있는 3개의 메모리 포트를 갖는다. 명령 포트(I)는 대응하는 명령 캐시로부터 64 비트의 데이타 워드를 추출한다. 로컬 데이타 포트(L)는 대응하는 명령 캐시로부터 32 비트의 데이타 워드를 판독하거나 또는 32 비트의 데이타 워드로 데이타 메모리 혹은 디지탈 영상/그래픽 프로세서에 대응하는 파라메타 메모리 속에 기록한다. 글로벌 데이타 포트(G)는 대응하는 명령 캐시로부터 32 비트의 데이타 워드를 판독하거나 혹은 데이타 메모리 또는 파라메타 메모리 또는 등속 호출 메모리(20) 중 어느 것으로부터 32 비트의 데이타 워드를 기록한다. 마스터 프로세서(60)는 2개의 메모리 포트를 포함한다. 명령 포트(I)는 명령 캐시(11 및 12) 중 어느 하나로부터 32 비트의 명령 워드를 추출한다. 데이타

포트(C)는 등속 호출 메모리(10)의 데이타 캐시(13 또는 14)와 파라메타 메모리(15), 또는 등속 호출 메모리(20)의 데이타 메모리와 파라메타 메모리 중 어느 곳으로부터 32 비트의 데이타 워드를 판독하거나 그곳에 32 비트의 데이타 워드를 기록할 수 있다. 전송 제어기(80)는 데이타 포트(C)를 통해 등속 호출 메모리(10 또는 20)의 섹션 중 어느 것을 액세스할 수 있다. 이렇게 15개의 병렬 메모리의 액세스는 어떤 하나의 메모리 사이클에서 요청될 수 있다. 등속 호출 메모리(10 및 20)는 많은 병렬 액세스를 지원하도록 25개의 메모리로 분할된다.

크로스바(50)는 마스터 프로세서(60), 디지탈 영상/그래픽 프로세서(71, 72, 73, 74) 및 메모리(10, 20)를 가진 전송 제어기(80)의 접속을 제어한다. 크로스바 (50)는 행 및 열로 배열된 복수의 교차점(51)을 포함한다. 교차점(51)의 각 열은 하나의 메모리 섹션 및 어드레스의 대응 범위에 대응한다. 프로세서는 상기 프로세서에 의해 출력된 어드레스의 최상위 비트를 통해 메모리 섹션 중 하나에 대한 액세스를 요청한다. 상기 프로세서에 의해 출력된 어드레스는 행을 따라 전송된다. 상기 어드레스를 가진 메모리 섹션에 대응하는 교차점(51)은 메모리 섹션에 대한 액세스를 허가하거나 또는 거부함으로써 응답한다. 진행중인 메모리 사이클 동안 상기 메모리 섹션에 대한 액세스를 어떤 다른 프로세서가 요청하지 않으면, 교차점(51)은 행 및 열을 결합시킴으로써 액세스를 허가한다. 이것은 메모리 섹션에 어드레스를 공급한다. 메모리 섹션은 상기 어드레스에 데이타 액세스를 가능하게 함으로써 응답한다. 이 데이타 액세스는 데이타 판독 연산 또는 데이타 기록 연산일 수 있다.

하나 이상의 프로세서가 동일한 메모리 섹션에 대한 액세스를 동시에 요청하면, 크로스바(50)가 요청을 하고 있는 프로세서 중 하나에만 액세스를 허가한다. 크로스바(50)의 각 열에서의 교차점(51)은 교신하여 우선 순위에 따라 액세스를 허가한다. 동일한 등급의 2개의 프로세서가 동시에 액세스를 요청하면, 크로스바(50)가 라운드 로빙 방식에 따라 액세스를 허가하고 가장 낮은 우선 순위를 가진 프로세서에게 최종적으로 액세스를 허가한다. 각각의 허가된 액세스는 요청 서비스를 필요로 하는 한 지속된다. 프로세서는 메모리 사이클마다 그들의 어드레스를 변경할 수 있으므로, 크로스바(50)는 사이클 대 사이클 방식에 따라 프로세서와 메모리 섹션간의 상호 접속을 변경할 수 있다.

마스터 프로세서(60)는 멀티프로세서 집적 회로(100)의 주 제어 기능을 수행한다. 이 마스터 프로세서(60)는 하드웨어 부동점 연산 유닛을 포함하는 32 비트의 감산된 명령 세트 컴퓨터(RISC) 프로세서이다. RISC 아키택쳐에 따르면, 메모리에 대한 모든 액세스는 로드 및 스토어 명령으로 수행되며, 대개 정수 및 논리 연산은 하나의 사이클에서 레지스터를 통해 행해진다. 그러나, 부동점 연산 유닛은 정수 및 논리 유닛에 의해 사용되는 것과 동일한 레지스터 파일을 사용할 때 연산을 행하도록 일반적으로 여러 사이클을 취한다. 레지스터 스코어 보드는 올바른 레지스터 액세스 순서가 유지되는 것을 보장한다. RISC 아키택쳐는 영상 처리에서의 제어 기능을 하는데 적합하다. 부동점 연산 유닛은 영상 처리에서 중요시되는 영상 회전 기능의 신속한 연산을 하게 한다.

마스터 프로세서(60)는 명령 캐시 메모리(11) 또는 명령 캐시 메모리(12)로부터 명령 워드를 추출한다. 이와 유사하게, 마스터 프로세서(60)는 데이타 캐시 (13) 또는 데이타 캐시(14)로부터 데이타를 추출한다. 각 메모리 섹션이 2K 바이트의 메모리를 포함하기 때문에, 4K 바이트의 명령 캐시 및 4K 바이트의 데이타 캐시가 존재하게 된다. 캐시 제어는 마스터 프로세서(60)의 적분 함수이다. 상술한 바와 같이, 마스터 프로세서(60)는 또한 크로스바(50)를 통해 다른 메모리 섹션을 액세스할 수 있다.

4개의 디지탈 영상/그래픽 프로세서(71, 72, 73, 74) 각각은 빠른 병렬 디지탈 신호 프로세서(DSP) 아키택쳐를 갖는다. 제3도는 디지탈 영상/그래픽 프로세서 (72, 73, 74)와 동일한 예의 디지탈 영상/그래픽 프로세서(71)의 개관도이다. 디지탈 영상/그래픽 프로세서(71)는 3개의 분리된 유닛 즉, 데이타 유닛(110), 어드레스 유닛(120), 및 프로그램

흐름 제어 유닛(130)을 사용하는 고도의 병렬 연산을 수행한다. 이들 3개의 유닛은 명령 파이프라인의 다른 명령에 따라 동시에 연산한다. 또한 이들 유닛 각각은 내부 병렬(internal parallelism)을 포함한다.

디지탈 영상/그래픽 프로세서(71, 72, 73, 74)는 다중 명령 다중 데이타 모드(MIMD; multiple instruction multiple data mode)에서 독립 명령 스트림을 실행할 수 있다. MIMD 모드에서, 각 디지탈 영상/그래픽 프로세서는 독립 또는 협조 관계에 있는 그의 대응 명령 캐시로부터 독립 프로그램을 실행한다. 후자의 경우에 크로스바(50)는 공유 메모리와 결합하여 프로세서간 교신을 할 수 있도록 한다. 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)는 또한 동기화 MIMD 모드에서 연산될 수 있다. 동기화 MIMD 모드에서는, 각 디지탈 영상/그래픽 프로세서의 프로그램 제어 흐름 유닛(130)이 모든 동기화 프로세서가 속행될 준비를 하고 있을 때까지 다음 명령을 추출하는 것을 금지한다. 이러한 동기화 MIMD 모드는 디지탈 영상/그래픽 프로세서의 개별 프로그램이 락(lock) 단계에서 밀착 결합된 연산으로 실행되도록 한다.

디지탈 영상/그래픽 프로세서(71, 72, 73, 74)는 단일 명령 다중 데이타 모드(SIMD; single instruction multiple data mode)에서는 다른 데이타에 대하여 동일한 명령을 실행할 수 있다. 이 모드에서 4개의 디지탈 영상/그래픽 프로세서에 대한 신호 명령 스트림은 명령 캐시 메모리(21)로부터 나온다. 디지탈 영상/그래픽 프로세서(71)는 추출 및 브랜칭(branching) 연산을 제어하고, 크로스바(50)는 동일한 명령을 다른 디지탈 영상/그래픽 프로세서(72, 73, 74)에 공급한다. 디지탈 영상/그래픽 프로세서(71)가 모든 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)에 대한 명령 추출을 제어함으로, 디지탈 영상/그래픽 프로세서는 SIMD 모드에서 본래대로 동기화 된다.

전송 제어기(80)는 멀티프로세서 집적 회로(100)용으로 결합된 직접 메모리 액세스(DMA) 머신 및 메모리 인터페이스이다. 전송 제어기(80)는 지능적으로 대기 행렬을 만들어 우선 순위를 설정하고 5개의 프로그래머블 프로세서의 데이타 요청 및 캐시 미스를 서비스한다. 마스터 프로세서(60) 및 디지탈 영상/그래픽 프로세서 (71, 72, 73, 74) 모두는 전송 제어기(80)를 통해 멀티프로세서 집적 회로(100) 외부의 메모리 및 시스템을 액세스한다. 데이타 캐시 또는 명령 캐시 미스는 전송 제어기(80)에 의해 자동적으로 처리된다. 캐시 서비스(S) 포트는 이러한 캐시 미스를 전송 제어기(80)로 전달한다. 캐시 서비스 포트(S)는 프로세서로부터의 정보를 판독하지만 메모리로부터의 정보를 판독하지는 않는다. 마스터 프로세서(60) 및 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)는 링크된 리스트 패킷 요청에 따라 전송 제어기(80)로부터 데이타 전송을 요청할 수 있다. 이들 링크된 리스트 패킷 요청은 발생지와 멀티프로세서 집적 회로(100)내 또는 멀티프로세서 집적 회로(100) 외부에 존재할 수 있는 목적지 메모리 어드레스간에 다차원의 정보 블럭이 전송될 수 있도록 한다. 또한, 전송 제어기(80)는 그들의 데이타를 유지하기 위해 주기적인 재생을 필요로 하는 동적 등속 호출 메모리(DRAM)용 재생 제어기를 포함한다.

프레임 제어기(90)는 멀티프로세서 집적 회로(100)와 외부 영상 포착 및 표시 시스템간의 인터페이스이다. 프레임 제어기(90)는 포착 및 표시 장치를 통한 제어를 제공하고, 이들 장치와 메모리간의 데이타의 전송을 자동적으로 조정한다. 결국, 프레임 제어기(90)는 2개의 독립된 영상 시스템을 통해 동시 제어를 제공한다. 통상적으로 프레임 제어기(90)의 응용은 사용자에 의해 제어되지만, 영상 시스템으로는 영상 포착용 제1 영상 시스템 및 영상 표시용 제2 영상 시스템이 있다. 이들 영상 시스템은 기본적으로 프레임 그래버(frame grabber) 또는 프레임 일시 저장을 위해 사용되는 독립적인 프레임 메모리를 포함한다. 프레임 제어기(90)는 재생 및 이동 레지스터 제어를 통해 비디오 다이나믹 등속 호출 메모리(VRAM)를 제어하도록 연산한다.

멀티프로세서 집적 회로(100)는 대규모 영상 처리용으로서 설계된다. 마스터 프로세서(60)는 내장 제어(embedded

control)를 제공하며, 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)의 활동을 지휘하고, 프로세서들이 만들어내는 결과를 해석한다. 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)는 픽셀 분석 및 조정에 최적이다. 픽셀이 데이타에서는 높고 정보에서는 낮다고 하면, 통상적인 응용에 있어서 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)는 픽셀을 잘 조사하여 새로운 데이타를 정보로 한다. 이어서 이 정보는 디지탈 영상/그래픽 프로세서(71, 72, 73, 74) 또는 마스터 프로세서(60)에 의해 분석될 수 있다. 크로스바(50)는 프로세서간의 교신을 중재한다. 크로스바(50)를 통해서 멀티프로세서 집적 회로(100)를 공유 메모리 시스템과 같이 구성할 수 있다. 메시지 전달이 상기한 아키택쳐에서 일차적인 교신의 형태로 될 필요는 없다. 그러나, 메시지는 공유 메모리를 통해 전달될 수 있다. 각 디지탈 영상/그래픽 프로세서, 크로스바(50)의 대응 섹션 그리고 대응하는 메모리(20)의 섹션은 동일한 폭을 갖는다. 이것으로 인해 디지탈 영상/그래픽 프로세서의 추가 또는 제거와, 대응하는 메모리를 모듈화함과 동시에 동일한 핀아웃(pinout)을 유지함으로써 아키택쳐의 융통성을 이룰 수 있다.

양호한 실시예에서 멀티프로세서 집적 회로(100)의 모든 부분은 하나의 집적 회로상에 배치될 수 있다. 양호한 실시예에서, 멀티프로세서 집적 회로(100)는 0.6 μm의 최소 배선폭(feature size)을 사용하는 상보 금속 산화물 반도체(CMOS)로 형성된다. 멀티프로세서 집적 회로(100)는 256 핀을 가진 핀 그리드 어레이 패키지 내에 구성된다. 입력 및 출력은 트랜지스터-트랜지스터 논리(TTL)의 논리 전압에 적합하다. 멀티프로세서 집적 회로(100)는 약 3백만개의 트랜지스터를 포함하고 있으며 50 MHz의 클럭 속도를 사용한다.

제3도는 디지탈 영상/그래픽 프로세서(72, 73, 74)와 실질적으로 동일한 예의 디지탈 영상/그래픽 프로세서(71)의 개관도이다. 디지탈 영상/그래픽 프로세서(71)는, 데이타 유닛(110), 어드레스 유닛(120), 그리고 프로그램 흐름 제어 유닛(130)을 포함한다. 데이타 유닛(110)은 논리 또는 산술 데이타 연산을 실행한다. 데이타 유닛(110)은 8개의 데이타 레지스터(D7-D0), 상태 레지스터(210) 및 다중 플래그 레지스터(211)를 포함한다. 어드레스 유닛(120)은 로컬 데이타 포트 및 글로벌 데이타 포트용 로드/스토어 어드레스의 발생을 제어한다. 이하 더욱 상세히 설명하는 바와 같이, 어드레스 유닛(120)은 2개의 실질적으로 동일한 어드레싱 유닛 즉, 로컬 어드레싱용 및 글로벌 어드레싱용의 유닛을 포함한다. 이들 어드레싱 유닛 각각은 관련 어드레스 모드에서 절대 어드레싱을 인에이블링하는 모두 " 0" 의 판독 전용 레지스터, 스택포인터, 5개의 어드레스 레지스터 및 3개의 인덱스 레지스터를 포함한다. 어드레싱 유닛은 양 어드레스 유닛으로부터 조합 어드레스를 형성할 때 사용되는 글로벌 비트 멀티플렉스 제어 레지스터를 공유한다. 프로그램 흐름 제어 유닛(130)은 명령 포트를 통해 명령 추출을 위한 어드레스의 발생을 포함하는 디지탈 영상/그래픽 프로세서(71)의 프로그램 흐름을 제어한다. 프로그램 흐름 제어 유닛(130)은, 프로그램 카운터(PC), 어드레스 파이프라인 스테이지 (stage)에서 사용중인 명령 어드레스를 유지하는 명령 포인터-어드레스 스테이지 (IRA), 실행 파이프라인 스테이지에서 사용중인 명령 어드레스를 유지하는 명령 포인터-실행 스테이지(IRA), 서브루틴으로부터 복귀하는 어드레스를 유지하는 명령 포인터-복귀 서브루틴(IPRS), 제로 오버 헤드 루프(zero overhead loop)를 제어하는 레지스터 세트, 대응하는 명령 캐시 메모리에서 명령 워드의 4개의 블럭의 최상위 비트를 유지하는 4개의 캐시 태그 레지스터(TAG3-0)를 포함한다.

디지탈 영상/그래픽 프로세서(71)는 제4도에 도시한 바와 같이 3 스테이지 파이프라인에 따라 연산한다. 데이타 유닛(110), 어드레스 유닛(120) 그리고 프로그램 흐름 제어 유닛(130)은 명령 파이프라인에서 다른 명령에 따라 동시에 연산한다. 3개의 스테이지는 시간순으로 추출, 어드레스 및 실행된다. 이러한 방식으로, 임의의 시간에서 디지탈 영상/그래픽 프로세서(71)는 3개의 명령의 다른 함수로 연산될 것이다. 프레이즈(phrase) 파이프라인 스테이지는 클럭 사이클을 조회하는 대신에 파이프라인이 진행될 때 특정한 사건이 발생되었음을 나타내도록 사용되며 스톨(stall) 조건 동안에는 사용되지 않는다.

프로그램 흐름 제어 유닛(130)은 추출 파이프라인 스테이지 동안 발생되는 모든 연산을 실행한다. 프로그램 흐름 제어

유닛(130)은 프로그램 카운터, 루프논리, 인터럽트 논리 및 파이프라인 제어 논리를 포함한다. 추출 파이프라인 스테이지 동안, 다음 명령 워드가 메모리로부터 추출된다. 프로그램 카운터에 포함되는 어드레스는 다음 명령 워드가 명령 캐시 메모리(21)에 저장되어 있는지를 판정하기 위해 캐시 태그 레지스터와 비교된다. 프로그램 흐름 제어 유닛(130)은 명령 캐시 메모리로부터 다음 명령 워드를 추출하도록 프로그램 카운터 내의 어드레스를 명령 포트 어드레스 버스(131)에 제공한다. 크로스바(50)는 상기 어드레스를 대응하는 명령 캐시로 전달하는데, 여기서 명령 캐시 메모리(21)는 상기 명령 워드를 명령 버스(132) 상에 복귀시킨다. 그밖에, 캐시 미스가 발생하면 전송 제어기(80)가 다음 명령 워드를 얻기 위해 외부 메모리를 액세스한다. 프로그램 카운터는 경신된다. 다음 명령 워드가 다음 순차 어드레스에 있으면, 프로그램 제어 흐름 유닛(130)이 프로그램 카운터에 증분을 기록한다. 그밖에, 프로그램 제어 흐름 유닛(130)이 루프 논리 또는 소프트웨어 브랜치에 따라 다음 명령 워드의 어드레스를 로드한다. 동기화 MIMD 모드가 활성화되면, 통신 레지스터에 동기 비트(sync bit)로 표시한 바와 같이 모든 특정한 디지탈 영상/그래픽 프로세서가 동기화 될 때까지 명령 추출이 대기 상태로 된다.

어드레스 유닛(120)이 어드레스 파이프라인 스테이지의 모든 어드레스 연산을 수행한다. 어드레스 유닛(120)은 2개의 독립된 어드레스 유닛, 즉 글로벌 포트용 및 로컬 포트용의 유닛을 포함한다. 하나 또는 2개의 메모리에 대한 명령 호출이 액세스되면 어드레스 유닛(120)이 어드레스 파이프라인 스테이지 동안 어드레스를 발생시킨다. 이 어드레스는 경쟁 검출/우선 순위화를 위한 각각의 글로벌 포트 어드레스 버스(121) 및 로컬 포트 어드레스 버스(122)를 통해 크로스바(50)에 공급된다. 경쟁이 없으면, 액세스된 메모리가 요청된 액세스를 하려고 준비하여 다음 실행 파이프라인 스테이지 동안 메모리 액세스가 발생된다.

데이타 유닛(110)은 실행 파이프라인 스테이지 동안 모든 논리 및 산술 연산을 수행한다. 메모리로의 모든 논리 및 산술 연산 그리고 모든 데이타 전송은 실행 파이프라인 스테이지 동안 발생된다. 글로벌 데이타 포트 및 로컬 데이타 포트는 어드레스 파이프라인 스테이지 동안, 실행 파이프라인 스테이지 동안 시작되는 임의의 메모리 액세스를 완료한다. 글로벌 데이타 포트 및 로컬 데이타 포트는 메모리 스토어에 필요한 모든 데이타의 배열 및 메모리 로드에 필요한 임의의 데이타 추출 및 부호 확장을 수행한다. 프로그램 카운터가 실행 파이프라인 스테이지의 임의의 연산 동안 데이타 목적지로서 특정되면 임의의 브랜치가 실시되기 전에 두 명령의 지연이 나타난다. 파이프라인된 연산은 이러한 지연을 필요로 하는데, 그 이유는 상기한 브랜치 명령 이후의 다음 두 명령이 이미 추출되었기 때문이다. RISC 프로세서의 실시에 따르면, 다른 유용한 명령은 2개의 지연 슬롯 위에 위치 설정될 수 있다.

디지탈 영상/그래픽 프로세서(71)는 3개의 내부 32 비트 데이타 버스를 포함한다. 이들은 로컬 포트 데이타 버스(LBUS, 103), 글로벌 포트 발생지 데이타 버스(GSRC, 105) 및 글로벌 포트 목적지 데이타 버스(GDST, 107)이다. 이들 3개의 버스는 데이타 유닛(110), 어드레스 유닛(120) 및 프로그램 흐름 제어 유닛(130)을 상호 접속시킨다. 또한, 이들 3개의 버스는 로컬 데이타 포트(141) 및 글로벌 데이타 포트(145)를 가진 데이타 포트 유닛에 접속된다. 데이타 포트 유닛(140)은 메모리 액세스를 제공하는 크로스바(50)에 결합된다.

로컬 데이타 포트(141)는 메모리로 데이타를 스토어하기 위한 버퍼(142)를 갖는다. 멀티플렉서/버퍼 회로(143)는 크로스바(50)를 통해 메모리로부터의 로컬 포트 데이타 버스(144)로부터, 로컬 포트 어드레스 버스(122)로부터 또는 글로벌 포트 데이타 버스(148)로부터 LBUS(103) 상에 데이타를 로드한다. 이와 같이 로컬 포트 데이타 버스(LBUS, 103)는 레지스터에 소스(스토어)된 또는 메모리에 소스(로드)된 32 비트 데이타를 실어 나른다. 따라서, 어드레스 유닛(120)에서의 산술 결과가 로컬 포트 어드레스 버스(122)를 통해 공급될 수 있는데, 즉 데이타 유닛 (110)의 산술 연산을 추가하기 위해 로컬 포트 데이타 버스(LBUS, 103)를 통해 멀티플렉서 버퍼(143)에 공급된다. 이하 이에 대한 설명을 상세히 설명한다. 버퍼 (142) 및 멀티플렉서 버퍼(143)는 데이타의 배열 및 추출을 실행한다. 로컬 포트 데이타 버스(LBUS, 103)는 데이타 유닛(110)의 데이타 레지스터에 접속된다. 또한, 로컬 버스 일시 유지 레지스터(LTD,

104)는 로컬 포트 데이타 버스(LBUS, 103)에 접속된다.

글로벌 포트 발생지 데이타 버스(GSRC, 105) 및 글로벌 포트 목적지 데이타 버스(GDST, 107)는 글로벌 데이타 전송을 중재한다. 이들 글로벌 데이타 전송은 메모리 액세스, 즉 레지스터 대 레지스터 전송 또는 프로세서들간의 커맨드 워드 전송일 수 있다. 글로벌 포트 발생지 데이타 버스(GSRC, 105)는 글로벌 포트 데이타 전송이 32 비트 발생지 정보를 운반한다. 데이타 발생지는 디지탈 영상/그래픽 프로세서(71)의 레지스터 중 어느 것 또는 임의의 데이타 또는 디지탈 영상/그래픽 프로세서(71, 72, 73, 74) 중 어느 것에 대응하는 파라메타 메모리일 수 있다. 상기 데이타는 글로벌 데이타 포트(145)를 통해 메모리에 저장된다. 멀티플렉서 버퍼(146)는 로컬 포트 데이타 버스(LBUS, 103) 또는 글로벌 포트 발생지 데이타 버스(GSRC, 105)로부터 라인을 선택하고, 데이타 배열을 수행한다. 멀티플렉서 버퍼(146)는 크로스바(50)를 통해 메모리에 적용하기 위해 글로벌 포트 데이타 버스(148)상에 상기 데이타를 기록한다. 글로벌 포트 발생지 데이타 버스(GSRC, 105)는 또한 데이타를 데이타 유닛(110)에 공급하여, 글로벌 포트 발생지 데이타 버스(GSRC, 105)의 데이타가 산술 논리 유닛 발생지 중 하나로서 사용될 수 있도록 한다. 이러한 후자의 접속 관계로 인해 디지탈 영상/그래픽 프로세서(71)의 임의의 레지스터가 산술 논리 유닛의 연산에 대한 발생지가 된다.

글로벌 포트 목적지 데이타 버스(GDST, 107)는 글로벌 버스 데이타 전송의 32 비트의 목적지 데이타를 운반한다. 이 목적지는 디지탈 영상/그래픽 프로세서(71)의 임의의 레지스터가 된다. 글로벌 데이타 포트(145) 내의 버퍼(147)는 글로벌 포트 목적지 데이타 버스(GDST, 107)의 데이타를 소스한다. 버퍼(147)는 임의의 필요한 데이타 추출 및 부호 확장 연산을 수행한다. 이 버퍼(147)는 데이타 발생지가 메모리인 경우에 연산함으로 로드가 수행된다. 산술 논리 유닛의 결과는 글로벌 포트 목적지 데이타 버스(GDST, 107)에 대한 다른 데이타 발생지로서 작용한다. 이로 인해 디지탈 영상 그래픽 프로세서(71)의 임의의 레지스터가 산술 논리 유닛의 연산의 목적지가 되게 한다. 또한 글로벌 버스 일시 유지 레지스터(GTD, 108)는 또한 글로벌 포트 목적지 데이타 버스(GDST, 107)에 접속된다.

멀티플렉서 버퍼(143, 146)를 포함하는 회로망을 레지스터대 레지스터 전송을 제공하기 위해 글로벌 포트 발생지 데이타 버스(GSRC, 105)와 글로벌 포트 목적지 데이타 버스(GDST, 107)간에 접속된다. 이로 인해 글로벌 포트 목적지 데이타 버스(GDST, 107)를 통해 디지탈 영상/그래픽 프로세서(71)의 임의의 레지스터로 기록되어지는 글로벌 포트 발생지 데이타 버스(GSRC, 105) 상의 디지탈 영상/그래픽 프로세서(71)의 임의의 레지스터로부터의 판독을 행할 수 있다.

글로벌 포트 목적지 데이타 버스(GDST, 107)를 통해 메모리로부터 디지탈 영상/그래픽 프로세서(71)의 임의의 레지스터의 로드를 실행하고, 동시에 글로벌 포트 발생지 데이타 버스(GSRC, 105)를 통해 임의의 레지스터로부터 데이타 유닛(110)의 산술 논리 유닛을 소스하는 것이 보다 가능하다. 유사하게, 글로벌 포트 발생지 데이타 버스(GSRC, 105)를 통해 디지탈 영상/그래픽 프로세서(71)의 임의의 레지스터내의 데이타를 메모리로 스토어하고, 동시에 글로벌 포트 목적지 데이타 버스(GDST, 107)를 통해 디지탈 영상/그래픽 프로세서(71)의 임의의 레지스터에 산술 논리 유닛의 연산 결과를 세이브하는 것이 보다 가능하다. 이들 데이타의 유용성에 대해서는 후술한다.

프로그램 흐름 제어 유닛(130)은 명령 버스(132)를 통해 명령 캐시 메모리(21)로부터 추출된 명령 워드를 수신한다. 이렇게 추출된 명령 워드는 2개의 64비트의 명령 레지스터의 지정된 명령 레지스터 - 어드레스 스테이지(IRA) 및 명령 레지스터 - 실행 스테이지(IRE)에 유리하게 저장된다. 명령 레지스터(IRA, IRE) 각각은 그들의 내용이 디코드되어 분배되게 한다. 디지탈 영상/그래픽 프로세서(71)는 디코드되었거나 부분적으로 디코드된 명령의 내용을 데이타 유닛(110) 및 어드레스 유닛(120)으로 운반하는 오피코드(opcode) 버스(133)를 포함한다. 후술하는 바와 같이, 명령 워드

는 32 비트, 15 비트 또는 3 비트의 즉시 필드를 포함할 수 있다. 프로그램 흐름 제어 유닛(130)은 상기한 즉시 필드(immediate field)를 그의 목적지에 제공하게 위해 글로벌 포트 발생지 데이타 버스(GSRC, 105)로 지향한다.

디지탈 영상/그래픽 프로세서(71)는 3개의 어드레스 버스(121, 122, 131)를 포함한다. 어드레스 유닛(120)은 글로벌 포트 어드레스 버스(121) 및 로컬 포트 어드레스 버스(122) 상에 어드레스를 발생시킨다. 후술하는 바와 같이, 어드레스 유닛(120)은 글로벌 포트 어드레스 버스(121) 및 로컬 포트 어드레스 버스(122) 상에 각각 어드레스를 제공하는 개별적인 글로벌 및 로컬 어드레스 유닛을 포함한다. 로컬 어드레스 유닛(620)은 그 디지탈 영상/그래픽 프로세서에 대응하는 데이타 메모리와 다른 메모리를 액세스할 수도 있다. 상기와 같은 경우에 있어서, 로컬 어드레스 유닛의 액세스는 글로벌 포트 어드레스 버스(121)를 통해 이루어진다. 프로그램 흐름 제어 유닛(130)은 프로그램 카운터 및 캐시 제어 논리로부터의 어드레스 비트의 조합으로부터 명령 포트 어드레스 버스(131) 상의 명령 어드레스를 소스한다. 이들 어드레스 버스(121, 122, 131) 각각은 어드레스, 바이트 스트로브(byte strobe) 및 판독/기록 정보를 운반한다.

제5도는 데이타 유닛(110)의 상세도이다. 제5도에는 데이타 유닛(110)의 모든 접속 상태가 예시되어 있지 않다. 특히, 여러 가지 제어 라인 등은 도면을 명료하기 위해 생략되어 있다. 따라서, 제5도는 데이타 유닛(110)의 완전한 연산에 대한 이해를 위한 다음의 설명을 통해 이해될 수 있을 것이다. 데이타 유닛(110)은 병렬 연산하는 복수의 부분을 포함한다. 데이타 유닛(110)은 D7-D0로 지정된 8개의 32 비트 데이타 레지스터(200)를 포함한다. 데이타 레지스터(D0)는 범용 레지스터로서 사용될 수 있지만 이와 아울러 특정한 명령과 함께 사용될 때는 특수한 기능을 갖는다. 데이타 레지스터(200)는 데이타 유닛 버스(201 206) 그리고 로컬 포트 데이타 버스(LBUS, 103), 글로벌 포트 발생지 데이타 버스(GSRC, 105) 및 글로벌 포트 목적지 데이타 버스(GDST, 107)에 접속된 다중 판독 및 기록 포트를 포함한다. 데이타 레지스터(200)는 또한 후술하는 회전 레지스터로서 설명되는 방식의 " 사이드웨이(sideways)" 식으로 판독될 수 있다. 데이타 유닛(110)은 또한 특정한 명령에 대한 산술 논리 유닛의 결과 상태를 스토어하는 상태 레지스터(210) 및 다중 플래그 레지스터(211)를 포함한다. 데이타 유닛(110)은 그의 주요한 연산을 행하기 위한 구성 요소로서 하드웨어 승산기(220) 및 3개의 입력 산술 논리 유닛(230)을 포함한다. 마지막으로, 데이타 유닛(110)은, 승산기 제1 입력 버스(201), 승산기 제2 입력 버스(202), 승산기 목적지 버스(203), 산술 논리 유닛 목적지 버스(204), 산술 논리 유닛 제1 입력 버스(205), 산술 논리 유닛 제2 입력 버스(206); 버퍼(104, 106, 108, 236); 멀티플렉서 Rmux221, Imux222, MSmux225, Bmux227, Amux232, Smux231, Cmux233 및 Mmux234; 적(積)좌측 시프터(224), 가산기(226), 원통형 회전기(235), LMO/RMO/LMBC/RMBC 회로(237), 확장 회로(238), 마스크 발생기(239), 입력 A 버스(241), 입력 B 버스(242), 입력 C 버스(243), 회전 버스(244), 함수 신호 발생기(245), 비트 0 캐리-인 발생기(246) 및 명령 디코드 논리(250)을 포함하는데, 이에 대한 설명은 후술한다.

데이타 유닛(110)의 다음의 설명 및 각 디지탈 영상/그래픽 프로세서(71, 72, 73, 74)의 사용에 대한 다음 설명을 용이하게 하기 위해 여러개의 부호를 사용한다. 이들 부호의 대부분은 설명을 필요로 하지 않는 표준 수학적 연산 부호이다. 이들 중 몇몇은 본 기술 분야에 숙련된 자에 친숙한 논리적 연산이지만 친숙하지 않은 부호들도 있다. 끝으로, 몇몇 부호는 본 발명에 독특한 연산을 나타낸다. 표 1은 이들 몇몇의 부호와 이들에 대응하는 연산에 대한 리스트이다.

부호	연산
~	비트 와이즈 NOT
&	비트 와이즈 AND
|	비트 와이즈 OR
^	비트 와이즈 배타적 OR
@	다중 플래그 레지스터 확장
%	마스크 발생
%!	수정된 마스크 발생
\\	좌측 회전
<<	좌측 이동
>>u	우측 제로 확장 이동
>>s	우측 부호 확장 이동
>>	우측 부호 확장 이동 디폴트의 경우
||	병렬 연산
*(A ± X)	어드레스 베이스 레지스터 A의 메모리 내용 ± 인덱스 레지스터 X 또는 오프셋 X
&*(A ± X)	어드레스 유닛 산술 어드레스 베이스 레지스터 A ± 인덱스 레지스터 X 또는 오프셋 X
*(A ± [X])	어드레스 베이스 레지스터 A의 메모리 내용 ± 스케일된 인덱스 레지스터 X 또는 오프셋 X

[표 1]

표 1에 제시된 연산들의 함축된 의미는 즉시 알 수는 없지만, 이들에 대해서는 후술한다.

제6도는 상태 레지스터(210)에 대한 필드 정의를 나타낸다. 상태 레지스터 (210)는 글로벌 포트 발생지 데이타 버스(GSRC, 105)를 통해 판독되어 글로벌 포트 목적지 데이타 버스(GDST, 107)를 통해 기록될 수 있다. 또한, 상태 레지스터(210)는 데이타 레지스터(200) 중 특정한 하나로부터 기록 또는 로드될 수 있다. 상태 레지스터(210)는 데이타 유닛(110) 내에서의 연산의 제어에 사용된다.

상태 레지스터(210)는 4개의 산술 논리 유닛의 결과의 상태 비트 " N" , " c" , " V" 및 " Z" 을 저장한다. 이들을 개별적으로 이하에 설명하고자 하는 바, 그들의 설정 연산을 집약하면 다음과 같다. 표 1에 제시되어 있는 명령 형태를 이하 상세히 설명한다. 32 비트의 즉시 필드를 포함하는 명령 워드에 있어서, 조건부 코드 필드가 " 비조건부" 이면 모든 4개의 상태 비트가 산술 논리 유닛(230)의 결과에 따라 설정된다. 조건부 코드 필드가 " 비조건부" 외의 조건을 특정하면, 조건이 사실이든 사실이 아니든 간에 어떤 상태 비트도 설정되지 않는다. 32 비트의 즉시 필드 연산을 포함하지 않고, 조건부 연산 필드를 포함하지 않는 명령 워드에 있어서는, 모든 상태 비트가 산술 논리 유닛(230)의 결과에 따라 설정된다. 조건부 연산을 허가하는 32 비트의 즉시 필드를 포함하지 않는 명령 워드에 있어서, 조건부 필드가 " 비조건부" 이거나 또는 " 비조건부" 가 아니고 조건이 사실인 경우, 명령 워드 비트 28-25는 상태 비트가 보호되지 않으면 안됨을 나타낸다. 모든 비보호 비트는 산술 논리 유닛(230)의 결과에 따라 설정된다. 조건부 연산을 하게 하는 32 비트의 즉시 필드를 포함하지 않는 명령 워드에 있어서는, 조건부 필드가 " 비조건부" 가 아니고 조건이 거짓이면, 어떤 상태 비트도 설정되지 않는다. 상태 설정 연산에서는 부울 연산 및 산술 연산에 대한 차이는 없다. 이하 상세히 설명하는 바와 같이, 이러한 연산에 의해, 브랜치를 정상적으로 필요로 하는 연산을 행할 수 있는 조건부 명령 및 발생지 선택을 할 수 있다.

상태 레지스터(210)의 산술 논리 유닛의 결과 비트는 다음과 같다. " N" 비트(비트 31)는 부정적인 결과 표시를 저장한다. " N" 비트는 산술 논리 유닛(230)의 최종 연산의 결과가 부정일 때 " 1" 로 설정된다. 이 비트는 상기 결과의 비트 31로 로드된다. 이하 설명되는 다중 산술 논리 유닛 연산에 있어서, " N" 비트는 산술 논리 유닛(230)의 복수의 섹션 중 제로 비교부의 AND로 설정된다. LMO/RMO/LMBC/RMBC회로(237)에 의해 실행되는 비트 검출 연산에 있어서, " N" 비트는 산술 논리 유닛(230)의 복수의 섹션 중 제로 비교부의 AND로 설정된다. 소프트웨어에 상기 비트의 기록은 정상적인 산술 논리 유닛의 결과의 기록 논리 보다 우선한다.

" C" 비트(비트 30)는 캐리 결과의 표시를 저장한다. " C" 비트는 산술 논리 유닛(230)의 최종 연산의 결과가 산술 논리 유닛의 비트 31로부터의 캐리-아웃에 기인한 경우 " 1" 로 설정된다. 다중 산술 및 비트 검출 동안, " C" 비트는 산술 논리 유닛(230)의 복수의 섹션의 캐리-아웃의 R로 설정된다. 이와 같이, " C" 비트는 섹션의 적어도 하나가 캐리-아웃이면 " 1" 로 설정된다. 소프트웨어에 상기 비트의 기록은 정상적인 산술 논리 유닛의 결과의 기록 논리 보다 우선한다.

" V" 비트(비트 29)는 오버플로 결과의 표시를 저장한다. " V" 비트는 산술 논리 유닛(230)의 최종 연산의 결과가 오버플로 조건에 의해 야기된 것이면 " 1" 로 설정된다. 이 비트는 산술 논리 유닛(230)의 비트 31의 캐리-인 및 캐리-아웃의 배타적 OR로 로드된다. 다중 산술 논리 유닛 연산 동안 " V" 비트는 산술 논리 유닛 (230)의 복수의 섹션의 캐리-아웃의 AND이다. 극좌의 1 및 극우의 1 비트 검출에 있어서, " V" 비트는 입력 워드에 " 1" 이 전혀 없으면 " 1" 로 설정되고, 그 외는 " V" 비트는 " 0" 로 설정된다. 극좌 비트 수정 및 극우 비트 수정의 비트 검출에 있어서, " V" 비트는 모든 입력 비트가 동일하면 " 1" 로 설정되고 그외는 " V" 비트는 " 0" 로 설정된다. 소프트웨어에서의 상기 비트에 대한 기록은 정상적인 산술 논리 유닛 결과의 기록 논리보다 우선한다.

" Z" 비트(비트 28)는 " 0" 결과의 표시를 저장한다. " Z" 비트는 산술 논리 유닛(230)의 최종 연산의 결과가 " 0" 결과를 발생시키면 " 1" 로 설정된다. 상기 " Z" 비트는 산술 연산 및 논리 연산용으로 제어된다. 다중 산술 및 비트 검출 연산에 있어서, 'Z" 비트는 산술 논리 유닛(230)의 복수의 섹션 중 제로 비교부의 OR 로 설정된다. 소프트웨어에서의 상기 비트에 대한 기록은 정상적인 산술 논리 유닛 결과의 기록 논리 회로망 보다 우선한다.

" R" 비트(비트 6)는 확장 회로(238)에 의해 사용되는 비트, 및 다중 플래그 레지스터(211)의 부분을 확장시키기 위해 확장 회로(238)를 사용하는 명령 중의 다중 플래그 레지스터(211)의 회전을 제어한다. " R" 비트가 " 1" 이면 확장 회로(238)를 통해 다중 플래그 레지스터(211)의 확장에 사용되는 비트는 최상위 비트이다. 산술 논리 유닛 기능 수정자가 다중 플래그 레지스터 회전을 특정하지 않는 다중 플래그 레지스터(211)의 확장을 포함하는 연산에 대하여, 다중 플래그 레지스터(211)는 " M사이즈" 필드에 따라 " 사후-회전 좌측(post-rotated left)" 이 된다. 산술 논리 유닛 기능 수정자가 다중 플래그 레지스터 회전을 특정하면, 다중 플래그 레지스터(211)는 " A사이즈" 필드에 따라 회전된다. " R" 비트가 " 0" 이면, 확장 회로(238)는 다중 플래그 레지스터(211)의 최하위 비트를 사용한다. " M사이즈" 필드에 따르면 어떤 회전도 발생되지 않는다. 그러나, 산술 논리 유닛 함수 수정자는 " A사이즈" 필드에 의해 회전을 특정한다.

" M사이즈" 필드(비트 5-3)는 다중 플래그 레지스터(211)로부터 산술 논리 유닛(230)의 C-포트로 마스크 데이타를 공급하는 특정 명령 분류에 사용되는 데이타 크기를 나타낸다. " M사이즈" 필드는 마스크 정보를 발생시키는데 얼마나 많은 다중 플래그 레지스터(211)의 비트가 사용되는지를 결정한다. 명령이 " A사이즈" 필드에 대응하는 회전을 특정하지 않고, " R" 비트가 " 1" 이면 다중 플래그 레지스터(211)는 자동적으로 " M사이즈" 필드에 의해 설정된 양만큼 " 사후-회전 좌측(post-rotuted left)" 된다.

이들 비트에 대한 코딩이 표 2에 도시되어 있다.

M사이즈 필드 5 4 3	데이타 사이즈 비트	다중 플래그 레지스터			
		회전량	사용된 비트수	비트 R = 1	R = 0 사용
0 0 0	0	64	64	-	-
0 0 1	1	32	32	31-0	31-0
0 1 0	2	16	16	31-16	15-0
0 1 1	4	8	8	31-24	7-0
1 0 0	8	4	4	31-28	3-0
1 0 1	16	2	2	31-30	1-0
1 1 0	32	1	1	31	0
1 1 1	64	0	0	-	-

[표 2]

상술한 바와 같이, 양호한 실시예는 각각 8, 16 및 32 비트의 데이타 크기에 대응하는 " 100" , " 101" 및 " 110" 의 " M사이즈" 필드를 지원한다. " 001" 의 " M사이즈" 필드에 대한 회전은 데이타 출력에서 어떤 변화가 없음을 나타낸다. " 001" , " 010" 및 " 011" 의 " M사이즈" 필드는 의미는 없으나, 다중 플래그 레지스터(211)를 64 비트로 확장시키는 데 사용될 수 있다.

" A사이즈" 필드(비트 2-0)는 산술 논리 유닛(230)에 의해 행해지는 다중 연산에 대한 데이타 사이즈를 나타낸다. 산술 논리 유닛(230)은 32의 병렬 비트를 포함한다. 특정한 명령 동안, 산술 논리 유닛(230)은 다중 독립 섹션으로 분할된다. 이것을 다중 산술 논리 유닛 연산이라 한다. 이렇게 산술 논리 유닛(230)이 분할됨으로써 32 비트 데이타 워드로 묶여지는 32 비트 이하의 픽셀에 대한 병렬 연산을 할 수 있다. 양호한 실시예에서, 산술 논리 유닛(230)은, 하나의 32 비트 연산; 16 비트 연산의 2개의 섹션; 그리고 8 비트 연산의 4개의 섹션을 지원한다. 이들 선택을 워드, 반 워드 및 바이트 연산이라 한다.

" A사이즈" 필드는, 산술 논리 유닛(230)의 다중 섹션의 수; 산술 논리 유닛(230)의 섹션 수와 동일한, 산술 논리 유닛의 연산 동안 설정되는 다중 플래그 레지스터 비트(211)의 비트수; 그리고 다중 산술 논리 유닛의 연산 동안 출력된 후에 다중 플래그 레지스터가 " 사후-회전 좌측(post-rotate left)" 되는 비트수를 나타낸다. " A사이즈" 필드에 의해 특정되는 회전량은 산술 논리 유닛 함수 수정자가 회전함에 따라 다중 산술을 표시할 때 " M사이즈" 필드 및 " R" 비트에 의해 특정되는 회전량 보다 우위를 점한다. 제3도에 이들 비트에 대한 코딩이 도시되어 있다. 본 발명의 양호한 실시예가 하나의 32 비트 섹션, 2개의 16 비트 섹션 그리고 4개의 8 비트 섹션의 다중 산술을 제공하지만, " A사이즈" 필드의 코딩은 4 비트씩의 8개의 섹션, 2 비트씩의 16개 섹션 그리고 1 비트씩의 32개 섹션의 세부적인 다중 산술을 제공할 수 있다. 산술 논리 유닛(230)의 각각의 추가의 섹션 분할이 실시 가능하다. 또한, " A사이즈" 필드의 코딩은 다중 플래그 레지스터(211)를 64 비트의 확장을 가능케 하는 64 비트 데이타 사이즈의 세부 사항을 지원한다.

A사이즈 필드 2 1 0	데이타 사이즈 비트	다중 플래그 레지스터		
		회전량	사용된 비트수	설정된 비트
0 0 0	0	64	64	-
0 0 1	1	32	32	31-0
0 1 0	2	16	16	15-0
0 1 1	4	8	8	7-0
1 0 0	8	4	4	3-0
1 0 1	16	2	2	1-0
1 1 0	32	1	1	0
1 1 1	64	0	0	-

[표 3]

상태 레지스터(210)의 " M사이즈" 및 " A사이즈" 필드는 다른 연산을 제어한다. 산술 논리 유닛(230)의 C - 포트에 가해지는 마스크를 발생시키기 위한 발생지로서 다중 플래그 레지스터(211)를 사용할 때, " M사이즈" 필드는 사용되는 비트수 및 회전량을 제어한다. 이 경우에 " R 비트는 최상위 비트 또는 최하위 비트가 사용되는지를 결정한다. 산술 논리 유닛(230)의 섹션에 대응하는 상태 비트에 대한 목적지로서 다중 플래그 레지스터(211)를 사용하면, " A사이즈" 필드는 로드되는 비트의 수 및 일치성 그리고 임의 회전량을 제어한다. " A사이즈" 필드의 특정 회전에 따른 다중 산술 논리 유닛 연산이 다중 플래그 레지스터로부터 유도된 C - 포트로마스크 데이타를 공급하는 명령으로 특정되면, " A사이즈" 필드의 회전량은 " R" 비트 및 " M사이즈" 필드의 조합의 회전량보다 우위를 점한다.

다중 플래그 레지스터(211)는 특정한 명령을 위해 산술 논리 유닛(230)의 C - 포트로 마스크 정보를 제공하는 32 비트 레지스터이다. 글로벌 포트 목적지 데이타 버스(GDST, 107)는 다중 플래그 레지스터(211)로 기록될 수 있다. 글로벌 포트 발생지 버스(GSRC)는 다중 플래그 레지스터(211)로부터 데이타를 판독할 수 있다. 아울러 다중 산술 논리 유닛 연산은 다중 플래그 레지스터(211)로 기록될 수 있다. 이와 같은 경우에, 다중 플래그 레지스터(211)는 산술 논리 유닛(230)의 독립된 섹션의 캐리 또는 제로 상태 정보를 기록한다. 실행되는 명령은 캐리 또는 제로가 저장되는지를 제어한다.

상태 레지스터(210)의 " M사이즈" 필드는 다중 플래그 레지스터(211)에서 사용되는 최하위 비트수를 제어한다. 이러한 수는 상기 표 2에 주어져 있다. 상태 레지스터(210)의 " R" 비트는 이들 비트를 공급하기에 앞서 다중 플래그 레지스터(211)가 사전에 왼쪽으로 회전되어 있는지를 제어한다. " M사이즈" 필드의 값은 " R" 비트가 " 1" 이면 회전량을 결정한다. 선택된 데이타는 이하 상술되는 32 비트 마스크를 발생시키는 확장 회로에 공급된다.

상태 레지스터(210)의 " A사이즈" 필드는 다중 산술 논리 유닛의 연산 동안 다중 플래그 레지스터(211)에 저장된 데이타를 제어한다. 전술한 바와 같이, 양호한 실시예에서, 산술 논리 유닛(230)은 각각 32 비트, 16 비트 및 8 비트의 데이타를 사용하는 하나, 2개 또는 4개의 분리된 섹션에 사용될 수 있다. 다중 산술 논리 유닛의 연산 실행시 " A사이즈" 필드는 규정된 데이타 사이즈를 통해 산술 논리 유닛의 각각의 분리된 결과의 상태 정보를 기록하는데 사용되는 다중 플래그 레지스터(211)의 비트수를 표시한다. 표 4에는 다중 플래그 레지스터(211)의 비트 설정이 요약되어 있다.

데이타 사이즈 비트	MF 비트를 설정하는 ALU 캐리-아웃 비트				MF 비트를 설정하는 제로(0)와 동일한 ALU 결과 비트			
	3	2	1	0	3	2	1	0
8	31	23	15	7	31-24	23-16	15-8	7-0
16	-	-	31	15	-	-	31-16	15-0
32	-	-	-	31	-	-	-	31-0

[표 4]

표 4는 8, 16 및 32 비트의 데이타 사이즈의 경우에 대해서만 표시하였다. 본 기술 분야에 숙련된 자는 64 비트, 4 비트, 2 비트 및 1 비트의 데이타 사이즈의 경우를 포함하도록 표 4를 확장하는 방법을 용이하게 알 수 있을 것이다. 또한, 전술한 설명은 다중 플래그 레지스터(211)에 캐리 또는 제로 상태를 저장하는 것과 관련된 것이다. 또한 부정 및 오버플로와 같은 다른 상태 비트를 저장하는 것도 실시가능하다.

다중 플래그 레지스터(211)는 각 산술 논리 유닛의 연산의 실행시 복수의 비트 위치를 왼쪽으로 회전시킬 수 있다. 회전량은 상부에 제공되어 있다. 다중산술 논리 유닛이 연산을 실행할 때, 결과 상태 비트 설정은 설정중인 이들 비트에 대한 회전보다 우위를 점한다. 다중 산술 논리 유닛이 연산을 실행할 때, 회전에 대한 대안은 결과 상태가 설정되지 않는 다중 플래그 레지스터(211)의 모든 비트를 클리어 하는 것이다. 이러한 클리어링은 마스크 데이타가 그 명령에 사용되는 경우 마스크 데이타의 발생 후에 행해진다. 다중 플래그 레지스터(211)가 산술 논리 유닛 결과를 기록하는 것과 동일한 시간에 소프트웨어에 의해 기록되면, 모든 비트를 로드하는 소프트웨어 기록 동안 양호한 연산이 행해진다. 이와 같이 소프트웨어 기록은 다중 플래그 레지스터(211)의 회전 및 클리어링보다 우선된다.

제7도는 산술 논리 유닛(230)을 다중 섹션으로 분리하는 것을 나타낸다. 제 7도에 도시한 바와 같이, 산술 논리 유닛(230)의 32 비트는 각 8 비트의 4개의 섹션으로 분리된다. 섹션(301)은 산술 논리 유닛 비트 7-0을 포함하고, 섹션(302)은 비트 15-8을 포함하며, 섹션(303)은 비트 23-16을, 섹션(304)은 비트 31-24를 포함한다. 제7도에는 간단화를 위해 이들 섹션의 입력 및 출력이 표시되어 있지 않다. 섹션(301, 302, 303) 각각 내의 캐리 경로는 공지된 기술에서와 같다.

멀티플렉서(311, 312, 313)는 섹션(301, 302, 303, 304)간의 캐리 경로를 제어한다. 이들 멀티플렉서 각각은 3개의 입력 중 하나를 선택하도록 제어된다. 제1 입력은 전 멀티플렉서의 출력으로부터, 혹은 제1 멀티플렉서(311)의 경우에는 비트 0 캐리-인 발생기(246)로부터의 캐리 룩 어헤드 경로(carry look ahead path)이다. 이러한 캐리 룩 어헤드 경로 및 이들의 사용은 종래 기술이므로 별도로 설명하지 않는다. 제2 섹션은 산술 논리 유닛(230)의 대응 섹션의 최종 비트로부터의 캐리-아웃이다. 마지막 섹션은 비트 0 캐리-인 발생기(246)로부터의 캐리-인 신호이다. 멀티플렉서(314)는 산술 논리 유닛(230)의 출력 캐리 경로를 제어한다. 멀티플렉서(314)는 멀티플렉서(313)에 의해 선택되는 캐리-아웃으로부터 캐리 룩 어헤드 경로 혹은 섹션(304)으로부터 비트 31에 대한 캐리-아웃 신호를 선택한다.

멀티플렉서(311, 312, 313, 314)는 선택된 데이타 사이즈에 따라 제어된다. 정상적인 경우에 있어서, 산술 논리 유닛(230)은 32 비트 데이타 워드에 따라 연산한다. 이것은 " 110" 과 동일한 상태 레지스터(210)의 " A사이즈" 필드로 표시된다. 이러한 경우에, 멀티플렉서(311)는 비트 7로부터 캐리-아웃을 선택하고, 멀티플렉서(312)는 비트 15로부터 캐리-아웃을 선택하며, 멀티플렉서(313)는 비트 23으로부터 캐리-아웃을 선택하고, 멀티플렉서(314)는 비트 31로부터 캐리-아웃을 선택한다. 이와 같이 4개의 섹션(301, 302, 303, 304)은 단일의 32 비트 산술 논리 유닛으로 서로 결합된다. 상태 레지스터(210)가 " 101" 의 " A사이즈" 필드를 통해 반 워드를 선택하면, 멀티플렉서(311)는 비트 7로부터 캐리-아웃을 선택하고, 멀티플렉서(312)는 비트 0 캐리-인 발생기로부터 캐리-인을 선택하고, 멀티플렉서(313)는 비트 23으로부터 캐리-아웃을 선택하고, 멀티플렉서(314)는 비트 31로부터 캐리-아웃을 선택한다. 섹션(301, 302)은 16 비트 유닛 속에 접속되고, 섹션(303, 304)은 16 비트 유닛 속에 접속된다. 멀티플렉서(312)는 비트 0와 아주 유사한 비트 16에 대한 비트 0 캐리-인 신호를 선택하는데, 그 이유는 비트 16이 16 비트 반 워드의 제1 비트이기 때문이다. 상태 레지스터(210)가 " 100" 의 " A사이즈" 필드를 통해 바이트를 선택하면 멀티플렉서(311, 312, 313)는 비트 0 캐리-인 발생기(246)로부터 캐리-인을 선택한다. 섹션(301, 302, 303, 304)은 4개의 독립된 8 비트 유닛으로 분할된다. 각 멀티플렉서에서의 비트 0 캐리-인 신호의 선택은 비트 8, 16 및 24 가 8 비트 바이트의 각 제1 비트이므로 적절하다.

또한 제7도는 제로 결과 검출을 나타낸다. 각 8 비트 제로 검출 회로(321, 322, 323, 324)는 대응하는 8 비트 섹션으로부터의 결과가 모두 제로" 00000000" 이면 " 1" 출력을 발생시킨다. AND 게이트(332)는 16개 비트 31-16 모두가 0일 때 " 1" 을 발생시키기 위한 8 비트 제로 검출 회로(321, 322)에 유사하게 접속된다. 끝으로, AND 게이트(341)는 AND 게이트(331, 332)에 접속되며, 모든 32 비트 31-0가 0일 때 " 1" 을 발생시킨다.

다중 산술 논리 유닛의 연산 동안 다중 플래그 레지스터(211)는 명령에 따라 캐리-아웃 또는 제로 비교값을 저장할 수 있다. 이들 저장된 결과값은 후속 연산 동안 C-포트에 대한 마스크를 제어한다. 표 4는 저장된 상태 비트에 대한 발생지를 나타낸다. 다중 플래그 레지스터(211)가 캐리-아웃 신호를 저장하는 경우에, 상태 레지스터(210)의 " A사이즈" 필드는 저장된 캐리-아웃의 일치 및 수를 결정한다. " A사이즈" 필드가 워드 연산을 특정하면, 다증 플래그 레지스터(211)가 비트 31의 캐리-아웃 신호와 동일한 단일 비트를 저장한다. " A사이즈" 필드가 반 워드 연산을 특정하면, 다중 플래그 레지스터(211)가 비트 31 및 15의 각 캐리-아웃 신호와 동일한 2 비트를 저장한다. " A사이즈" 필드가 바이트 연산을 특정하면, 다중 플래그 레지스터(211)가 비트 31, 32, 15 및 7 각각의 캐리-아웃 신호와 동일한 4개의 비트를 저장한다. " A사이즈" 필드는 제로 결과값의 저장이 선택될 때 다중 플래그 레지스터(211)에 저장된 제로 결과값의 수 및 일치를 제어한다. " A사이즈" 필드가 워드 연산을 특정하면, 다중 플래그 레지스터(211)는 비트 31-0가 " 0" 인 경우를 나타내는 AND 게이트(341)의 출력과 동일한 신호 비트를 저장한다. " A사이즈" 필드가 반 워드 연산을 특정하면, 다중 플래그 레지스터(211)가 AND 게이트(331, 332) 각각의 출력과 동일한 2 비트를 저장한다. " A사이즈" 필드가 바이트 연산을 특정하면 다중 플래그 레지스터(211)가 8 비트 제로 검출 회로(321, 322, 323, 324) 각각의 출력과 동일한 4개의 비트를 저장한다.

본 발명의 영역 내에서는, 4 비트 연산의 8개 섹션; 2 비트 연산의 16개 섹션; 그리고 1 비트 연산의 32개 섹션과 같은 산술 논리 유닛(230)의 추가의 다중 연산을 기술적으로 실시할 수 있다. 상태 레지스터(210)의 " M사이즈" 및 " A사이즈" 필드는 상기와 같은 추가의 다중 연산 형태를 지원할 수 있는 코딩을 포함하고 있다. 본 기술 분야에 숙련된 자는 멀티플렉서 및 AND 게이트를 추가로 사용함에 의해 제7도에 도시한 회로를 용이하게 수정 및 연장시킬 수 있을 것이다. 이들 후자의 실행 가능한 선택은 산술 논리 유닛(230) 구성에 있어서 복잡성을 가중시키므로 양호한 실시예에서는 지원되지 않는다. 또한 이러한 기술은 64 비트 데이타를 사용하는 데이타 처리 장치로까지 확대될 수 있으며, 동일한 기술을 통해 이러한 확장을 실시할 수 있다.

데이타 레지스터 D7-D0로 지정된 데이타 레지스터(200)는 로컬 포트 데이타 버스(LBUS, 103), 글로벌 포트 발생지 데이타 버스(GSRC, 105) 및 글로벌 포트 목적지 데이타 버스(GDST, 107)에 접속된다. 데이타 레지스터(200)를 나타내는 장방형 내의 화살표는 데이타 액세스의 방향을 표시한다. 좌측점 화살표는 데이타 레지스터(200)로부터 재현되는 데이타를 표시한다. 우측점 화살표는 데이타 레지스터(200)에 기록되는 데이타를 표시한다. 로컬 포트 데이타 버스(LBUS, 103)는 데이타 발생지 또는 데이타 목적지로서의 데이타 레지스터(200)에 양방향 접속된다. 글로벌 포트 목적지 데이타 버스(GDST, 107)는 데이타 레지스터(200)에 기록되는 데이타에 대한 데이타 발생지로서의 데이타 레지스터(200)에 접속된다. 글로벌 포트 발생지 데이타 버스(GSRC, 107)는 정상적인 데이타 레지스터 모드 및 이하 기술하는 회전 레지스터 특징에서 데이타 레지스터(200)으로부터 재현되는 데이타에 대한 데이타 목적지로서의 데이타 레지스터(200)에 접속된다. 상태 레지스터 (210) 및 다중 플래그 레지스터(211)는 글로벌 포트 발생지 데이타 버스(GSRC, 106)를 통해 판독되어 글로벌 포트 목적지 데이타 버스(GDST, 107)를 통해 기록될 수 있다. 데이타 레지스터(200)는 승산기의 제1의 입력 버스(201), 승산기의 제2의 입력 버스(202), 산술 논리 유닛의 제1의 입력 버스(205) 및 산술 논리 유닛의 제2의 입력 버스(206)에 데이타를 공급한다. 데이타 레지스터(200)는 승산기의 목적지 버스(203) 및 산술 논리 유닛의 목적지 버스(204)로부터 입력 데이타를 수신하기 위해 접속된다.

레지스터 D7-D0로 지정된 데이타 레지스터(200)는 제8도에 도시한 바와 같이 256 비트 회전 레지스터를 형성하도록 접속된다. 이 회전 레지스터는 일반적으로 회전(ROT) 레지스터(208)이다. 이것은 8개의 32 비트 회전 레지스터 ROT0, ROT1, …ROT7를 포함하는 256 비트 레지스터를 형성한다. 제8도는 회전 레지스터 ROT0, TOT1,…ROT7의 정의를 부분적으로 나타낸다. 이들 회전 레지스터는 데이타 레지스터 D7-D0와 관련하여 사이드웨이로 정의된다. 회전 레지스터(208)는 이하 기술하는 바와 같이, 비-산술 논리 유닛 명령 DROT에 의해 회전될 수 있다. 이러한 회전 동안 데이타 레지스터 D7의 최하위 비트는 데이타 레지스터 D6 등의 최상위 비트로 회전된다. 데이타 레지스터 D0의 최하위 비트는 데이타 레지스터 D7의 최상위 비트에 다시 접속된다. 회전 레지스터(208)는 한번에 4개의 8 비트(1 바이트)씩 판독될 수 있다. 4개의 8 비트(1 바이트)는 아래의 표 5와 제8도에 도시한 바와 같이 데이타 레지스터(200) 각각에서 동일한 비트 수를 가진 비트의 각 옥테트(octet)이다.

회전 레지스터 비트	각 D7-D0 비트로부터의 비트의 옥테트
31-24	24
23-16	16
15-8	8
7-0	0

[표 5]

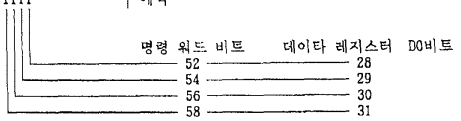
DROT 명령이 실행될 때 256 비트 회전 레지스터(208)는 1 비트 위치씩 오른쪽으로 회전된다. D7과 같은 각 레지스터의 각 바이트 A, B, C, D의 최하위 비트 0는 글로벌 포트 발생지 데이타 버스(GSRC, 105) 상에 출력되는 회전 레지스터의 특정 비트 수로 도시한 바와 같이 맵된다. 회전 레지스터(208)는 양호한 실시예에서는 판독만하지만 다른 실시예에서는 기록도 가능하다.

회전 레지스터(208)는 영상 회전(image rotation), 직교 변형(orthogonal transform) 그리고 미러 변형(mirror transform)에 유용하다. 8 개의 DROT 명령과 병렬로 회전 레지스터(208)로부터 메모리로의 32 비트 저장을 실행하면, 데이터의 4 개의 8×8 비트 패치를 시계 방향으로 90 도 회전시킨다. 회전된 데이타는 타켓 메모리 위치에 저장된다. 레지스터 로딩, 메모리 어드레스 저장, 및 데이타 사이즈 수정의 여러 가지 조합에 의해, 8×8 비트 패치의 다양한 시계 방향 및 반시계 방향 회전을 행할 수 있다. 그리고 큰 영역의 회전은 전체 바이트를 이동시킴으로써 행할 수 있다. 하나의 모드에서 레지스터 D7-D0로의 레지스터 파일 액세스를 제공하고, 그리고 DROT 연산에서 회전 레지스터 액세스를 제공하는 현저한 직교 구조(orthogonal structure)는 레지스터 파일에 대한 액세스만을 제공하는 것보다 약간 복잡하다.

데이타 레지시터 D0는 2중 기능을 갖는다. 다른 데이타 레지스터 D7-D1과 동일한 방식으로 정상적인 데이타 레지스터로서 사용될 수 있다. 또한, 데이타 레지스터 D0는 몇몇 기능을 실행할 때 어떤 특정한 기능을 정의할 수 있다. 데이타 레지스터 D0의 최상위 반 워드의 몇몇 비트는 모든 형태의 확장된 산술 논리 유닛 연산 동작을 특정한다. 데이타 레지스터 D0의 최하위 반 워드 비트의 몇몇은 다중의 승산 연산 동안 승산기의 선택을 특정할 수 있다. 데이타 레지스터 D0의 5개의 최하위 비트는 특정한 명령 분류에 사용되는 디폴트 원통형 회전량을 특정한다. 제9도는 데이타 유닛(110)의 연산을 특정할 때 데이타 레지스터 D0의 내용을 표시 한다.

데이타 레지스터 D0의 " FMOD" 필드(비트 31-28)는 확장된 산술 논리 유닛(EALU)의 연산을 요구하는 명령을 실행할 때 산술 논리 유닛의 기본 연산의 수정을 하게 한다. 표 6은 이들 수정자의 선택을 나타낸다. 표 6에 표시한 바와 같이, 몇몇 명령 포맷에 있어서 특정한 명령 워드 비트는 함수 수정자와 같이 동일한 방식으로 디코드된다. 이에 대해서는 후술한다.

함수 수정자 코드	수정 실행
0000	정상 연산
0001	카인
0010	마스크 발생 명령이면 %! 마스크 발생 명령이 아니면 LMO
0011	마스크 발생 명령이면(%! 및 카인) 마스크 발명 명령이 아니면 RMO
0100	A-포트=0
0101	A-포트=0 및 카인
0110	마스크 발생 명령이면(A-포트=0 및 %!) 마스크 발생 명령이 아니면 LMBC
0111	마스크 발생 명령이면(A-포트=0 및 %! 및 카인) 마스크 발생 명령이 아니면 RMBC
1000	다중 산술 논리 유닛 연산,(캐리-아웃-->다중 플래그 레지스터)
1001	다중 산술 논리 유닛 연산, 제로 결과-->다중 플래그 레지스터
1010	다중 산술 논리 유닛 연산(캐리-아웃-->다중 플래그 레지스터) 상태 레지스터의 "Asize" 필드에 의해 회전
1011	다중 산술 논리 유닛 연산, 제로 결과-->다중 플래그 레지스터 상태 레지스터의 "Asize" 필드에 의해 회전
1100	다중 산술 논리 유닛 연산, 캐리-아웃-->다중 플래그 레지스터 다중 플래그 레지스터 클리어
1101	다중 산술 논리 유닛 연산, 제로 결과-->다중 플래그 레지스터, 다중 플래그 레지스터 클리어,
1110	예약
1111	예약



[표 6]

표 6에 제시된 수정 연산에 대한 설명이 이하에 설명되어 있다.

" FMOD" 필드가 " 0000" 이면, 정상, 비수정 연산된다. 수정 " 카인" 의 경우는 산술 논리 유닛(230)의 비트 0에 대한 캐리-인이 상태 레지스터(210)의 " C" 비트로 된다. 이로인해 캐리 가산(add with carry), 차용 감산(subtract with borrow) 및 차용 부정(negate with borrow) 연산을 할 수 있다. 수정 " %!" 는 마스크 발생을 대상으로 한다. " %!" 수정이 행해지면 마스크 발생기(239)가 올 " 0" 외에 제로 회전량에 대한 올 " 1" 을 발생시킨다. 이러한 기능은 C-포트로 공급되는 올 " 0" 의 마스크가 올 " 1" 이 공급된 것처럼 연산하도록 마스크 발생기(239)에 의해 발생되는 마스크를 수정하거나 또는 산술 논리 유닛(230)의 함수를 수정함으로써 이행될 수 있다. 이러한 수정은 몇몇의 회전 연산에 유용하다. 수정, 즉 " LMO" , " RMO" , " LMBC" 및 " RMBC" 는 LMO/RMO/LMBC/TMBC 회로(237)의 제어를 지정한다. 수정 " LMO" 은 제2의 산술 입력의 최좌측 " 1" 을 찾아낸다. 수정 " LMBC" 는 신호 비트(비트 31)와 다른 최좌측 비트를 찾아낸다. " RMBC" 수정은 제1 비트(비트 0)와 다른 최우측 비트를 찾아낸다. 이들 수정은 산술 논리 유닛(230)의 C 포트가 마스크 발생기(239)로부터 마스크를 수신하지 못했을 때만 관련된다.

수정 " A-port=0" 는 산술 논리 유닛(230)의 A-포트로의 입력이 유효하게 제로(0)임을 표시한다. 이것은 제로 출력을 제공하는 멀티플렉서 Amux232를 통해 발생되거나, 혹은 산술 논리 유닛(230)의 연산이 동일한 효과를 가진 방식으로 수정될 수 있다. " A-port=0" 수정은 특정한 부정, 절대값 및 우측 이동 연산에 사용된다. " 다중 산술 논리 유닛 연산" 수정은 산술 논리 유닛(230)의 캐리 경로를 하나 또는 그 이상으로 분할하여 둘 또는 그 이상의 독립된 산술

논리 유닛의 연산을 병렬로 효과적으로 하기 위한 것을 나타낸다. 상태 레지스터(210)의 " A사이즈" 필드는 상기 다중 산술 논리 유닛 섹션의 수를 제어한다.

다중 플래그 레지스터(211)는 다중 산술 논리 유닛 연산의 선택 수와 동일한 복수의 상태 비트를 저장한다. " 캐리-아웃-->다중 플래그" 수정에 있어서, 캐리-아웃 비트 또는 비트들이 다중 플래그 레지스터(211)에 저장된다. " 제로 걸과-->다중 플래그" 수정에서는 대응하는 산출 논리 유닛 섹션에 대한 제로의 결과 표시가 다중 플래그 레지스터(211)에 저장된다. 이러한 처리는 다중 플래그 레지스터(211)의 설명과 더불어 상술하였다. 저장 연산 동안, 다중 플래그 레지스터(211)내의 비트들은 " 회전" 수정에 따라 회전되거나 또는 " 클리어" 수정에 따라 클리어될 수 있다. 이들 선택은 다중 플래그 레지스터(211)의 설명과 함께 상술하였다.

데이타 레지스터 D0의 " A" 비트(비트 27)는 산술 논리 유닛(230)이 확장 산술 논리 유닛의 연산동안 산술 또는 부울 논리 연산을 실행하는지를 제어한다. " A" 비트가 " 0" 이면 비트 0-캐리-인 발생기로부터 산술 논리유닛(230)의 비트 0으로의 캐리-인은 일반적으로 " 0" 이다. 이하 상세히 설명하는 바와 같이, 특정한 확장 산술 논리 유닛 연산은 " A" 비트가 논리 연산을 나타내는 " 0" 인 경우에도 " 1" 의 캐리-인 비트를 가질 수 있다.

데이타 레지스터 D0의 " EALU" 필드(비트 19-26)는 확장 산술 논리 유닛 연산을 정의한다. 8개의 " EALU" 필드 비트는 모든 형태의 확장 산술 논리 유닛 연산에 사용되는 산술 논리 유닛 기능 코드 비트(arithmetic logic unit function code bit)를 특정한다. 이들 비트들은 산술 논리 유닛(230)에 대한 제어 신호가 된다. 이들은 " FMOD" 필드에 따라 직접 또는 수정되어 산술 논리 유닛(230)으로 전달될 수 있다. 몇몇의 명령에 있어서 " EALU" 필드의 비트는 반전되어 " EALUF" 로 되거나 혹은 확장된 산술 논리 유닛이 오연산하게 된다. 이 경우에 산술 논리 유닛(230)에 공급되는 8개의 제어 비트가 반전된다.

데이타 레지시터 D0의 " C" 비트(비트 18)는 확장된 산술 논리 유닛의 연산 동안 산술 논리 유닛(230)의 비트 0에 대한 캐리-인을 나타낸다. 확장된 산술 논리 유닛 연산 동안 산술 논리 유닛의 비트 0으로의 캐리-인 값은 " C" 비트로 주어진다. 이로 인해 넌(NON)-EALU 연산의 식에 의해서 보다, 캐리-인 값이 직접 특정될 수 있다.

데이타 레지스터 D0의 " I" 비트(비트 17)는 역 캐리-인 비트로서 표시된다. " C" 비트 및 " S" 비트(이하 정의됨)와 함께 " I" 비트는 산술 논리 유닛 연산의 기능 코드 비트가 반전될 때 산술 논리 유닛(230)의 비트 0으로 캐리-인을 반전시킬 수 있는지를 결정한다. 이에 대해서는 후술한다.

데이타 레지스터 D0의 " S" 비트(비트 16)는 부호 확장 섹션을 표시한다. " S" 비트는 확장된 산술 논리 유닛의 연산(" A" 비트 = 1)을 실행할 때 사용된다. " S" 비트가 " 1" 이면, 산술 논리 유닛 코드 비트 F3-F0(비트 22-19로부터 발생)는 데이타의 제1의 산술 논리 유닛의 입력 버스(206)의 부호 비트(비트 31)가 " 0" 일때 반전되고, 상기 부호 비트가 " 1" 이면 반전되어서는 안된다. 이하에서는, 산술 논리 유닛 코드 비트 F3-F0를 조건부로 반전시키는 효과에 대하여 설명한다. 이러한 반전은 특정한 산술 연산에 있어서 회전된 입력을 부호 확장시키는데 유리하다. 확장된 산술 논리 유닛의 연산이 부울(" a" 비트 = 0)이면 " S" 비트는 무시되고 산술 논리 유닛 코드 비트 F3-F0가 변화하지 않는다.

표 7은 데이타 레지스터 D0의 " C" , " I" 및 " S" 비트의 상호 작용을 표시한다. " I" 비트 또는 제1의 입력 부호에 대한 " X" 엔트리는 비트가 결과, 즉 돈케어(don't care) 조건을 제어하지 않음을 표시한다.

S	I	제1 입력 부호	역 C	역 F3-F0
0	X	X	아니오	아니오
1	0	0	아니오	아니오
1	0	1	아니오	예
1	1	0	아니오	아니오
1	1	1	예	예

[표 7]

" S" 비트가 " 1" 이고 산술 논리 유닛(230)의 B 포트용으로 예정된 제1 입력의 부호 비트가 " 0" 이면, " C" 비트값으로 설정되는 산술 논리 유닛(230)의 비트 0에 대한 캐리-인의 값은 " I" 비트의 값에 따라 선택적으로 반전될 수 있다. 이로써 입력의 부호에 따라 캐리-인이 선택적으로 반전되거나 또는 반전되지 않는다. 또한, " S" 비트가 " 1" 이면, 입력의 부호에 따라 산술 논리 유닛 코드 비트 F3-F0가 선택적으로 반전된다. 산술 논리 유닛 코드 비트 F3-F0의 반전의 이러한 선택은 " FMOD" 필드에 의해 무시될 수 있다. " FMOD" 필드가 " 캐리-인=상태 레지스터의 캐리 비트" 를 특정하면, 캐리-인은 " S" 및 " I" 비트의 값이 무엇이든 간에 상태 레지스터(210)의 " C" 비트와 동일하게 된다. 또, 산술 논리 유닛(230)의 비트 0에 대한 캐리-인은 " A" 비트가 부울 연산을 표시하는 " 0" 라 해도 확장된 산술 논리 유닛의 동작을 위한 " C 비트를 통해 " 1" 로 설정되게 된다.

데이타 레지스터 D0의 " N" 비트(비트 15)는 산술 논리 유닛의 동작을 분할 또는 다중 섹션으로 실행할 때 사용된다. 이 " N" 비트는 비(NON)-다중 마스크 비트로 부른다. " FMOD" 필드를 통해 다중 동작을 특정하는 몇몇의 확장된 산술 논리 유닛의 동작에 있어서, 명령은 마스크 발생기(239)를 통해 산술 논리 유닛(230)의 C-포트로 전달되는 마스크를 특정한다. 이 " N" 비트는 마스크가 산술 논리 유닛(230)과 동일한 섹션 수로 분할되는지를 결정한다. 상기 다중 섹션 수는 상태 레지스터(210)의 " A사이즈" 필드에 의해 설정된다. " N" 비트가 " 0" 이면, 마스크는 다중 마스크로 분할된다. " N" 비트가 " 1" 이면, 마스크 발생기(239)는 단일의 32비트 마스크를 발생시킨다.

" E" 비트(비트 14)는 명시된 다중 캐리-인을 표시한다.

이 비트는 산술 논리 유닛(230)의 C-포트로의 입력에 의해 작동시간에서 특정되도록 한다. " A" 비트 및 " E" 비트가 " 1" 이고 " FMOD" 필드가 캐리-인 함수를 표시하지 않으면, " S" , " I" 및 " C" 비트의 효과는 무효로 된다. 다중 산술 동안 각 섹션으로의 캐리 입력은 C포트 및 함수 신호 F0로 입력된 대응 섹션의 최하위 비트의 배타적 OR로서 간주된다. 다중 산술이 선택되지 않으면 산술 논리 유닛(230)의 비트 0에 대한 단일 캐리-인은 C 포트 및 함수 신호 F0로 입력된 최하위 비트(비트 0)의 배타적 OR이다. 이것은 특히 다른 기능이 다른 섹션에서 행해지는 다중 산술을 행하는데 유용하다. 하나의 확장된 산술 논리 유닛의 연산은 (A ^ B) & Cl(A ^ B) & C에 대응한다. C-포트 입력용 마스크에 의해, 올 " 0" 를 가진 섹션이 " 0" 의 적절한 캐리-인을 가진 가산을 발생시키고, 올 " 1" 의 섹션이 " 1" 의 적절한 캐리-인을 가진 감산을 발생시킨다.

데이타 레지스터 D0의 " DMS" 필트(비트 12-8)는 승산기에 따른 이동을 정의한다. 이 이동은 결과값을 세이브하거나 혹은 결과 값을 라운딩 논리로 전달하기에 앞서 곱 좌측 이동기(224)에서 발생된다. 이러한 좌측 이동 동안 완전히 이동된 최상위 비트는 폐기되고, 제로가 최하위 비트로 이동된다. " DMS" 필드는 임의의 승산/확장된 산술 논리 유닛의 동작 동안 유효하다. 양호한 실시예에서 데이타 레지스터 D0 비트 9-8는 좌측으로 이동하는 0,1,2 또는 3의 위치를 선택한다. 표 8은 디코딩을 표시한다.

DMS 필드		좌측 이동량
9	8	
0	0	0
0	1	1
1	0	2
1	1	3

[표 8]

" DMS" 필드는 0에서 31의 위치까지 좌측으로 이동된 양을 표시할 수 있는 5개의 비트를 포함한다. 양호한 실시예에서, 곱 좌측 이동기(224)는 크기 및 복잡성으로 인해 0 내지 3까지 이동하도록 제한되어 있다. 따라서, 데이타 레지스터 D0의 비트 12-10는 좌측 이동량을 설정하는데 무시된다. 그러나, 필요한 경우 " DMS" 필드로부터 0 내지 31 위치의 전범위내에서 좌측 이동량을 제공하도록 실시할 수 있다.

데이타 레지스터 D0의 " M" 비트(비트 7)는 다중 승산 연산을 표시한다. 승산기(220)는 32 비트 결과값을 발생시키기 위해 2개의 16비트 수를 승산하거나 또는 한쌍의 16비트 결과값을 발생시키기 위해 2쌍의 8 비트수를 동시에 승산할 수 있다. 이 " M" 비트는 " M" =0 이면 하나의 16 ×16배, 또는 " M" =1 이면, 2개의 8 x 8배를 선택할 수 있다. 이러한 연산은 다중 산술 논리 유닛 연산과 유사하며 이하 상세히 설명한다.

데이타 레지스터 D0의 " R" 비트(비트 6)는 라운딩 동작이 승산기(220)로부터의 결과에 따라 발생하는지를 특정한다. " R 비트가 " 1" 이면, 승산기(220)의 연산과 함께 이하 설명되는 라운딩 동작이 발생한다. " R" 비트가 " 0" 이면, 어떤 라운딩도 발생되지 않고 승산기(220)로부터의 32 비트 결과 값이 목적지 레지스터에 기록된다. 데이타 레지스터 D0에 특정한 비트를 사용하는 것은 단지 이러한 모드를 트리거링하기 위한 양호한 실시예임을 주목해야 한다. 특정한 명령 워드비트를 통해 라운딩 모드를 동일하게 인에이블 할 수 있다.

데이타 레지스터 D0의 " DBR" 필드(비트 4-0)는 특정한 명령 동안 사용되는 원통형 회전기(235)의 디폴트 원통형 회전량을 특정한다. " DBR" 필드는 원통형 회전기(235)가 좌측으로 회전하는 비트 위치의 수를 특정한다. 이들 5비트는 0 내지 31 위치의 좌측 회전을 특정할 수 있다. " DBR" 필드의 값은 또한 멀티플렉서 Mmux(234)를 통해 마스크 발생기(239)에 공급될 수 있다. 마스크 발생기(239)는 산술 논리 유닛(230)의 C-포트로 공급되는 마스크를 형성한다. 마스크 발생기(239)의 동작에 대해서는 이하 설명한다.

승산기(220)는 하드웨어 싱글 사이클 승산기이다. 상술한 바와 같이, 승산기(220)는 32 비트의 결과를 얻기 위해 한쌍의 16비트수를 승산하거나 또는 동일한 32 비트 데이타 워드에서 2개의 16비트 결과를 얻기 위해 2쌍의 8비트수를 승산하도록 동작된다.

산술 논리 유닛(230)은 데이타 유닛(110) 내에서 산술 및 논리 연산을 수행한다. 산술 논리 유닛(230)은 3개의 입력 산술 및 논리 연산을 수행하기 위한 3개의 입력 포트를 포함한다. 복수의 버스 및 보조 하드웨어가 3개의 입력을 제공한다.

입력 A 버스(241)는 산술 논리 유닛(230)의 A-포트에 데이타를 제공한다. 멀티플렉서 Amux(232)는 명령에 따라 승산기의 제2의 입력 버스(202) 또는 산술 논리 유닛의 제1의 입력 버스(205)로부터 입력 A버스(241)에 데이타를 제공한다. 승산기의 제2의 입력 버스(202) 상의 데이타는 데이타 레지스터(200) 중 선택된 하나로부터 또는 멀티플렉서 Imux(222) 및 버퍼(223)를 통해 명령의 즉시 필드로부터 야기될 수 있다. 산술 논리 유닛의 제2의 입력 버스(205) 상의 데이타는 데이타 레지스터(200) 중 선택된 하나로부터 또는 버퍼(106)를 통해 글로벌 포트 발생지 데이타 버스 GSRC 버스(105)로부터 야기될 수 있다. 이와 같이 산술 논리 유닛(230)의 A-포트로 공급되는 데이타는 데이타 레지스터(200) 중 하나로부터, 명령 워드의 즉시 필드로부터 야기되거나 또는 글로벌 발생지 데이타 버스(GSRC, 105) 및 버퍼(106)를 통해 디지탈 영상/그래픽 프로세서(71)의 다른 레지스터로부터의 장거리 발생지일 수 있다.

입력 B버스(242)는 산술 논리 유닛(230)의 B-포트로 데이타를 공급한다. 원통형 회전기(235)는 입력 B버스(242)에 데이타를 공급한다. 이와 같이 원통형 회전기(235)는 산술 논리 유닛의 제2 입력 버스(206)로부터 데이타를 수신

한다. 산술 논리 유닛의 제2의 입력 버스(206)는 데이타 레지스터(200) 중 선택된 하나로부터 데이타를 공급하고, 버퍼(104)를 통해 글로벌 포트 발생지 데이타 버스 Gscr 버스(105)로부터 데이타를 공급하며, 또한 버퍼(236)로부터 특정한 데이타 워드를 공급한다. 버퍼(236)는 인에이블되면 산술 논리 유닛의 제2의 입력 버스(206)에 " 00000000000000000000000000000001" 의 일정한 32 비트 데이타(소위 헥사 " 1")를 공급한다. 이하, 헥사로 진행되는 데이타 또는 어드레스를 헥사데시멀이라 한다. 글로벌 포트 발생지 데이타 버스(Gscr, 105)로부터의 데이타는 전술한 바와 같이 장거리 발생지로서 원통형 회전기(235)에 공급될 수 있다.

버퍼(236)가 인에이블 되면, 원통형 회전기(235)는 입력 버스(242) 상에 임의의 상수 형태 즉 2^N을 발생시키는데, 여기서 N은 원통형 회전량이다. 이런 형태의 상수는 32 비트 데이타 워드의 단일 비트만을 제어할 수 있는 연산에 유용하다. 산술 논리 유닛의 제2 입력 버스(206) 및 원통형 회전기(235)에 공급되는 데이타는 명령에 따른다.

원통형 회전기(235)는 0로부터 32 위치까지의 수신된 데이타를 회전시키는 32 비트 회전기이다. 이것은 좌측 회전기이지만, n 비트의 우측 회전은 좌측으로 회전하는 32-n 비트에 의해 얻을 수 있다. 회전 버스(244)로부터 입력된 5개의 비트는 원통형 회전기(235)에 의해 제공되는 회전량을 제어한다. 이러한 회전은 원형이며 비트 손실은 없다. 원통형 회전기(235)의 좌측으로부터 회전된 비트는 다시 우측으로 순환된다. 멀티플렉서 SmuX(231)는 회전 버스(244)를 제공한다. 멀티플렉서 Smux(231)는 여러개의 입력을 갖는다. 이들 입력은, 승산기의 제1의 입력 버스(201)의 5개의 최하위 비트; 승산기의 제2의 입력 버스(202)의 5개의 최하위 비트; 데이타 레지스터 D0의 " DBR" 필드로부터의 5개의 비트; 그리고 일정한 5개의 제로 비트 " 00000" 를 포함한다. 승산기의 제2의 입력 버스(202)가 멀티플렉서 Imux(222 및 223)를 통해 즉시 데이타를 수신하기 때문에 병렬 워드는 원통형 회전기(235)에 즉시 회전량을 공급할 수 있다. 멀티플렉서 Smux(231)는 명령에 따라 원통형 회전기(235)에서 회전량을 결정하기 위해 이들 입력 중 하나를 선택한다. 이들 회전량 각각은 5개의 비트이므로, 0 내지 31 비트의 범위에서 좌측 회전을 설정할 수 있다.

또, 원통형 회전기(235)는 멀티플렉서 Bmux(227)에 데이타를 공급한다. 이로 인해 원통형 회전기(235)로부터 회전된 데이타가 산술 논리 유닛(230) 동작과 병렬로 승산기 목적지 버스(203)를 통해 데이타 레지스터(200) 중 하나에 저장되게 된다. 원통형 회전기(235)는 멀티플렉서 Bmux(227)를 통해 멀티플렉서 Rmux(221)와 멀티플렉서 목적지 버스(203)를 공유한다. 이렇게 회전된 데이타는 승산 동작이 발생되면 세이브될 수 없다. 양호한 실시예에서 재기록 방법은 특히 확장된 산술 논리 유닛의 연산에 의해 지원되며, 산술 논리 유닛(230)의 결과와 마찬가지로 원통형 회전기(235) 결과에 대해 동일한 레지스터 목적지를 특정함으로써 디스에이블 될 수 있다. 이러한 경우에는 산술 논리 유닛의 목적지 버스(204) 상에 나타나는 산술 논리 유닛(230)의 결과만이 세이브된다.

상술된 설명은 원통형 회전기(235)에 관한 것이지만, 본 기술 분야에 숙련된 자는 데이타 순환되지 않은 이동기를 사용하여 실질적인 일체감을 얻을 수 있음을 이해할 수 있을 것이다. 특히, 산술 논리 유닛(230)의 B 포트에 모든 비트들이 사용되지 않는 이동 및 마스크 동작에 있어서, 회전 버스(244)에 의해 제어되는 이동기가 필요한 기능을 제공한다. 이러한 경우에 회전 버스(244) 상의 최상위 비트와 같은 추가의 비트는 우측 이동 또는 좌측 이동을 형성하는지를 표시한다. 회전 버스(244) 상의 5개의 비트는 이동의 크기를 표시하기 위해 계속해서 요구된다. 따라서, 이하의 설명으로 부터 이동기는 많은 예를 통해 원통형 회전기(235)로 대체될 수 있음을 이해할 수 있다.

입력 C버스(243)는 산술 논리 유닛(230)의 C-포트에 데이타를 공급한다. 멀티플렉서 Cmux(233)는 입력 C버스(243)에 데이타를 공급한다. 멀티플렉서Cmux(233)은 4개의 발생지로부터 데이타를 수신한다. LMO/RMO/LMBC/RMBC 회로(237), 확장 회로(238), 승산기의 제2의 입력 버스(202) 및 마스크 발생기(239)가 있다.

LMO/RMO/LMBC/RMBC 회로(237)는 최좌측 " 1" , 최우측 " 1" , 명령 또는 데이타 레지스터 D0의 " FMOD" 필드

에 따라 산술 논리 유닛의 제2의 입력 버스(206) 상의 데이타의 최좌측 비트 수정 또는 최우측 비트 수정을 결정하는 전용 하드웨어 회로이다. LMO/RMO/LMBC/RMBC 회로(237)는 멀티플렉서 Cmux(233)에 검출된 양에 대응하는 값의 32 비트수를 제공한다. 최좌측 비트 수정은 부호 비트 32와 다른 최좌측 비트의 위치로서 정의되며, 최우측 비트 수정은 비트 0와 다른 최우측 비트의 위치로서 정의된다. 그 결과는 아래의 표 9에 표시되어 있는 검출된 비트 위치에 대응하는 2진수이다. 이 값들은 검출된 비트 위치의 큰 끝(big endian) 비트수가 효과적이며, 이들의 결과는 31 - (비트 위치)이다.

비트 위치	결과
0	31
1	30
2	29
3	28
4	27
5	26
6	25
7	24
8	23
9	22
10	21
11	20
12	19
13	18
14	17
15	16
16	15
17	14
18	13
19	12
20	11
21	10
22	9
23	8
24	7
25	6
26	5
27	4
28	3
29	2
30	1
31	0

[표 9]

이러한 결정은 표준화 및 최좌측 또는 최우측 " 1" 또는 영상의 에지와 같이 수정된 비트인지를 가려내기 위한 영상 압축에 유효하다. LMO/RMO/LMBC/RMBC 회로(237)는 가상 속도 경로이므로, 산출 논리 유닛의 제2 입력 버스(206)에 결합된 발생지는 데이타 레지스터(200)의 하나로 제한되는 것이 바람직하다. 최좌측 " 1" 및 최우측 " 1" 의 연산에 있어서, 상태 레지스터(210)의 오버플로를 나타내는 " V" 비트는 발생지에 " 1" 이 전혀 없으면 " 1" 로 설정되고 발생지에 " 1" 이 있으면 " 0" 로 설정된다. 최좌측 비트 변화 및 최우측 비트 변화 연산에 있어서, 발생지의 모든 비트가 같

으면 " V" 비트는 " 1" 로 설정되고, 어떤 변화가 검출되면 " 0" 으로 설정된다. " V" 비트가 이들 임의의 동작에 의해 " 1" 로 설정되면, LMO/RMO/LMBC/RMBC의 결과는 효과적으로 32가 된다. 상태 레지스터(210)의 동작과 관련한 상세한 설명은 상술한 바와 같다.

확장 회로(238)는 다중 플래그 레지스터(211) 및 상태 레지스터(210)로부터 입력을 수신한다. 상술한 상태 레지스터(210)의 " M사이즈" 필드에 따라, 확장 회로(238)는 32 비트를 충족시키기 위해 다중 플래그 레지스터(211)에 저장된 최하위 비트의 몇몇을 복제한다. 확장 회로(238)는 최하위 비트를 32번 확장하고, 2개의 최하위 비트를 16번 확장하며 혹은 4개의 최하위 비트를 8번 확장할 수 있다. 상태 레지스터(210)의 " A사이즈" 필드는 32 비트의 산술 논리 유닛(230)이 독립된 데이타 동작을 위한 독립된 섹션으로 분할되는 처리를 제어한다. 이것은 픽셀 사이즈에 있어서 산술 논리 유닛(230)의 32 비트 폭보다 작은 연산에 유용하다. 이러한 처리 및 그 사용의 예에 대해서는 후술한다.

마스크 발생기(239)는 멀티플렉서 Cmux(233)를 통해 입력 C버스(243)에 공급될 수 있는 32 비트 마스크를 발생시킨다. 마스크는 멀티플렉서 Mmux(234)로부터 입력되는 5비트에 따라 발생된다. 멀티플렉서 Mmux(234)는 승산기의 제2의 입력버스(202)의 5개의 최하위 비트, 또는 데이타 레지스터 D0로부터의 " DBR" 필드를 선택한다. 양호한 실시예에서는, 마스크 발생기(239)에 의해 값 N의 입력이 발생하여 최하위 비트로서 N 올 " 1" 을 갖고, 최상위 비트로서 32-N 올 " 0" 를 갖는 마스크가 발생되게 된다. 이것이 N개의 우측 자리맞춤 올 " 1" 을 갖는 출력을 형성한다. 이것은 마스크 발생기(239)의 4개의 가능한 연산 방법 중 바로 하나이다. 제2 실시예에서는, 마스크 발생기(239)가, 최하위 비트에서는 N 올 " 0" 이고, 최상위 비트에서는 N-32 올 " 1" 인 N개의 우측 자리맞춤 올 " 0" 을 갖는 마스크를 발생시킨다. 동일하게 N개의 좌측 자리맞춤 올 " 1" 또는 N개의 좌측 자리맞춤 올 " 0" 을 가진 마스크를 발생시키도록 마스크 발생기(239)에 대해 실시 가능하다. 표 10은 다중 산술이 선택되지 않을 때의 양호한 실시예에 따른 마스크 발생기(239)의 동작을 나타낸다.

마스크 발생기 입력	마스크 - 비 다중 연산
0 0 0 0 0	0000 0000 0000 0000 0000 0000 0000 0000
0 0 0 0 1	0000 0000 0000 0000 0000 0000 0000 0001
0 0 0 1 0	0000 0000 0000 0000 0000 0000 0000 0011
0 0 0 1 1	0000 0000 0000 0000 0000 0000 0000 0111
0 0 1 0 0	0000 0000 0000 0000 0000 0000 0000 1111
0 0 1 0 1	0000 0000 0000 0000 0000 0000 0001 1111
0 0 1 1 0	0000 0000 0000 0000 0000 0000 0011 1111
0 0 1 1 1	0000 0000 0000 0000 0000 0000 0111 1111
0 1 0 0 0	0000 0000 0000 0000 0000 0000 1111 1111
0 1 0 0 1	0000 0000 0000 0000 0000 0001 1111 1111
0 1 0 1 0	0000 0000 0000 0000 0000 0011 1111 1111
0 1 0 1 1	0000 0000 0000 0000 0000 0111 1111 1111
0 1 1 0 0	0000 0000 0000 0000 0000 1111 1111 1111
0 1 1 0 1	0000 0000 0000 0000 0001 1111 1111 1111
0 1 1 1 0	0000 0000 0000 0000 0011 1111 1111 1111
0 1 1 1 1	0000 0000 0000 0000 0111 1111 1111 1111
1 0 0 0 0	0000 0000 0000 0000 1111 1111 1111 1111
1 0 0 0 1	0000 0000 0000 0001 1111 1111 1111 1111
1 0 0 1 0	0000 0000 0000 0011 1111 1111 1111 1111
1 0 0 1 1	0000 0000 0000 0111 1111 1111 1111 1111
1 0 1 0 0	0000 0000 0000 1111 1111 1111 1111 1111
1 0 1 0 1	0000 0000 0001 1111 1111 1111 1111 1111
1 0 1 1 0	0000 0000 0011 1111 1111 1111 1111 1111
1 0 1 1 1	0000 0000 0111 1111 1111 1111 1111 1111
1 1 0 0 0	0000 0000 1111 1111 1111 1111 1111 1111
1 1 0 0 1	0000 0001 1111 1111 1111 1111 1111 1111
1 1 0 1 0	0000 0011 1111 1111 1111 1111 1111 1111
1 1 0 1 1	0000 0111 1111 1111 1111 1111 1111 1111
1 1 1 0 0	0000 1111 1111 1111 1111 1111 1111 1111
1 1 1 0 1	0001 1111 1111 1111 1111 1111 1111 1111
1 1 1 1 0	0011 1111 1111 1111 1111 1111 1111 1111
1 1 1 1 1	0111 1111 1111 1111 1111 1111 1111 1111

[표 10]

이와 같이 " 0" 의 값 N은 32 올 " 0" 을 발생시킨다. 그러나 몇몇의 상황에서는 " 0" 의 값이 32개의 올 " 1" 을 발생시키는 것이 바람직하다. 이러한 함수는 상태 레지스터(210)의 " FMOD" 필드 또는 확장된 산술 논리 유닛 동작을 실행시 명령 비트 52, 54, 56 및 58에서 특정되는 " %!" 수정에 의해 선택된다. 이러한 함수는 마스크 발생기(239)에 의해 발생되는 마스크를 수정하거나, 혹은 모든 올 " 0" 이 공급되어도 C포트로 제공된 모든 올 " 0" 의 마스크가 동작하도록 산술 논리 유닛(230)의 함수를 수정함으로써 이행될 수 있다.

다른 실시가능한 마스크 기능의 유사한 수정이 가능하다. 이렇게 " %" 수정은 N=0인 경우, 올 " 0" 로 N개의 우측 자리맞춤 올 " 0" 을 갖는 마스크를 발생시키는 마스크 발생기(239)를 변화시킬 수 있다. 이와 유사하게, " %" 수정은 N=0인 경우, 모든 올 " 1" 로 N개의 좌측 자리맞춤 올 " 1" 을 발생시키는 마스크 발생기(239)를 변화시키거나, 모든 올 " 0" 으로 N개의 좌측 자리 맞춤 올 " 0" 을 발생시키는 마스크 발생기(239)를 변화시킬 수 있다.

다중 산술의 선택은 마스크 발생기(239)의 연산을 수정할 수 있다. 상태 레지스터의 " A사이즈" 필드가 " 110" 이면, 이것은 32 비트의 데이타 사이즈를 선택하고, 마스크 발생기(239)의 연산은 표 10에 도시한 것으로부터 변화되지 않는다. 상태 레지스터의 " A사이즈" 필드가 " 101" 이면, 이것은 16비트의 데이타 사이즈를 선택하고, 마스크 발생기(239)는 2개의 독립된 16비트 마스크를 형성한다. 이는 표 11에 도시되어 있다. 이러한 경우 마스크(239)로의 최상위 비트 입력은 무시된다. 표 11은 상기 비트를 돈케어 " X" 로서 나타내고 있다.

마스크 발생기 입력	마스크 - 반 워드 연산							
X 0 0 0 0	0000	0000	0000	0000	0000	0000	0000	0000
X 0 0 0 1	0000	0000	0000	0001	0000	0000	0000	0001
X 0 0 1 0	0000	0000	0000	0011	0000	0000	0000	0011
X 0 0 1 1	0000	0000	0000	0111	0000	0000	0000	0111
X 0 1 0 0	0000	0000	0000	1111	0000	0000	0000	1111
X 0 1 0 1	0000	0000	0001	1111	0000	0000	0001	1111
X 0 1 1 0	0000	0000	0011	1111	0000	0000	0011	1111
X 0 1 1 1	0000	0000	0111	1111	0000	0000	0111	1111
X 1 0 0 0	0000	0000	1111	1111	0000	0000	1111	1111
X 1 0 0 1	0000	0001	1111	1111	0000	0001	1111	1111
X 1 0 1 0	0000	0011	1111	1111	0000	0011	1111	1111
X 1 0 1 1	0000	0111	1111	1111	0000	0111	1111	1111
X 1 1 0 0	0000	1111	1111	1111	0000	1111	1111	1111
X 1 1 0 1	0001	1111	1111	1111	0001	1111	1111	1111
X 1 1 1 0	0011	1111	1111	1111	0011	1111	1111	1111
X 1 1 1 1	0111	1111	1111	1111	0111	1111	1111	1111

[표 11]

마스크 발생기(239)의 연산은 " 100" 의 " A사이즈" 필드를 통한 바이트 데이타의 선택을 위해 유사하게 수정된다. 마스크 발생기(239)는 그 입력의 단지 3개의 최하위 비트만을 사용하는 4개의 독립된 마스크를 형성한다. 이것이 표 12에 도시되어 있다.

마스크 발생기 입력	마스크 - 바이트 연산							
X X 0 0 0	0000	0000	0000	0000	0000	0000	0000	0000
X X 0 0 1	0000	0001	0000	0001	0000	0001	0000	0001
X X 0 1 0	0000	0011	0000	0011	0000	0011	0000	0011
X X 0 1 1	0000	0111	0000	0111	0000	0111	0000	0111
X X 1 0 0	0000	1111	0000	1111	0000	1111	0000	1111
X X 1 0 1	0001	1111	0001	1111	0001	1111	0001	1111
X X 1 1 0	0011	1111	0011	1111	0011	1111	0011	1111
X X 1 1 1	0111	1111	0111	1111	0111	1111	0111	1111

[표 12]

상술한 바와 같이, 4비트씩의 8개의 섹션, 2비트씩의 16개의 섹션, 1비트씩의 32개 섹션의 다중 연산을 지원하는 것이 가능하다.

본 기술 분야의 숙련된 자는 표 10, 11 및 12에 도시한 바와 같이, 이들이 다른 데이타 사이즈가 마스크 발생기(239)의 연산에 대한 유사한 수정을 필요로 함을 이해할 수 있을 것이다.

데이타 유닛(110)은 3개의 입력 산술 논리 유닛(230)을 포함한다. 산술 논리 유닛(230)은 3개의 입력 버스를 포함하는데, 즉 입력 A비트(241)는 A포트로 입력을 공급하고; 입력 B비트(242) - 는 B - 포트로 입력을 공급하며; 입력 C비트(243)는 C - 포트로 입력을 공급한다. 산술 논리 유닛(230)은 산술 논리 유닛의 목적지 버스(204)에 그 결과를 공급한다. 이러한 결과는 데이타 레지스터(200)의 데이타 레지스터 중 하나에 저장될 수 있다. 그밖에, 이 결과는 버퍼(108) 및 글로벌 포트 목적지 데이타 버스(GDST, 107)를 통해 디지탈 영상/그래픽 프로세서(71) 내의 다른 레지스터에 저장될 수 있다. 이러한 연산은 장거리 연산(long distance operation)이라 부른다. 이 명령은 그 결과의 목적지를 특정한다. 함수 신호 발생기(245)로부터 산술 논리 유닛(230)으로 공급되는 함수 코드 비트는 특정한 사이클 동안 산술 논리 유닛(230)에 의해 실행되는 특정한 3개의 입력 함수를 결정한다. 비트 0 캐리 - 인 발생기(246)는 산술 논리 유닛(230)의 제1 비트, 즉 비트 0으로 공급되는 캐리 - 인 신호를 형성한다. 전술한 바와 같이, 다중 산술 연산 동안 비트 0 캐리 - 인 발생기(246)는 다중 섹션의 각각의 최하의 비트로 캐리 - 인 신호를 공급한다.

제10도는 산술 논리 유닛(230)의 전형적인 비트 회로(400)의 구성을 블럭 다이어그램의 형태로 나타낸 것이다. 산술 논리 유닛(230)은 32 비트의 데이타 워드에 따라 동작하며 병렬로 32 비트 회로(400)를 구성한다. 산술 논리 유닛(230)의 각 비트 회로(400)는 3개의 입력 A_i, B_i 및 C_i의 대응 비트; 전술한 비트 회로(400)에서 C_{in0} 로 지정된 제로 캐리 - 인 신호, 전술한 비트 회로(400)에서 C_{in1} 으로 지정된 1캐리 - 인 신호; 산술 인에이블 신호 A_{en} ; 전술한 비트 회로에서의 역 킬 신호 $\overline{K}_{i-1}$;캐리 - 인 신호 C_{in0} 또는 C_{in1} 의 선택을 위한 캐리 감지 선택 신호; 및 8개의 역 함수 코드 비트F7-F0 를 수신한다. 제1 비트(비트0)용 캐리 - 인 신호 C_{in0} 및 C_{in1} 는 동일하며 전술한 특수 회로에 의해 발생된다. 입력 신호 A_i, B_i 및 C_i는 산술 논리 유닛(230)의 각 비트용으로 형성되기 때문에 다르다. 산술 인에이블 신호 Aen 및 역 함수 코드 비트F7-F0 는 32 비트 회로(400) 모두에 대해 동일하다. 산술 논리 유닛(230)의 각 비트 회로(400)는 대응하는 1비트 결과 S_i, 초기 제로 신호 Z_i; 다음 비트 회로용으로 제로 캐리 - 인 신호 C_{in0} 를 형성하는 C_{out0} 로 지정된 제로 캐리 - 아웃 신호; 다음 비트 회로용으로 1 캐리 - 인 신호 C_{in1} 을 형성하는 C_{out1} 로 지정된 1 캐리 - 아웃 신호 그리고 다음 비트 회로용으로 역 킬 신호 $\overline{K}_{i-1}$ 을 형성하는 역킬 신호 $\overline{K}_i$를 발생시킨다. 제로 캐리 - 아웃 신호 C_{out0} 또는 32 비트 산술 논리 유닛(230)에서 최종 비트의 1 캐리 - 아웃 신호 C_{out1} 중 선택된 것이, " C" 비트가 상기 명령에 대한 수정으로부터 보호되지 않으면 상태 레지스터(210)에 저장된다. 추가로, 다중 산술 동안, 명령은 분리된 산술 논리 유닛 섹션으로부터의 캐리 - 아웃 신호가 다중 플래그 레지스터(211)에 기억되어야 하는지를 특정한다. 이 경우에 선택된 제로 캐리 - 아웃 신호 C_{out0} 및 1 캐리 - 아웃 신호 C_{out0} 및 1 캐리 - 아웃 신호 C_{out1} 가 다중 플래그 레지스터(211)에 저장되게 된다.

비트 회로(400)는 결과 논리(401), 캐리-아웃 논리(402) 및 부울 함수 발생기(403)를 포함한다. 부울 함수 발생기(403)는 역 함수 코드 비트 F7-F0 에 따른 각 비트 입력 A_i, B_i및 C_i의 부울 조합을 형성한다. 부울 함수 발생기는 대응하는 전파 신호 P_i, 발생 신호 G_i및 킬 신호 K_i를 발생시킨다. 결과 논리(401)는 캐리 감지 선택 신호에 의해 선택되는 전술한 비트 회로(400)로부터 캐리-인 신호 C_{in0} 또는 캐리-인 신호 C_{in1} 중 하나와 전파 신호 C_{in1} 을 조합시켜 비트 결과 S_i및 초기 제로 신호 Z_i를 형성한다. 캐리-아웃 논리(402)는 전파 신호 P_i, 발생 신호 G_i, 킬 신호 K_i, 2개의 캐리-인 신호 C_{in1} 및 산술 인에이블 신호 A_{en} 를 수신한다. 캐리-아웃 논리(402)는 다음 비트 회로(400)에 공급되는 2개의 캐리-아웃 신호 C_{out0} 및 C_{out1} 을 발생시킨다.

제11도 및 제12도는 함께 산술 논리 유닛(230)의 전형적인 비트 회로(400)를 나타내고 있다. 제11도는 산술 논리 유닛(230)의 각 비트 회로(400)의 결과 논리(401) 및 캐리-아웃 논리(402)를 상세하게 나타내고 있다. 제12도는 산술 논리 유닛(230)의 각 비트 회로(400)의 결과 논리(401) 및 캐리-아웃 논리(402)를 상세하게 나타내고 있다. 제12도는 산술 논리 유닛(230)의 각 비트 회로(400)의 대응하는 부울 함수 발생기(403)를 상세하게 나타내고 있다.

각 결과 논리(401)는 대응하는 결과 신호 S_i및 초기 제로 신호 Z_i를 발생시킨다. 결과 논리(401)는 캐리 감지 선택 신호(carry sense select signal) 및 이전의 비트 회로로부터의 2 개의 캐리-인 신호, 역 전파 신호 $\overline{P}_i$, 역 킬 신호$\overline{K}_{i-1}$로부터 이들 신호를 형성한다. 캐리-아웃 논리(402)는 2개의 캐리-아웃 신호 및 역 킬 신호 $\overline{K}_i$를 형성한다. 이들 신호들은 상기 비트 회로(400)의 2개의 캐리-인 회로, 역 전파 신호 $\overline{P}_i$, 역 발생 신호$\overline{G}_i$ 및 킬 신호 K_i로부터 형성된다. 각 전파 신호는 "1" 캐리-인 신호가 비트 회로(400)를 통해 다음 비트 회로(400)로 전파되는지 또는 흡수되는지를 나타낸다. 발생 신호는 비트 회로(400)에 대한 입력이 다음 비트 회로(400)에 "1" 캐리-아웃 신호를 발생시키는지를 나타낸다. 킬 신호는 비트 회로(400)에 대한 입력이 다음 비트회로로 "0" 캐리-아웃 신호를 발생시키는지를 나타낸다. 전파 신호 P_i, 발생 신호 G_i및 킬 신호 K_i는 상호 배타적이다. 이들 신호 중 하나만이 각 입력의 조합을 위해 발생된다.

산술 논리 유닛(230)의 각 비트 회로(400)는 32 비트를 통해 캐리 리플 시간(carry ripple time)을 감소시키는 기술을 사용한다. 산술 논리 유닛(230)은 캐리 섹션을 8비트씩 4개의 섹션으로 분할한다. 상기 섹션 각각의 최하위 비트 회로(400)는 "0" 로 하드와이어 되는 제로 캐리-인 신호 C_{in0} 와 "1" 로 하드와이어 되는 1 캐리-인 신호 C_{in1} 을 갖는다. 각 비트 회로(400)는 다음 비트 회로에 2개의 결과 및 2개의 캐리-아웃 신호를 형성한다. 각 섹션을 통해 캐리 리플이 완료되면, 전술한 캐리 섹션의 최상위 비트로부터 출력된 실제 캐리가 캐리 감지 선택 신호를 형성한다. 한 섹션의 비트에 의해 발생되는 실제 결과의 선택은 멀티플렉서를 통해 캐리 선택 신호에 의해 가능하다. 제1 캐리 섹션은 후술하는 비트 0 캐리-인 발생기(246)로부터 캐리 선택 신호를 수신한다. 이러한 기술로 인해, 캐리 선택을 통한 캐리 리플이 동시에 발생되게 된다. 이로 인해 캐리 감지 선택 및 용장성 캐리 라인에 대한 몇몇의 추가적인 하드웨어의 비용으로 이 결과물을 얻는데 요구되는 시간 소모를 감소시킬 수 있다.

캐리-아웃 논리(402)는 캐리-인 신호의 아웃 신호의 수정을 제어한다. 캐리-아웃 논리(402)는 2개의 캐리-인 신호 C_{in0} 및 C_{in1} 에 따라 동작하는 동일한 회로를 포함한다. 인버터(412)에 의해 형성되는 반전파 신호 $\overline{P}_i$ 및 그 역, 전파 신호 P_i는 통과 게이트(413, 423)를 제어한다. 전파 신호 P_i가 "1" 이면 1 캐리-인 라인(410)은 통과 게이트(413)를 통해 1캐리-아웃 라인(411)에 접속되고, 제로 캐리-인 라인(420)은 통과 게이트(423)를 통해 제로 캐리-아웃 라인(421)에 접속된다. 이러한 방식으로 캐리-인 신호가 캐리-아웃으로 전파된다. 전파 신호 P_i가 "0" 이면, 1 캐리-인 라인(410)이 1캐리-아웃 라인(411)으로부터 고립되고, 제로 캐리-인 라인(420)이 캐리-아웃 라인(421)으로부터 고립된다. 발생 신호 G_i가 "1" 이면, 즉 역 발생 신호 G_i가 "0" 이면 P-채널 MOSFET(금속 산화물 반도체 전계 효과 트랜지스터)(414)가 캐리-아웃 라인(411)에 공급 전압을 제공하기 위해 턴온되고, P채널 MOSFET(424)가 공급 전압을 캐리-아웃 라인(421)에 제공하기 위해 턴온된다. 발생 신호 G_i가 "0" 이면, 즉 역 발생 신호 G_i가 "1" 이면 P채널 MOSFET(414, 424)가 공급 전압을 캐리-아웃 라인(421)에 제공하기 위해 턴온된다. 발생 신호 G_i가 "1" 이면, 즉 역 발생 신호 G_i가 "0" 이면 P-채널 MOSFET(금속 산화물 반도체 전계 효과 트랜지스터)(414)가 캐리-아웃 라인(411)에 공급 전압을 제공하기 위해 턴온되고, P채널 MOSFET(424)가 공급 전압을 캐리-아웃 라인(421)에 제공하기 위해 턴온된다. 발생 신호 G_i가 "0" 이면, 즉 역 발생 신호 $\overline{G}_i$가 "1" 이면 P채널 MOSFET(414, 424)가 차단되어 캐리-아웃라인(411, 421)에 영향을 주지 않는다. 킬 신호 K_i가 "1" 이면 N채널 MOSFET(415)가 접지와 캐리-아웃 라인(411)을 결합시키고, N채널 MOSFET(425)가 접지와 캐리-아웃 라인(421)을 결합시킨다. 킬신호 K_i가 "0" 이면 N-채널 MOSFET(415, 425)가 영향을 주지 않는다. 인버터(422)는 다음 비트 회로에 공급되는 역 킬 신호$\overline{K}_i$를 발생시킨다.

배타적 OR 회로(431, 433)는 결과 논리(401) 2개의 결과를 형성한다. 배타적 OR 회로(431, 433) 각각은 반전 입력(428)으로부터 전파 신호 P_i를 수신한다. 배타적 OR 회로(431, 433) 각각은 반전 입력을 통해 인버터(427)로부터 전파 신호 P_i와 비반전 입력을 통해 인버터(428)로부터 반전파 신호$\overline{P}_i$를 수신한다. 배타적 OR 회로는 비반전 입력을 통해 인버터(426)로부터 역 제로 캐리-인 신호 $\overline{C}_{in0}$를 수신하고, 현재의 캐리 섹션의 최하위 비트로 "0" 캐리-인의 경우에 대한 결과를 형성한다. 이와 유사하게, 배타적 OR 회로(433)는 비반전 입력을 통해 인버터(416)로부터 역의 1 캐리-인 신호$\overline{C}_{in1}$ 를 수신하고, 현재의 캐리 섹션의 최하위 비트로 "1" 캐리-인의 경우에 대한 결과를 형성한다. 인버터(422, 34)는 멀티플렉서(435)에 입력을 공급한다. 멀티플렉서(435)는 캐리 장치 선택 신호에 따라 이들 신호중 하나를 선택한다. 이 캐리 감지 선택 신호는 전술한 캐리 섹션의 최상위 비트로부터 실제 캐리-아웃 신호에 대응한다. 인버터(436)로부터 멀티플렉서(435)의 반전된 출력은 소망의 비트 출력 S_i이다.

결과 논리(401)는 또한 상기 비트 회로에 대한 초기 제로 신호 Z_i를 형성한다. 이 초기 제로 신호 Z_i는 상기 비트 회로(400)의 결과 S_i가 "0" 로 되는지의 초기 표시를 제공한다. 배타적 OR 회로(437)는 반전 입력을 통해 인버터(427)로부터 전파 신호 P_i를, 그리고 비반전 입력을 통해 인버터(428)로부터 역 전파 신호 P_i를 수신한다. 배타적 OR회로(437)는 또한 비 반전 입력을 통해 전술한 비트 회로(400)로부터 역킬 신호 $\overline{K}_{i-1}$ 를 수신한다. 배타적 OR 회로(437)는 전술한 비트킬 신호 K_{i-1} "0" 캐리-아웃 신호를 발생시키고, 전파 신호 P_i가 또한 "0" 인 경우에 대한 초기 제로 신호 Z_i를 형성한다. K_{i-1} 가 "0" 인 경우에 대한 초기 제로 신호 S_i를 형성한다. K_{i-1} 가 "0" 이면, 제로 캐리-아웃 신호 C_{out0} 및 1 캐리-아웃 신호 C_{out1} 가 캐리-인 신호 C_{in0} 및 C_{in1} 의 상태가 어떻든 "0" 이 된다. 이 초기 제로 신호 Z_i는 캐리가 캐리 섹션을 통해 리플하기 전에 얻을 수 있다. 따라서, 이 초기 제로 신호 Z_i는 산술 논리 유닛(230)으로부터 제3 출력의 결정을 신속하게 할 수 있다.

제12도에 도시한 산술 논리 유닛(230)의 각 비트 회로(400)의 부울 함수 발생기(403)는 비트 회로(400) 용의 전파 신호 P_i, 발생 신호 G_i그리고 킬 신호 K_i를 발생시킨다. 부울 함수 발생기(403)는 4개의 레벨로 구성되어 있다. 제1 레벨은 통과 게이트(451, 452, 453, 454, 455, 456, 457, 458)을 포함한다. 통과 게이트 (451, 453, 455, 457)는 제1 감지부에서 인버터(459)로부터 입력 C_i및 역입력$\overline{C}_i$에 의해 제어된다. 통과 게이트(452, 454, 456, 458)는 제1 감지부에서 입력 C_i및 역입력$\overline{C}_i$에 의해 제어된다. 입력 C_i의 상태에 따라, 통과 게이트(451, 453, 455, 457)가 도통되거나 혹은 통과 게이트(452, 454, 456, 458)가 도통된다. 제2 레벨은 통과 게이트(461, 462, 463, 464)를 포함한다. 통과 게이트(461, 463)는 제1 감지부에서 인버터(465)로부터 입력 B_i및 역입력$\overline{B}_i$에 의해 제어된다. 통과 게이트(462, 464)는 대향 감지부에서 제어된다. 입력 B_i의 상태에 따라, 통과 게이트(461, 463)가 도통되거나 통과 게이트(462, 464)가 도통된다. 제3 레벨은 통과 게이트(471, 472, 473)를 포함한다. 통과 게이트(471)는 제1 감지부에서 인버터 (473)로부터의 입력 A_i및 역 입력$\overline{A}_i$에 의해 제어된다. 통과 게이트(472, 473)는 대향 감지부에서 제어된다. 입력 A_i의 상태에 따라, 통과 게이트(471)가 도통되거나 통과 게이트(472, 473)가 도통된다. 제1 레벨은 대응하는 반전된 함수 코드 비트$\overline{F7}$-$\overline{F0}$ 에 결합된 인버터(441, 442, 443, 444, 445, 446, 447, 448)를 포함한다. 인버터(441, 442, 443, 444, 445, 446, 447, 448)는 부울 함수 발생기(403)에 입력 구동을 제공하고 산술 논리 유닛(230)에 의해 수행되는 논리 함수를 결정한다.

부울 함수 발생기(403)는 반전된 함수 코드 비트 $\overline{F7}$-$\overline{F0}$ 의 상태에 의해 선택된 함수와, 대응하는 입력 신호 A_i, B_i및 C_i에 따라 전파 신호 P_i를 형성한다. 인버터(476)에 대한 입력에서의 전파 신호 P_i는 통과 게이트(451, 452, 453, 454, 455, 456, 457, 458, 461, 462, 463, 464, 471 또는 472)를 통한 임의의 경로가 인버터(441, 442, 443, 444, 445, 446, 447 또는 448) 중 하나로부터 " 1" 을 결합시킬 때 " 1" 이 된다. 모든 다른 경우에 있어서는 이 전파 신호 P_i는 " 0" 이다. 인버터(476)는 제11도에 도시한 결과 논리(401)에 접속된 역 전파 신호 $\overline{P}_i$를 형성한다.

각 통과 게이트(451, 452, 453, 454, 455, 456, 457, 458, 461, 462, 463, 464, 471, 472, 473)는 병렬로 배치되고 N채널 MOSFET 및 P채널 MOSFET로 구성된다. N채널 MOSFET의 게이트는 제어 신호를 수신한다. 상기 전계 효과 트랜지스터는 그의 게이트 입력이 스위치 임계 전압 이상이면 도통된다. P채널 MOSFET의 게이트는 인버터(459, 465 또는 474) 중 하나를 통해 제어 신호의 역에 의해 구동된다. 이 전계 효과 트랜지스터는 그 게이트 입력이 스위치 임계값 미만이면 도통한다. P채널 MOSFET가 N채널 MOSFET의 동작과 역으로 동작하기 때문에 대응하는 인버터(459, 467 또는 474)는 이들 2개의 전계 효과 트랜지스터의 도통 또는 비 도통을 보장한다. 병렬로 구성된 N채널 및 P채널 전계 효과 트랜지스터는 제어된 입력의 극성이 어떠하든가에 요구될 때 도통을 보장한다.

3 상태 AND 회로(480)는 발생 신호 G_i및 킬 신호 K_i를 형성한다. 발생 신호 G_i, 킬 신호 K_i및 전파 신호 P_i는 양호한 실시예에서 서로 배타적이다. 따라서, 전파 신호 P_i는 3 상태 AND 회로(480)의 출력을 제어한다. 전파 신호 P_i가 " 1" 이면, 3상태 AND 회로(480)는 디스에이블되며, 발생 신호 G_i및 킬 신호 K_i는 " 0" 가 된다. 이와 같이 발생 신호 G_i 및 킬 신호 K_i둘다 캐리 신호를 수정하지 않는다. 통과 게이트(473)는 부울 함수 발생기(403)의 부분으로부터 3상태 AND 회로(480)의 하나의 입력으로 출력을 결합시킨다. 통과 게이트(473)의 게이트 입력은 제1 감지부에서 제1 입력 비트 A_i에 결합된다. N채널 MOSFET(475)는 조건부로 3상태 AND 회로(480)의 입력을 접지에 결합시킨다. 제1 입력 비트$\overline{A}_i$의 역은 N채널 MOSFET(475)에 게이트 입력을 공급한다. 통과 게이트(473)와 N 채널 MOSFET(475)는 와이어드 OR 관계로 결합되지만, 게이트 입력은 교호로 도전성을 가지므로 OR 연산이 일어나지 않는다. N 채널 MOSFET(475)는 A_i=" 0" 일 때 " 0" 입력이 3상태 AND 회로(480)로 입력되게 하는 역할을 한다. 산술 인에이블 신호는 3상태 AND 회로(480)에 제2 입력을 공급한다.

3상태 AND 게이트(480)는 다음과 같이 동작한다. 전파 신호 P_i가 " 1" 이면 P채널 MOSFET(481) 및 n채널 MOSFET(482)가 도통되고, 통과 게이트(483)는 비도통 상태로 된다. 이것은 P채널 MOSFET(414, 424) 그리고 N채널 MOSFET(415, 425)를 차단하여 이들 전계 효과 트랜지스터 모두를 도통시키지 않는다. 이와 같이 3상태 AND 회로(480)의 출력은 캐리-아웃 라인(411, 421)을 통해 신호를 수정시키지 않는 높은 임피던스 상태로 된다. 전파 신호 P_i가 " 0" 이면, P채널 MOSFET(481) 및 N채널 MOSFET(481)는 비도통 상태로 되고, 통과 게이트(483)는 도통 상태로 된다. 따라서, 회로는 2개의 입력의 논리 AND를 형성한다. N채널 MOSFET(475) 및 통과 게이트(473)의 결합점에서 산술 인에이블 또는 신호 중 하나가 " 0" , 또는 양자 모두 " 0" 이면 P채널 MOSFET(484) 또는 P채널 MOSFET(485) 중 적어도 하나가 공급 전압 V+(논리 " 1")을 역 발생 신호 $\overline{G}_i$로서, 캐리-아웃 논리(402)의 P채널 MOSFET(414,424)의 게이트에 접속시킨다. 이와 같이 P채널 MOSFET(414, 424)는 비도통 상태로 된다. 동시에 통과 게이트(483)는 도통 상태로 되고, 킬 신호 K_i와 같이 " 1" 신호를 캐리-아웃 논리(402)의 N채널 MOSFET(415, 425)의 게이트에 공급한다. 이것이 제로 캐리-아웃 신호 C_{out0} 를 " 0" 로 되게 하는 제로 캐리-아웃라인(421) 및 1 캐리-아웃 신호 C_{out1} 를 " 0" 로 되게 하는 1 캐리-아웃 라인(411)을 통해 신호를 하강시킨다. 입력 양자가 " 1" 이면, N채널 MOSFET(486) 및 N채널 MOSFET(487)의 직렬 결합이 N채널 MOSFET(415, 425)의 게이트에 접지(논리 " 0")를 제공한다. 캐리-아웃 논리(402)의 N채널 MOSFET(415, 425)는 차단 및 비토통 상태로 된다. 동시에, 통과 게이트(483)는 상기 " 0" 을 P채널 MOSFET(414, 424)의 게이트에 결합시킨다. 이로 인해, 캐리-아웃 논리(402)의 P채널 MOSFET(414, 424)가 도통 상태로 된다. 이것이 제로 캐리-아웃 신호 C_{out0} 를 " 1" 로 되게 하는 제로 캐리-아웃 라인(421) 그리고 1 캐리-아웃 신호 C_{out1} 을 " 1" 로 되게 하는 1 캐리-아웃 라인(411)을 통해 능동적으로 신호를 상승시킨다.

제10도 및 제21도에 도시한 비트 회로 구성은 전파항, 발생항, 결과항 그리고 2개의 캐리-아웃 항을 형성한다. 비트 회로(400)는 다음과 같이 전파항 Pi을 형성한다.

$$P_i = F0\&(\sim A_i\&\sim B_i\&\sim C_i) \mid F1\&(A_i\&\sim B_i\&\sim C_i) \mid F2\&(\sim A_i\&\ B_i\&\sim C_i)$$

$$\mid F3\&(A_i\&\ B_i\&\sim C_i) \quad \mid F4\&(\sim A_i\&\sim B_i\&\ C_i) \mid F5\&(A_i\&\sim B_i\&\ C_i)$$

$$\mid F6\&(\sim A_i\&\ B_i\&\ C_i) \quad \mid F7\&(A_i\&\ B_i\&\ C_i)$$

비트 회로(400)는 다음과 같이 발생항 G_i을 형성한다.

$$G_i = A_i\&[(F0\&\sim F1\&\sim B_i\&\sim C_i) \mid (F2\&\sim F3\&\ B_i\&\sim C_i)$$

$$\mid (F4\&\sim F5\&\sim B_i\&\ C_i) \mid (F6\&\sim F7\&\ B_i\&\ C_i)]$$

비트 회로(400)는 다음과 같이 킬항 K_i을 형성한다.

$$K_i = \sim G_i\&\sim P_i$$

비트 회로(400)는 다음과 같이 결과 항 S_i을 형성한다.

$$S_i = P_i\hat{}(C_{in0}\&\ CSS \mid C_{in1}\&\sim CSS)$$

여기서, CSS는 캐리 감지 선택 신호이다. 비트 회로(400)는 다음과 같이 2개의 캐리-아웃 신호 C_{out0} 및 C_{out1} 을 형성한다.

$$C_{out0} = (P_i \ \& \ C_{in0}) \mid (G_i \ \& \ A_{en}) \mid \sim(K_i \ \& \ A_{en})$$

$$C_{out1} = (P_i \ \& \ C_{in1}) \mid (G_i \ \& \ A_{en}) \mid \sim(K_i \ \& \ A_{en})$$

임의의 특정한 비트 i에 대하여 전파 신호 P_i, 발생 신호 C_i 및 킬 신호 K_i는 상호 배타적이다. 이들 2개의 신호는 동시에 발생되지 않는다.

각 비트 회로(400)의 구성은 산술 논리 유닛(230)이 반전된 함수 코드 비트 $\overline{F7}$-$\overline{F0}$ 에 따라 256개의 가능한 3입력 부울 함수 중 어느 하나 혹은 256개의 가능한 3개의 입력 혼합된 부울 및 산술 함수 중 어느 하나를 실행하는 것을 가능케 한다. 산술 인에이블 신호 및 반전된 함수 코드 비트 $\overline{F7}$-$\overline{F0}$ 를 포함하는 9개의 입력이 512개의 함수의 선택을 할 수 있게 한다. 이하 상세히 설명하는 바와 같이, 데이타 유닛(110)의 데이타 경로가 여러 가지 경로로 산술동작을 신속히 하도록 3개의 입력 산술 논리 유닛(230)의 사용을 유리하게 할 수 있다.

표 13은 하나의 함수 코드 $\overline{F7}$-$\overline{F0}$ 에 응답하는 비트 회로(400)의 간단한 부울 논리 함수를 나타내고 있다. 부울 논리 함수가 있고, 산술 인에이블 신호가 "0" 이므로 발생 및 킬 함수는 디스에이블된다. 부울 확장 산술 논리 유닛의 연산에 있어서는 전술한 바와 같이 비트 0 캐리-인 발생기(246)로부터 캐리-인 신호 C_{in0} 및 C_{in1} 을 특정하는 것이 가능하기 때문에 캐리 리플이 허용된다.

8-비트 ALU 코드 필드	함수 코드 비트	논리 연산
58	F7	A & B & C
57	F6	~A & B & C
56	F5	A & ~B & C
55	F4	~A & ~B & C
54	F3	A & B & ~C
53	F2	~A & B & ~C
52	F1	A & ~B & ~C
51	F0	~A & ~B & ~C

[표 13]

이들 연산들은 제11도 및 제12도를 참조함으로써 확인될 수 있다. F7="1" 이고 F6-F0 모두가 "0" 인 예에서 인버터(441, 442, 443, 444, 446, 447, 448)의 각각은 "0" 을 출력한다. 인버터(445)만이 "1" 출력을 발생시킨다. 전파 신호는 통과 게이트(455)를 턴온시키는 C_i="1", 통과 게이트(463)를 턴온시키는 B_i="1" 그리고 통과 게이트(472)를 턴온시키는 A_i="1" 인 경우에만 "1" 이다. 모든 다른 결합은 "0" 의 전파 신호에 귀결된다. 이것은 논리 연산이므로, 제로 캐리-인 신호 C_{in0} 및 1 캐리-인 신호 C_{in1} 양자는 "0" 이다. 이와 같이 배타적 OR 회로(431, 433) 양자가 전파 신호를 복귀시킴으로 S_i="1" 이다. 표 13의 다른 엔트리들도 유사하게 확인될 수 있다.

3개의 입력 A, B 및 C의 전체 256개의 부울 논리 함수는 함수 코드 비트 F7-F0를 적절히 선택함으로써 인에이블 된다. 3개의 입력의 상태표는 8개의 위치를 포함하기 때문에 3개의 입력으로 가능한 부울 논리 함수는 2^8=256이 된다. 2개의 입력 함수는 함수 코드 비트 F7-F0를 쌍으로 선택함으로써 달성되는 서브셋 함수이다. 입력 A에 관계없이 B 및 C의 부울 함수가 요구된다고 하면, F7=F6, P5=F4, F3=F2 및 F1=F0의 선택은 입력 A로부터 독립적인 것을 보장한다. 통과 게이트(471, 472)에 접속된 부울 함수 발생기(403)의 브랜치는 동일하게 구동된다. 이것은 A_i=" 1" 이거나 혹은 A_i=" 0" 이든 결과가 동일하다는 것을 보장한다. 이러한 선택은 입력 B 및 C의 모든 16개의 부울 논리 함수를 특정할 수 있는 4개의 제어 가능한 신호 쌍을 계속해서 제공한다. 2개의 입력의 상태표는 4개의 위치를 포함하므로, 4개의 입력의 가능한 부울 논리 함수는 2^4=16이 된다. 유사하게 F7=F5, F6=F4, F3=F1 및 F2=F0의 선택은 입력 B로부터 독립적임을 보장하며 입력 A 및 C의 16개의 부울 논리 함수를 특정하기 위한 4개의 제어가능한 신호 쌍을 제공한다. F7=F3, F6=F2, F5=F1 및 F4=F0의 선택은 입력 C와 관계없이 입력 A 및 B의 16개의 부울 논리 함수의 4개의 제어 가능한 신호 쌍을 통해 선택할 수 있다.

명령 워드는 산술 논리 유닛(230)에 의해 수행되는 함수, 즉 이러한 함수가 산술 또는 부울 논리인지를 결정한다. 표 13에 나타낸 바와 같이, 명령 워드는 부울 논리 연산용의 함수 코드 비트로 코드된 필드를 포함한다. 이 필드, 즉 명령 워드의 8비트 산술 논리 유닛 필드(비트 58-51)는 명령이 산술 논리 유닛(230)에 대한 부울 논리 연산을 특정할 때 함수 코드 비트로 직접 코드된다.

" 8비트 산술 논리 유닛" 필드는 명령이 산술 연산을 특정할 때 다르게 코드된다. 실행 가능한 산술 기능의 연구는 이들 산술 기능의 서브 세트가 가장 흔히 사용되는 연산임을 나타낸다. 함수 코드 비트 F7-F0 세트를 2위치의 헥사데시멀로서 표시하면, 이들 가장 흔히 사용되는 함수는 숫자 9,6 및 5만으로 보통 형성된다. 함수 코드 비트의 이들 세트에서는 F7= F6, F5= F4, F3= F2 및 F1= F0이다. 비트 57, 55, 53 및 51은 15개의 연산과 비산술 논리 유닛 연산의 특정한 경우를 위해 예약된 올 제로의 " 8비트 산술 논리 유닛" 필드를 특정한다. 이하, 비-산술 논리 유닛의 연산에 대해 설명한다. 산술 연산을 실행할 때 함수 코드 비트 F6=비트 57, 함수 코드 비트 F4=비트 55, 함수 코드 비트 F4=비트 53 및 함수 코드 비트 F2=비트 51이 된다. 다른 함수 코드 비트는 F7= F6, F5= F4, F3= F2 및 F1= F0로 설정된다. 이들 함수 및 이들의 대응하는 함수 코드 비트는 표 14에 도시되어 있다. 또 표 14는 디폴트 코딩에 대한 수정을 나타내고 있다.

8비트 ALU 코드필드 5555 7531	구동되는 함수 코드 비트 76543210	Hex	동작 설명
0000	10101010	AA	비산술 논리 유닛의 연산을 위해 예약
0001	10101001	A9	A-B 좌측 이동 "1"확장
0010	10100110	A6	A+B 좌측 이동 "0"확장
0011	10100101	A5	A-C
0100	10011010	9A	부호=0가 95로 플립되면 A-B 우측 이동 "1"확장 A-B 우측 이동 부호 확장
0101	10011001	99	A-B
0110	10010110	96	-@MF가 99 A-B로 플립되고, 부호=1 A+|B| 이면 C에 따라 A+B/A-B
0111	10010101	95	A-B 우측 이동 "0" 확장
1000	01101010	6A	A+B 우측 이동 "0" 확장
1001	01101001	69	~@MF가 66 A+B로 플립되고 부호=1 A-|B| 이면 A-B/A+B
1010	01100110	66	A+B
1011	01100101	65	부호=0가 6A로 플립되면 A+B 우측 이동 "1" 확장 A+B 우측 이동 부호 확장
1100	01011010	5A	A+C
1101	01011001	59	A-B 좌측 이동 "0" 확장
1110	01010110	56	A+B 좌측 이동 "1" 확장
1111	01100000	60	(A&C) + (B &C), 필드 A+B

[표 14]

명령 워드 비트(57, 55, 53, 51)의 여러개의 코딩은 표 14에 도시한 바와 같이 수정된 형태로 실행된다. 좌 또는 우측 이동을 나타내는 기능은 원통형 회전기(235) 및 마스크 발생기(238)와 연관하여 사용될 수 있다. 상술한 설명에서 " 부호" 는 산술 논리 유닛의 제2 입력 버스(206), 버스 구동 원통형 회전기(235)의 비트 31이다. 이것은 부호 번호의 부호 비트이다. 이 부호 비트에서 " 0" 는 정(+)의 수를 나타내고, 이 부호 비트에서 " 1" 은 부(2의 보수)수를 나타낸다. " 0100" 의 비트 57, 55, 53 및 51 상태는 우측 이동 " 1" 확장과 함께 A-B의 정규 함수로 된다. 산술 논리 유닛의 제2 입력 버스(206)의 비트 31의 " 1" 이면, 이 연산은 A+｜B｜(A 플러스 B의 절대값)으로 수정된다. " 1011" 의 비트 57, 55, 53 및 51 상태는 C의 비트 와이즈 상태의 C의 비트 라이즈 상태에 따라 A+B 또는 A-B의 정규 함수로 된다. 이 명령이 다중 플래그 레지스터 마스크 연산(@MF)을 특정하지 않는 연산 A-B로 수정된다. 산술 논리 유닛의 제2의 입력의 버스(206)의 비트 31이 " 1" 이면, 이 연산은 A+｜B｜(A 플러스 B의 절대값)으로 수정된다. " 1011" 의 비트 57, 55, 54 및 51 상태는 C의 비트 와이즈 상태에 따라 A+B 또는 A-B의 정규 함수로 된다. 이 명령이 다중 플래그 레지스터 마스크 연산(@MF)을 특정하지 않으면 연산은 A+B로 수정된다. 산술 논리 유닛의 제2의 입력 버스(206)의 비트 31가 " 1" 이면, 이 연산은 A-｜B｜(A 마이너스 B의 절대값)로 수정된다 " 1001" 의 비트 57, 55, 53 및 51 상태는 우측 이동 " 1" 확장과 함께 A+B의 정규 함수가 된다. 산술 논리 유닛의 제2의 입력 버스(206)의 비트 31가 " 0" 이면 이 연산은 우측 이동 부호 확장과 함께 A+B로 수정된다.

2개의 코드는 보다 유용한 연산을 제공하도록 수정된다. " 0000" 의 비트 57, 55, 53 및 51 상태는 후술하는 바와 같이 비-산술 논리 유닛의 연산은 지원하도록 예약되는 A(A가 아님)의 정규 함수가 된다.

" 1111" 의 비트 57, 55, 53 및 51 상태는 A의 정규 함수가 된다. 이것은 C의 상태에 의해 제어되는 A 및 B의 필드가산 또는 (A & C) + (B & C)로 수정된다.

표 14에 표시되어 있는 연산의 기본 세트는 산술 명령에 특정될 수 있다. 명령 워드 비트 58, 56, 54 및 52는 표 6에 제시되어 있는 이들의 기본적 연산의 수정된 연산을 제어한다. 이들 수정된 연산은 표 6 및 상태 레지스터(210)의 설명과 관련하여 상술되어 있다. 상술한 바와 같이 특정한 명령은 확장된 산술 논리 유닛을 특정한다. 확장된 산술 논리 유닛(EALU)의 동작을 통해 256개의 산술 연산의 각각을 특정하는 것이 여전히 가능하다. 이들 명령에 있어서, 데이타 레지스터 D0의 " A" (비트 27)는 산술 또는 부울 논리 연산 중 어느 하나를 특정하고, " EALU" 필드(비트 26-19)는 함수 코드 비트 F7-F0를 특정하며, " FMOD" 필드(비트 31-28)는 기본 함수의 수정을 특정한다. 또한 데이타 레지스터 D0의 " C" , " I" , " S" , " N" 및 " E" 필드는 산술 논리 유닛(230)의 비트 0으로, 즉 다중 산술이 인에이블 되는 경우, 각 섹션의 최하위 비트로 캐리-인의 제어를 허가한다. 4가지 형태의 확장된 산술 논리 유닛의 연산이 있다. 이들 중 2개는 승산기(220)를 사용하는 병렬 승산 연산을 특정한다. 확장된 산술 논리 유닛의 참(EALUT) 연산에 있어서 함수 코드 비트 F7-F0는 데이타 레지스터 D0의 " EALU" 필드의 대응 비트와 동일하다. 확장된 산술 논리 유닛의 거짓(EALUF) 연산에 있어서, 데이타 레지스터 D0의 " EALU" 필드의 개개의 비트는 함수 코드 비트 F7-F0를 형성하도록 반전된다. 확장된 산술 논리 유닛의 연산은 반전된 코드 비트가 유용한 관련 동작을 몇몇의 알고리즘 동안 수행하기 때문에 유용하다. 모든 함수 코드 비트를 반전시킨다는 것은 통상적으로 역 함수를 의미한다. 그리고, 이와 관련된 동작은 데이타 레지스터(208)를 재로딩시키지 않고 다른 명령을 통해 액세스 될 수 있다. 다른 확장된 산술 논리 유닛의 연산에 있어서, 함수 코드 비트 F7-F0는 데이타 레지스터 D0의 " EALU" 필드의 대응 비트와 동일하지만, 산술 논리 유닛(230)에 대한 데이타 경로는 다르게 인에이블된다. 이하에서 그 작용을 설명한다.

데이타 유닛(110)의 연산을 프로그램 흐름 제어 유닛(130)에 의해 추출되는 명령 워드에 응답한다. 명령 디코드 논리(250)는 오피코드 버스(133)를 통해 실행 파이프라인 스테이지에서의 명령에 대응하는 데이타를 수신한다. 명령 디코드 논리(250)는 수신된 명령 워드에 따라 멀티플렉서 Fmux(221), Imux(222), MSsux(225), Bmux(227), Amux(232), Cmux(233), Mmux(234) 및 Smux(231)의 연산용 제어 신호를 발생시킨다. 또한, 명령 디코드 논리(250)는 수신된 명령 워드에 따라 버퍼(104, 106, 108, 223, 236)의 동작을 제어한다. 이들 함수의 제어 라인은 설명을 간

단히 하기 위해 생략되었다. 멀티플랙서 및 버퍼의 특정한 제어 기능은 제18도와 관련한 명령 워드 포맷의 설명에 맞추어 이하 설명하도록 하겠다. 명령 디코드 논리(250)는 또한 산술 논리 유닛(230)의 제어를 위해 함수 신호 발생기(245) 및 비트 0 캐리-인 발생기(246)로 부분 디코드된 신호를 공급한다. 부분 디코딩에 대한 특정한 하드웨어에 대해서는 도시하지 않았지만, 본 발명에 숙련된 자는 제18도와 연관하여 명령 워드 포맷의 설명으로부터 이들 함수를 제공할 수 있다. 또한, 명령 디코드 논리(250)는 제7도와 관련하여 전술한 바와 같이 멀티플렉서(311, 312, 313, 314)의 제어에 의해 산술 논리 유닛(230)의 선택적인 다중 섹션 동작을 제어할 수 있다.

제13도는 함수 신호 선택기(245a)의 상세도이다. 함수 신호 선택기(245a)는 제5도에 도시한 함수 신호 발생기(245)의 일부를 형성한다. 함수 신호 발생의 전화면(full picture)에 있어서, 제13도는 제14도에 도시한 함수 신호 수정자(245b)로 간주되어야 한다. 멀티플렉서는 입력으로부터 출력으로 비트의 흐름을 나타내는 화살표를 가진 장방형으로 표시되어 있다. 입력은 소문자로 표시되어 있다. 제어 라인은 화살표에 수직한 멀티플렉서의 장방형으로 들어오는 그려진 대응하는 대문자로 표시되어 있다. 특정한 대문자로 표시된 제어 라인이 활성화될 때 대응하는 소문자를 가진 입력이 선택되어 멀티플렉서의 출력에 접속된다. 멀티플렉서OMUX(500)의 입력 " a" 는 두 부분에서 입력을 수신한다. 명령 워드의 비트(57, 55, 53, 51)는 입력 " a" 의 비트 라인 6, 4, 2 그리고 0에 각각 접속된다. 인버터(501)는 각 명령 워드 비트를 반전시키고 이들을 입력 " a" 의 비트 라인 7, 5, 3 및 1에 공급한다. 입력 " a" 은 제어 라인 " A" 이 활성화될 때 선택된다. 그리고, 선택되면 8개의 입력 비트 라인이 그들의 8개의 대응하는 수의 출력 비트 라인 7-4 및 3-0에 접속된다. 제어 라인 " A" 는 AND 게이트(502)에 의해 제공된다. AND 게이트(503)는 명령 분류 7-0 중 어느 것의 명령의 실행을 나타내는 제1 입력을 수신한다. 명령 워드 비트 63는 이것을 나타낸다. 이들 명령 분류는 이하 상세히 설명한다. AND 게이트(502)는 명령 워드의 비트 59에 의해 공급되는 제2 입력이다. 후술하는 바와 같이, " 1" 과 같은 비트 59에 산술 연산을 나타낸다. NAND 게이트(503)는 AND 게이트(502)에 제3 입력을 제공한다. NAND 게이트(503)는 4개의 명령 워드 비트(57, 55, 53 또는 51) 중 어느 것이 로우인지를 감지한다. 따라서, 제어 입력 " A" 은 명령 분류 7-0 중 어느 것이 선택될 때 활성화되는데, 즉 명령 워드의 산술 비트 59는 " 1" 이고, 명령 워드 비트(57, 55, 53, 51)는 모두 " 1" 이 아니다. 표 14로부터 " 1111" 의 비트(57, 55, 53,, 51) 상태는 본래 함수 신호라기보다 변형된 함수 신호 헥사 " 60" 로 되는 것을 알 수 있다.

멀티플렉서 OMUX(500)으로서 입력 " b" 는 상수 헥사 " 60" 이다. 멀티플렉서 OMUX(500)는 AND 게이트(504)가 제어 " B" 를 활성화하면 이러한 입력을 선택한다. AND 게이트(504)는 명령이 명령 워드 비트 63로서 표시한 바와 같이 분류 7-0 내에 있으면 제어 " B" 를 활성화시키며, 명령 워드 비트 59는 산술 연산을 나타내는 " 1" 이고, 비트(57, 55, 53, 51)의 상태는 " 1111" 이다. 표 14와 관련하여 전술한 바와 같이, 이를 조건 하에서는 함수 헥사 " 60" 이 명령으로 표시한 함수 신호로 대체된다.

멀티플렉서 OMUX(500)으로의 입력 " c" 는 모든 8개의 명령 워드 비트(58-51)를 수신한다. 멀티플렉서 OMUX(500)는 AND 게이트(505)가 제어 " C" 를 활성화하는 경우 이 입력을 선택한다. AND 게이트(505)는 인버터(506)를 통해 반전된 명령 워드 비트 59 및 명령 분류 7-0의 어느 것을 나타내는 것을 수신한다. 이와 같이 명령 워드 비트(58-51)는 명령 분류 7-0에서 256의 부울 연산 중 어느 것을 수행하도록 선택된다. 제어 입력 " D" , " E" , " F" , " G" 및 " H" 와 관련된 동작을 위한 명령 워드는 " 011" 과 동일한 비트 63-61을 갖는다. 이러한 조건이 일치하면 비트 60-57은 동작 형태를 정의한다.

멀티플렉서 OMUX(500)로의 입력 " d" 는 상수 헥사 " 66" 이다. 이 입력은 병렬 부호 승산 및 가산(MPYS‖ADD) 또는 병렬 비부호 승산 및 가산(MPYU‖ADD)를 실행하는 명령을 위해 선택된다. 이들 명령은 일반적으로 니모닉 MPYx ‖ADD라고 한다.

멀티플렉서 OMUX(500)으로의 입력 " e" 는 상수 헥사 " 99" 이다. 이 입력은 병렬 부호 승산 및 감산(MPYS ‖ SUB) 또는 병렬 비부호 승산 및 감산(MPYU ‖ SUB)를 실행하는 명령을 위해 선택된다. 이들 명령은 일반적으로 니모닉 MPYx ‖ SUB라고 한다.

멀티플렉서 OMUX(500)으로의 입력 " f" 는 상수 헥사 " A6" 이다. 이 입력은 DIVI 연산을 위해 선택된다. 제산에서 사용되는 이러한 DIVI 연산의 동작을 이하 상세히 설명하도록 하겠다.

멀티플렉서 OMUX(500)으로의 입력 " g" 는 비트 26-19로부터의 확장된 산술 논리 유닛의 함수 코드에 따라 데이타 레지스터 D0의 " EALU" 필드(비트 26-19)로부터 공급된다. 제어 입력 " G" 은 OR 게이트(507)가 MPYx ‖ EALUT 연산 또는 EALU 연산을 검출할 때 데이타 레지스터 D0로부터 " EALU" 필드를 선택하도록 활성화 상태로 된다. 전술한 바와 같이, EALUT에서 T 첨자는 EALUF에서 역(거짓)과 반대로 EALU 코트의 참을 특정한다. EALU 입력은 데이타 레지스터 D0의 " EALU" 필드가 EALU 또는 EALU %를 표시할 때 제어 입력 " G" 에 활성화된다.

인버터(508)는 멀티 플랙서 OMUX(500)의 입력 " h" 로 공급하기 위해 데이타 레지스터 D0의 " EALU" 필드의 개개의 비트를 반전시킨다. 멀티플렉서 OMUX(500)의 입력 " h" 은 제어 입력 " H" 에서의 MPYx ‖ EALUF 연산의 검출에 따라 선택된다. 전술한 바와 같이 EALUF의 F첨자는 레지스터 D0의 " EALU" 필드의 개개의 비트가 함수 코드 비트 F7-F0의 특정을 위해 반전된 것을 표시한다.

제13도에 표시한 멀티플렉서 AEmux(510)는 산술 인에이블 신호를 발생시킨다. 이 산술 인에이블 신호는 모든 비트 회로(400)의 3상태 AND 게이트(480)에 공급된다. 멀티플렉서 AEmux(510)로의 " a" 입력은 데이타 레지스터 D0의 " A" 비트(비트 27)이다. OR게이트(511)는 3개의 입력, 즉 MPYx ‖ EALUT, EALU, 및 MPYx ‖ EALUF를 수신한다. 명령이 이들 3개의 연산 중 어느 것을 선택하면, 멀티플렉서 AEmux로의 제어 입력 " A" 이 데이타 레지스터 D0의 " A" 비트(비트 27)를 선택한다. 멀티플렉서 AEmux(510)로의 " b" 입력은 명령 워드의 " ari" 비트(비트 59)이다. 후술하는 바와 같이, 이 " ari" 비트는 특정한 명령 형태를 위한 산술 연산을 선택한다. 이 입력은 명령이 분류 7-0 중 임의의 것이면 선택된다. 이러한 경우에 산술 연산(" ari" = " 1") 또는 부울 연산(" ari" = " 0")을 특정하는 " ari" 비트는 산술 논리 유닛(230)으로 직접 전달된다. 멀티플렉서 AEmux(510)의 " C" 입력은 상수 " 1" 이다. 게이트(512)는 명령이 확장된 산술 논리 유닛 명령 또는 명령 분류 7-0 내에 있지 않으면 이 입력을 선택한다. 이러한 명령은 DIVI 연산 및 MPYx ‖ ADD 및 MPYx ‖ SUB 연산을 포함한다. 이러한 명령은 DIVI 연산 및 DPYx ‖ ADD 및 MPYx ‖ SUB연산을 포함한다. OR 게이트(513)는 명령이 멀티플렉서 AEmux(510)의 출력에 의해 표시되는 산술 연산 또는 OR 게이트(511)에 의해 표시되는 " 임의의 EALU" 연산일때 산술 또는 EALU 신호를 제공한다.

제14도는 함수 신호 수정자(245b)를 나타낸다. 함수 신호 수정자(245b)는 데이타 레지스터 D0의 " FMOD" 필드에 따른 함수 신호 발생기(245a)로부터 설정된 함수 신호 또는 명령에 따른 명령 비트(58, 56, 54, 52)를 수정한다. 멀티플렉서 Fmux(520)는 함수 수정자 코드를 선택한다.

멀티플렉서 Fmux(520)로의 " a" 입력은 모두 올 " 0" (헥사" 0")이다. NOR 게이트(521)는 멀티플렉서 Fmux(520)의 제어 라인 " A" 을 공급한다. NOR 회로(521)는 제13도에 도시한 OR 게이트(511)로부터 " 임의의 EALU" 신호를 수신하는 제1 입력 및 AND 게이트(522)의 출력에 접속되는 제2 입력을 갖는다. AND 게이트(522)는 명령 워드의 " ari" 비트(비트 59)로 부터의 제1 입력 그리고 명령 분류 7-0의 명령을 나타내는 제2 입력을 수신한다. 이와 같이, NOR 게이트(521)는 명령이 임의의 확장된 산술 논리 유닛의 연산이 아니고, 명령 워드의 " ari" 비트가 " 0" 혹은 명령이 명령 분류 7-0 내에 있지 않을 Fmux(520)로의 헥사 " 0" 입력을 선택할 수 있는 활성 출력을 발생시킨다.

멀티플렉서 Fmux(520)로의 " b" 입력은 명령 워드의 비트 58, 56, 54 및 52를 수신한다. 제어 입력 " B" 는 AND 게이트(522)의 출력을 수신한다. 이와 같이 멀티플렉서 Fmux(520)는 명령이 임의의 명령 분류 7-0에 있거나 " ari" 의 명령 비트가 설정될 때 명령 워드의 비트 58, 56, 54 및 52를 선택한다.

멀티플렉서 Fmux(520)의 " C" 입력은 데이타 레지스터 D0의 " FMOD" 필드(비트 31-28)좌 비트를 수신한다. 제어 입력 " C" 은 OR 게이트(511)로부터 " 임의의 EALU" 신호를 수신한다. 멀티플렉서 Fmux(520)는 명령이 확장된 산술 논리 유닛의 연산을 요구하면 데이타 레지스터 D0의 " FMOD" 필드를 선택한다.

멀티플렉서 Fmux(520)는 활성 함수 수정 코드를 선택한다. 활성 함수 수정 코드는 이하 기술하는 바와 같이 산술 논리 유닛(230)으로 제공되는 함수 신호를 수정한다. 함수 수정 코드는 표 6에 명시되어 있는 동작을 제어하도록 디코드된다. 상술한 바와 같이, 이들 수정된 연산은 산술 논리 유닛(230)의 분할(splitting), 산술 논리 유닛(230)으로부터의 제로 또는 캐리-아웃에 의한 다중 플래그 레지스터(211)의 하나 또는 그 이상의 비트 설정, 다중 플래그 레지스터(211)의 회전 또는 클리어링, 4개의 모드 중 하나로의 LMO/RMO/LMBC/RMBC 회로(237)의 동작, 및 비트 0 캐리-인 발생기(246)의 동작 등을 포함한다. 함수 수정 코드의 특정 상태에 관련하여 수행되는 동작은 표 6에 제시되어 있다.

함수 수정자(245b) 내의 3개의 회로 블럭은 제13도에 도시한 멀티플렉서 OMUX(500)로부터의 함수 코드 비트 F7-F0을 수정할 수 있다. Mmux 블럭(530)은 C-포트로의 입력을 모드 올 " 1" 로 효과적으로 설정하도록 동작할 수 있다. A-포트 블럭(540)은 A-포트로의 입력을 모두 올 " 0" 으로 효과적 설정하도록 동작할 수 있다. 부호 확장 블럭(550)은 함수 코드 비트 F3-F0를 플립할 수 있는 부호 확장 유닛이다.

Mmux 블럭(530)은 수정 없이 함수 코드 비트 F3-F0를 정상적으로 전달하는 멀티플렉서(531)를 포함한다. 산술 논리 유닛(230)의 C-포트에 대한 입력을 올 " 1" 로 효과적으로 설정하기 위해서는 멀티플렉서(531)가 함수 코드 비트 F3-F0 상에 함수 코드 비트 F7-F4를 복제해야 한다. 멀티플렉서(531)는 AND 게이트(533)에 의해 제어된다. AND 게이트(533)는 다음의 3개의 조건 모두가 존재하면, 즉 1) 함수 수정자 코드 멀티플렉서 Fmux(520)가 " OX1X" 정합 검출기(532)(X=돈케어)로 검출된 4개의 코드 " 000" , " 1100" , " 0110" 또는 " 0111" 중 임의의 것이고; 2) 명령이 마스크 발생 동작을 요청하고; 그리고 3) 멀티플렉서 Mmux(234)로부터의 출력이 " 0" 이면 C-포트에 대한 입력을 모두 올 " 1" 로 효과적으로 설정하도록 활성화된다. 상술한 바와 같이, 함수 코드 비트 F3-F0 상에 함수 코드 비트 F7=F3, F6=F2, F5=F1 및 F4=F0의 선택은 입력 C에 관계없이 입력 A 및 B의 16개의 부울 논리 함수의 선택을 할 수 있게 한다. 표 6으로부터 4개의 함수 수정자 코드 " OXIX" 는 " %!" 수정을 포함한다. 제14도에 따르면, " %!" 수정은 마스크 발생기(239)에 의해 발생되는 마스크를 수정하는 것에 의하지 않고 산술 논리 유닛(230)으로 송출된 함수 코드 비트를 수정함으로써 달성된다.

A-포트 블럭(540)은 수정 없이 정상적으로 함수 코드 비트 F7-F0를 전달하는 멀티플렉서(541) 및 접속 회로(542)를 포함한다. 산술 논리 유닛(230)의 A-포트에 대한 입력을 올 " 0" 로 효과적으로 설정하기 위해서는 멀티플렉서(541) 및 접속회로(541)가 함수 코드 비트 F7, F5, F3, F1 각각에 함수 코드 비트 F6, F4, F2 및 F0을 복제해야 한다. 멀티플렉서(541) 및 접속회로(542)는 OR 게이트(544)에 의해 활성화될 때 이러한 대체를 할 수 있다. OR 게이트(544)는 " O10X" 정합 검출기(543)에 접속된 제1 입력 및 AND 게이트(546)에 접속된 제2 입력을 갖는다. AND 게이트(546)는 " 011X" 정합 검출기(545)에 접속된 제1 입력을 갖는다. 정합 검출기(543, 545)는 함수 수정자 코드가 그들의 검출 상태와 일치하는지를 결정한다. AND 게이트(546)는 명령이 마스크 발생 동작을 요구하는지를 나타내는 신호를 수신하는 제2 입력을 갖는다. 산술 논리 유닛(230)의 A-포트로의 입력은 함수 코드 비트 F7, F5, F3 및 F1 각각 대신에 함수 코드 비트 F6, F4, F2 및 F0로 교환함으로써 효과적으로 제로(0)로 될 수 있다. 전술한 바와 같이, 이러한 대체가 A입력에 관계없이 산술 논리 유닛(230)의 출력을 만든다. 이러한 대체는, 1) " 010X" 정합 검출기(543)에서 함수 수정자 코드가 일치하는 것으로 밝혀질 때; 또는 2) 명령이 마스크 발생 동작을 요구하고, 함수 수정자 코드가 " 010X" 가 정합 검출기(545)에서 일치하는 것으로 밝혀지고 명령이 마스크 발생 동작을 요구할 때 발생된다.

부호 확장 블럭(550)은 통상 수정없이 함수 코드 비트 F3-F0를 전달하는 배타적 OR 게이트(551)를 포함한다. 그러

나, 이들 함수 코드 비트 F3-F0는 산술 논리 유닛 부호 확장 및 특정한 조건 하에서의 절대값의 목적을 위해 반전된다. A-포트 블럭(540)으로부터의 함수 코드 비트 F7-F4는 부호 확장 블럭(550)에 의해 항상 수정 없이 전달된다. AND 게이트(552)는 배타적 OR 게이트(551)가 함수 코드 비트 F3-F0를 반전시키는지를 제어한다. AND 게이트(552)는 제13도에 도시한 OR 게이트(513)로부터 산술 또는 확장된 산술 논리 유닛의 신호를 수신하는 제1 입력을 갖는다. AND 게이트(552)로의 제2 입력은 멀티플렉서(553)로부터 나온다.

멀티플렉서(553)는 제13도의 OR게이트(511)로부터의 " 임의의 EALU" 신호에 의해 제어된다. 멀티플렉서(553)는 " 임의의 EALU" 신호가 활성화될 때 AND 게이트(554)로부터 제1 신호를 선택하고, " 임의의 EALU" 신호가 비활성화 상태일 때 혼합 AND/OR 게이트(556)로부터 제2 신호를 선택한다. AND 게이트(554)의 출력은 인버터(555)에 의해 반전되는 부호 비트(비트 31)로 표시한 바와 같이 산술 논리 유닛의 제2 입력 버스(206) 상의 데이타가 정(+)일 때 " 1" 로 되어 데이타 레지스터 D0의 " S" 비트(비트 16)가 " 1" 이 된다. 혼합 AND/OR 게이트(556)의 출력은 다음의 경우에 활성화된다. 즉, 1) 인버터(555)에 의해 반전되는 부호 비트(비트 31)로 표시한 바와 같이, 산술 논리 유닛의 제2 입력 버스(206) 상의 데이타가 정(+)이고, 2) 명령이 명령 분류 7-0 내에 있고; 그리고 3) a) " 0100" /" 0100" 정합 검출기(557)에서 명령 비트 57, 55, 53, 및 51이 일치하는 것으로 밝혀지거나, b) AND 게이트(560)가 " 1001" /" 0110" 정합 검출기에서 명령 비트 57, 55, 53 및 51이 일치하는 것을 검출하고, 인버터(559)로 표시한 바와 같이 명령이 다중 플래그 레지스터 마스크 동작(@MF)을 요구하지 않을 경우이다.

부호 확장 블럭(550)은 표 14에 표시한 예외를 수행한다. 명령이 산술 동작을 특정했던 것을 표시하는 비활성 " 임의의 EALU" 신호는 멀티플렉서(553)에 대한 제2 입력을 선택한다. 혼합 AND/OR 게이트(556)는 명령이 명령 분류 7-0 내에 있는지 그리고 부호 비트가 " 0" 인지를 결정한다. 이들 조건 하에서, 명령 워드 비트 57, 55, 53 및 51이 " 0100" 이면 함수 코드는 함수 코드 비트 F3-F0를 반전시킴에 의해 헥사 " 9a" 로부터 헥사 " 95" 로 플립된다. 이와 유사하게, 명령 워드 비트 57, 55, 53 및 51이 " 1011" 이면, 함수 코드는 함수 코드 비트 F3-F0를 반전시킴에 의해 헥사 " 65" 로부터 헥사 " 6a" 로 플립된다. 명령 워드 비트 57, 55, 53 및 51가 " 1001" 이고 명령이 인버터(599)로 표시한 바와 같이 다중 플래그 레지스터 마스크 동작을 요구하지 않으면, 함수 코드는 헥사 " 69" 로부터 헥사 " 66" 로 플립된다. 이러한 함수 코드 비트의 세트는 산술 논리 유닛(230)으로 하여금 A-｜B｜, 즉 A 마이너스 B의 절대값을 수행하도록 한다. 명령 워드 비트 57, 55, 53 및 51이 " 0110" 이고, 명령이 다중 플래그 레지스터 마크 동작을 요구하지 않으면 함수 코드는 헥사 " 96" 으로부터 헥사 " 99" 로 플립된다. 이것은 함수 A+｜B｜, 즉 A 플러그 B의 절대값을 실행한다. 함수 코드의 이들 플립은 산술 논리 유닛의 제2의 입력 버스(206) 상의 데이타의 부호 비트(비트 31)에 따른다.

제15도는 비트 0 캐리-인 발생기(246)를 나타낸다. 전술한 비트와 같이, 0 캐리-인 발생기(246)는 산술 논리 유닛(230)의 제1 비트에 공급되는 캐리-인 신호 C_{in} 를 발생시킨다. 아울러, 비트 0 캐리-인 발생기(246)로부터의 캐리-인 신호 C_{in} 는, 명령이 다중 산술 논리 유닛 동작을 요청하면, 다중 섹션의 각각의 제1 비트로 공급된다. 멀티플렉서 Xmux(570)는 명령 디코드 논리(250)로부터의 6개의 대응 제어 입력에 따라 비트 0 캐리-인 신호 C_{in} 용의 6개의 가능한 발생지 중 하나를 선택한다.

멀티플렉서 Zmux(570)의 " 입력 " a" 에는 다중 플래그 레지스터(211)의 비트 31 가 제공된다. 멀티플렉서 Zmux(570)는 명령이 DIVI 동작을 요청하면, 이 입력을 비트 0 캐리-인 신호 Cin으로서 선택한다.

멀티플렉서 Zmux(570)에 대한 " b" , " c" 및 " d" 는 혼합 논리 함수로 형성된다. 멀티플렉서 Zmux(570)의 입력 " b" 는 함수 코드 비트 F6, F2 및 F0의 부울 함수인 신호를 수신한다. 회로(571)에 의해 형성되는 부울 식은 (F0 & F6)｜(F0 & F2)｜(F2 & F6)이다. 멀티플렉서 Zmux(570)의 입력 " C" 은 배타적 OR 게이트(573)에 의해 공급되는 제1 입력과 AND 게이트(574)에 의해 공급되는 제2 입력을 가진 배타적 OR 게이트(572)에 의해 공급된다. 배타적 OR 게이트(573)는 산술 논리 유닛(230)의 앞선 동작이 비트 31, 즉 최종 비트의 캐리-아웃 신호 C_{out} 를 발생시 켰는지를 나타내는 데이타 레지스터 D0의 제1의 입력 " C" 비트(비트 18)을 갖는다. XOR 게이트(573)의 제2 입력은 MPYx‖EALUF 동작에 대한 명령 요구를 나타내는 신호를 수신한다. AND 게이트(574)는 정(+)의 부호를 검출하기 위해 산술 논리 유닛의 제2 입력 버스(206) 상에 존재하는 부호 비트(비트 31)를 반전하는 인버터(575)로부터의 제1 입력을 갖는다. AND 게이트(574)는 데이타 레지스터 D0의 " 1" 비트(비트 17)로 부터의 제2 입력 및 데이타 레지스터 D의 " S" 비트(비트 16)으로부터의 제3 입력을 갖는다. 상술한 바와 같이, " I" 비트는 " S" 비트가 부호 확장이 인에이블되는 것을 표시할 때 캐리-인의 반전에 기인한 것이다. 이러한 동작은 제14도에 도시한 함수 수정자(246b)의 AND 게이트(554) 및 XOR 게이트(551)의 부호 확장 동작을 보충하여 완전하게 한다. 멀티플렉서 Zmux(570)의 입력 " d" 은 XOR 게이트(576)로부터 나온다. XOR 게이트(576)는 함수 코드 비트 F0에 공급되는 제1 입력 및 입력 C버스(243) 상의 데이타의 비트 0에 공급되는 제2 입력을 갖는다.

멀티플렉서 Zmux(570)의 입력 " b" 은 AND 게이트(581)가 제어 입력 " B" 를 활성 상태로 설정할 때 선택된다. 이것은 OR 게이트(513)로부터 " 산술 또는 EALU" 가 활성화될 때 발생하므로, 명령은 인버터(582)로 표시한 바와 같이 확장된 산술 논리 유닛의 연산을 요구하지 않으며, 어떤 다른 멀티플렉서 Zmus(570)의 입력도 인버터(583, 584, 585)에 의해 제어되는 것과 같이 적용되지 않는다.

멀티플렉서 Zmuz(570)의 입력 " c" 는 AND 게이트(586)가 제어 입력 " c" 으로 활성 출력을 공급할 때 선택된다. AND 게이트(586)는 " 임의의 EALU" 동작에 대한 명령 요구를 나타내는 신호에 응답한다. AND 게이트(586)로의 입력의 나머지는 입력 " d" , " e" 또는 " f" 중 어느 것이 인버터(584, 585, 586)를 통해 활성화될 때 AND 게이트(586)가 활성화되지 않도록 한다.

멀티플렉서 Zmux(570)의 입력 " d" 는 제어 라인 " D" 이 AND 게이트(587)로부터 있을때 선택된다. AND 게이트(587)는 명령이 산술 동작 또는 확장된 산술 논리 유닛의 연산일 때 활성화되며, AND 게이트(589)는 활성화되고 입력 " e" 는 인버터(585)에 의해 표시한 바와 같이 선택되지 않는다. AND 게이트(589)는 명령이 다중 플래그 레지스터 마스크 연산(@MF)의 확장을 특정하고 명령 워드 비트 57, 55, 53 및 51가 " 0110" /" 1001" 정합 회로(588)에서 일치할 때 활성화된다. 이들 명령 워드 비트는 포트 C에 대한 입력에 따르는 포트 A와 B간의 가산 또는 감산에 기인하는 함수 코드 헥사 " 69" 및 헥사 " 96" 에 대응한다. 명령 분류가 다중 플래그 레지스터 확장을 포함함으로써 어떤 함수 코드 플립핑도 포함되어 있지 않다. 제7도는 다중 모드에서 분할 산술 논리 유닛의 복수의 섹션에 캐리-인 신호를 제공하는 것을 나타낸다.

멀티플렉서 Zmux(570)의 입력 " e" 는 상태 레지스터(210)의 " C" 비트(비트 30)로부터 나온다. 전술한 바와 같이, 상태 레지스터(210)의 " C" 비트는 산술 논리 유닛(230)의 최종 연산의 결과가 비트 31로부터의 캐리-아웃에 기인하는 경우, " 1" 로 설정된다. AND 게이트(594)는 제어 입력 " E" 를 공급한다. AND 게이트(594)는 명령이 산술 연산 또는 확장된 산술 논리 유닛 연산을 특정하고, 다음 논리가 참일 때, 즉 1) " OX01" 정합 검출기(591)에서 함수 수정자 코드가 일치하고; 또는 (OR 게이트(590)) 2) 명령이 마스크 발생 연산을 요구하고, (AND 게이트(593))" OX11" 정합 검출기(592)에서 함수 수정자 코드가 일치하면 활성화된다.

멀티플렉서 Zmus(570)의 입력 " f" 에는 상수 " 0" 가 공급된다. 멀티플렉서 Zmux(570)는 OR 게이트(513)로부터의 " 산술 또는 EALU" 신호가 명령이 인버터(595)에 의해 반전되는 부울 연산을 특정하는 것을 나타낼 때 상기 입력을 선택한다.

Zmux(570)의 출력은 정상적으로 Ymux(580)를 통해 수정 없이 전달되어 비트 0 캐리-인 출력에서 나타난다. 데이타 레지스터 D0 " A" 비트(비트27) 및 " E" 비트(비트 14)가 둘다 " 1" 이 아닌 다중 산술 연산에서, Ymux는 복수의 동일한 캐리-인 신호를 발생시킨다. 상태 레지스터(210)의 " A사이즈" 필드를 통해 반 워드 연산을 선택함으로써 Ymux가 비트 0 캐리-인 출력 및 비트 16 캐리-인 출력에 Zmux(570)의 출력을 발생시킨다. 이와 유사하게, 바이트 연산의 선택 시 Ymux(580)는 비트 0 캐리-인 출력 비트 8 캐리-인 출력, 비트 16 캐리-인 출력 및 비트 24 캐리-인 출력에 Zmus(570)의 출력을 공급한다.

Ymux(580)의 연산은 데이타- 레지스터 D0의 " A" 비트(비트 27) 및 " E" 비트(비트 14)가 모두 " 1" 일 때 다르게 된다. AND 게이트(577)는 이러한 조건을 형성하며 Ymux의 연산을 제어한다.

이것은 캐리-인 신호가 다중 산술이 다르게 되는 동안, 산술 논리 유닛(230)의 다른 섹션으로 공급되는 유일한 경우이다. AND 게이트(577)가 이런 조건을 검출하면, 캐리-인 신호는 함수 코드 비트 F0의 배타적 OR 및 산술 논리 유닛(230)의 대응 섹션의 C 입력의 최하위 비트에 의해 형성된다. " A사이즈" 필드가 워드 연산을 선택하면, 즉 산술 논리 유닛(230)이 단일의 32 비트 섹션을 형성하면, Ymux(580)에 의해 형성된 비트 0 캐리-인 출력은 XOR 게이트(596)에 의해 형성되는 입력 C 버스 비트 0와 함수 코드 비트 F0의 배타적 OR이다. 어떤 다른 캐리-인 신호도 형성되지 않는다. " A사이즈" 필드가 2개의 16비트 섹션을 형성하는 반 워드 연산을 선택하면, Ymux(580)에 의해 형성되는 비트 0 캐리-인 출력이 XOR 게이트(596)의 출력이 되고, 비트 16에 대한 캐리-인은 XOR 게이트(598)에 의해 형성되는 입력 C 버스 비트 16과 함수 코드 비트 F0의 배타적 OR이다. 바이트 다중 산술에 있어서는, Ymux(580)에 의해 형성된 비트 0 캐리-인 출력이 XOR 게이트(596)의 출력이고, 비트 8 캐리-인은 XOR 게이트(597)에 의해 형성되고, 비트 16 캐리-인은 XOR 게이트(598)에 의해 형성되며, 비트 24 캐리-인은 XOR 게이트(599)에 의해 형성된다.

제13도, 14도 및 15도는 표를 실행하는 특정 블럭을 나타낼 뿐만 아니라, 특정 회로망의 블럭 다이어그램이 간결성이 결여된 경우에도 숙련된 자가 양호한 실시예를 구성할 수 있도록 표 및 도면이 논리 회로망을 간략하게 정의하는 직선적 전송 처리를 나타내고 있다. 제13도 및 제14도의 회로는 제5도에 도시한 데이타 유닛(110)의 일부인 명령 디코드 논리(250)를 통한 여러 가지 멀티플렉서 및 특수 회로를 위한 제어는 포함하고 있지 않다. 그러나, 이들 회로의 제어는 간단하며, 본 기술 분야에 통상의 지식을 가진 자의 역량에 달려 있다. 따라서, 이에 대해서는 설명을 간단히 하기 위해 추가의 설명은 하지 않는다.

산술 논리 유닛(230)은 각 입력으로 향하는 다른 하드웨어 기능을 가진 3개의 32 비트 입력을 포함한다. 따라서, 각 입력을 공급하는 하드웨어로부터의 결과를 결합하도록 산술 논리 유닛(230)을 사용하여 많은 다른 기능을 실행할 수 있다. 산술 논리 유닛(230)은 부을 또는 비트x 비트의 논리적 조합(bit by bit logical combination), 산술 조합(arithmetic combination) 및 3개의 입력의 혼합된 부울 및 산술 조합(mixed Boolean and arithmetic combination of 3 inputs)을 수행할 수 있다. 이후, 혼합된 부울 및 산술 함수는 실행의 간단화를 위해 산술 연산이라 한다. 산술 논리 유닛(230)은 부울 함수 또는 산술 함수 중 어느 하나를 선택하는 하나의 제어 비트를 갖는다. 부울 함수는 산술 논리 유닛(230)의 비트 회로(400)로부터 또는 비트 회로(400) 간에 어떤 캐리도 발생시키지 않는다. 따라서, 산술 논리 유닛(230)의 각 비트 회로(400)는 32개의 개별적인 비트 와이즈 결과를 독립적으로 형성하는 비트 회로에 3개의 입력을 결합시킨다. 산술 연산 동안, 각 비트 회로(400)는 인접한 하위 비트로부터 캐리-인을 수신하고 다음 최상위 배트 위치에 캐리-아웃을 발생시킨다. 8비트 제어 신호(함수 코드 비트 F7-F0)는 산술 논리 유닛(230)에 의해 수행되는 기능을 제어한다. 이로 인해, 256 부울 함수 중 하나 및 256 산술 함수 중 하나를 선택할 수 있다. 함수 코드 비트 F7-F0의 함수 신호 넘버링(numbering)은 마이크로 소프트(사)의 윈도우에 사용되는 것과 동일하다. 비트 0 캐리-인 발생기(246)는 산술 모드에 있을 때 캐리-인 신호를 공급한다. 산술 모드에서, 산술 논리 유닛(230)은 보다 작은 데이타 섹션을 병렬 다중 처리하기 위해 2개의 독립된 16비트 섹션 또는 4개의 독립된 8비트 섹션으로 분할될 수 있다. 비트 0 캐리-인 발생기(246)는 산술 논리 유닛(230)이 하나, 둘 또는 4개의 섹션 각각으로 동작할 때 하나, 둘 또는 4개의 캐리-인 신호를 공급한다. 양호한 실시 예에서, 데이타 유닛(110)용 어셈블러는 대수식 입력 구문에 따른 함수

코드 비트의 적절한 세트를 선택하는 식 평가기(expression evaluator)를 포함한다.

실행되는 특정한 명령은 산술 논리 유닛(230)의 함수를 결정한다. 이하 설명하는 바와 같이, 양호한 실시예에서, 명령 워드는 부울 또는 산술 연산 중 하나를 표시하는 필드를 포함한다. 다른 명령 워드 필드는 산술 논리 유닛(230)에 공급되는 함수 코드 비트를 특정한다. 산술 명령에 있어서, 이들 명령 워드 필드의 제1 서브셋은 표 14에 따라 가능한 산술 논리 유닛의 연산의 서브셋을 특정한다. 이 명령 워드 필드의 제2 서브셋은 표 6에 따라 명령 함수의 수정을 특정한다. 함수 코드 비트의 모든 가능한 수정 및 부울 및 산술 명령에 대한 함수 수정은 확장된 산술 논리 유닛(EALU) 명령에 의해 특정될 수 있다. 이 경우에, 제9도에 도시한 데이타 레지스터 D0 내의 미리 정의된 필드는 산술 논리 유닛(230)의 연산을 특정한다.

산술 논리 유닛(230)이 모든 3개의 입력을 결합할 수 있지만, 많은 유용한 함수는 몇몇의 입력을 포함하고 있지 않는다. 예를 들어, 식 A& B는 돈케어로서 C입력을 처리하며, 식 A | C는 돈케어로서 B 입력을 처리한다. 다른 데이타 경로 하드웨어가 각 입력보다 선행하기 때문에 임의의 입력을 사용 또는 무시할 수 있는 능력이 소망의 기능에 필요한 데이타 경로 하드웨어의 선택을 지원한다. 표 15는 C-입력이 마스크 또는 합병(merging) 제어로서 다루어지는 유용한 3개의 입력 식의 예를 나타낸다. 데이타 유닛(110)이 산술 논리-유닛(230)의 C-입력의 데이타 경로에서 확장 회로(238) 및 마스크 발생기(239)를 포함하기 때문에 마스크로서 C-입력을 사용하는 것은 당연한 것이다.

논리적 함수	일반적 사용
(A&C) | (B&~C)	C에 따라 A 및 B의 비트 x 비트 멀티플렉싱(합병) C에 대응 비트가 1이면 A 선택.
(A&~C) | (B&C)	C에 따라 A 및 B의 비트 x 비트 멀티플렉싱(합병) C의 대응 비트가 1이면 B 선택.
(A | B)&~C	A 및 B의 논리적 OR 그리고 C가 1이면 0로
(A&B)&~C	A 및 B의 논리적 AND 그리고 C가 1이면 0로
A | (B&C)	C가 0이면 논리가 A와 OR되기 전에 B입력을 0으로
A | (B | ~C)	C가 0이면 논리가 A와 OR되기 전에 B입력을 1로

[표 15]

3개의 입력 산술 논리 유닛(230)은 산술 논리 유닛(230)을 관통하는 단일 패스(single pass)에서 혼합된 부울 및 산술 연산을 수행할 수 있다. 혼합된 부울 및 산술 함수는 산술 기능에 앞서 부울 연산의 실행을 지원한다. 이동과 가산, 이동과 감산 또는 가산 또는 감산에 앞선 필드 마스킹과 같은 다양한 혼합 기능을 다른 데이타 경로 하드웨어와 결합된 적절한 산술 논리 유닛의 기능에 의해 수행할 수 있다. 산술 논리 유닛(230)은 256개의 다른 산술 함수를 지원하지만, 이들 중 서브셋만이 대개 프로그래밍을 위해 필요로 된다. 아울러, 캐리-인 및 부호 확장과 같은 추가의 선택이 제어될 필요가 있다. 흔히 사용되는 몇몇의 예가 아래 표 16에 나열되어 있다.

[표 16]

디콜드 캐리-인 제어 및 부호 확장 선택과 조합된 산술 함수의 가장 일반적으로 유용한 세트는 연산의 기본 세트 중 명령 세트에서 직접 얻을 수 있다. 상기 기본 세트는 부호 비트에 따라 산수 논리 유닛의 기능적 제어를 수정하고, 그리고 디폴트 캐리-인 선택을 사용하는 연산을 포함한다. 이하, 이들 중 몇몇의 예에 대하여 상세히 설명한다.

명시 캐리-인 및 부호 확장 제어와 함께 모든 256개의 산술 함수는 확장된 산술 논리 유닛(EALU)명령을 통해 얻어질 수 있다. 상기 기본 세트는 부호 비트에 따라 산술 논리 유닛의 기능적 제어를 수정하고, 그리고 디폴트 캐리-인 선택을 사용하는 동작을 포함한다. 이하, 이들 중 몇몇의 예에 대하여 상세히 설명한다.

명시 캐리-인 및 부호 확장 제어와 함께 모든 256개의 산술 함수는 확장된 산술 논리 유닛(EALU) 명령을 통해 얻어질 수 있다. 확장된 산술 논리 유닛의 명령에 있어서 함수 코드 비트, 함수 수정자 및 명시 캐리-인 및 부호 확장 제어는 데이타 레지스터 D0에 특정되어 있다. 상기 확장된 산술 논리 유닛 명령 동안의 데이타 레지스터 D0의 코딩은 제9도와 관련하여 상술한 바와 같다.

2진수는 부호 또는 비부호로서 지정될 수 있다. 비부호 2진수는 사용되는 비트 범위내에서는 비-부정수(non-negative integer)이다. N비트의 비부호 2진수는 0과 2^N-1 간의 임의의 정수일 수 있다. 부호 2진수는 최상위 비트에서 부호 표시를 캐리한다. 최상위 비트가 "0" 이면 수는 정(+) 또는 제로(0)이다. 최상위 비트가 "1" 이면 수는 부(-) 또는 제로(0)이다. N비트의 부호 2진수는 $-2^{N-1}-1$ 내지 $2^{N-1}-1$로부터의 임의의 정수일 수 있다. 산술 논리 유닛(230)의 동작을 이해하는 데에 있어서는 캐리-아웃 또는 오버플로를 발생시키는 방법을 아는 것이 중요하다.

두개의 비부호 수들의 합은 그 합이 더 이상 수를 위해 사용되는 비트수로 표시할 수 없을 때 오버플로된다. 이 상태는 최상위 비트로부터 캐리-아웃의 발생을 통해 인식된다. 산술 논리 유닛(230)은 8비트, 16비트 또는 32 비트의 수에 따라 동작할 수 있도록 구성될 수 있다. 이 때, 캐리-아웃은 M 플래그 레지스터(211)에 저장될 수 있으며, 정밀도를 유지하도록 사용될 수 있다. 2개의 비부호수는 그 차가 제로(0) 보다 작을 때 언더플로우한다. 부(-)의 수는 비부호수의 표시로 표시될 수 없다. 이하의 예는 비부호 감산 동안 캐리-아웃이 발생되는 방법을 나타내고 있다.

제1의 예는 7("00000111") - 5("00000101")를 나타낸다. 산술 논리 유닛 (230)은 2의 보수 가산에 의해 감산을 실행한다. 비부호 2진 보수의 2의 보수는 수를 반전시켜 1을 가산함으로써 발생될 수 있는데, 즉 $-X = \overline{X}+1$로 될 수 있다. 산술 논리 유닛(230)은 수를 논리적으로 반전시켜(또는 1의 보수화) 최하위 비트에 1의 캐리-인을 주입함으로써 수를 부정한다. 먼저 5는 보수 "11111001" 을 발생시키는 반전된 비트 와이즈이다. 산술 논리 유닛(230)은 제1 비트의 캐리-인 입력으로 주입된 "1" 을 가진 7에 이것을 가산한다. 이것이 다음의 결과를 만들어 낸다.

```
  00000111      7
+ 11111010     -5
+        1
-----------   ---
 100000010      2
```

이것은 최상위 비트로부티 "1" 의 캐리-아웃을 만들어 낸다.

2의 보수 감산에서는 이러한 캐리-아웃이 비-차용(not-borrow)을 표시한다. 따라서 이러한 감산 동안에는 언더플로우가 없다. 다음에는 5-7를 나타낸다. "00000111" 의 8비트의 1의 보수는 "11111000" 이다.

```
  00000101      5
+ 11111000     -7
+        1
-----------   ---
 011111110     -2
```

이 때, "0"의 캐리-아웃은 차용을 나타내므로, 그 결과는 제로(0)보다 작아져 언더플로우가 발생된다. 비부호 감산의 마지막 예는 0-0이다. 0의 8비트의 1의 보수는 "11111111"이다.

	00000000	0
+	11111111	-0
+	1	
	100000000	0

1의 캐리-아웃의 발생은 언더플로우가 없다는 것을 의미한다.

부호 수의 경우는 더욱 복잡하다. 부호 가산에 따른 오버플로는 양자의 피 연산자가 정(+)이고, 결과의 부호 비트가 결과가 정(+)으로부터 부(-)로 넘어가는 것을 나타내는 1(즉, 부)일때 발생한다. 또한, 가산에 따른 오버플로는 양자의 피 연산자가 부(-)이고 그 결과가 0(즉, 정) 부호 비트를 갖는 경우에 발생한다. 혹은, 다시 말해서 가산에 따른 오버플로는 피연산자의 부호 비트 양자가 동일하고 결과가 다른 부호 비트를 가질 때 발생한다. 유사하게, 감산의 오버플로는 피연산자가 동일한 부호를 갖고 결과가 다른 부호 비트를 가질 때 발생한다.

상태 레지스터(210) 또는 Mflags 레지스터(211)에 캐리 비트를 설정할 때, 비트 또는 비트들은 항상 산술 논리 유닛(230)에 의해 발생되는 "자연(natural)" 캐리-아웃이며, 대개의 다른 마이크로 프로세서의 가산 동안에는 캐리-아웃 비트에 따라 "캐리 상태"를 설정하지만 감산 동안은 비-캐리-아웃(또는 차용)에 따라 캐리 상태를 설정한다. 이들의 다른 마이크로 프로세서는 산술 논리 유닛에 대한 적절한 캐리-인을 얻기 위해 차용 감산을 수행할 때 비-캐리를 재-반전시켜야 한다. 이러한 차이는 동일한 브랜치 조건을 얻기 위해 다른 프로세서보다 본 발명을 사용하는 조건부 브랜치 식의 세트가 약간 다르기 때문이다. 캐리/비-차용의 감지를 산술 논리 유닛에 의해 발생된 것과 동일하게 함으로써 각 디지탈 영상/그래픽 프로세서가 이들을 이용하는 많은 방법을 단순화할 수 있다.

산술 명령의 기본 세트에서, 디폴트 캐리는 가산용으로서는 "0"이고 감산용으로서는 "1"이다. 명령 세트 및 어셈블러의 양호한 실시예는 32-비트 산술 연산에서 가산 또는 감산용으로서 캐리-인을 자동적으로 올바르게 설정한다. 명령 세트는 또한 다중 정밀도의 캐리 가산 또는 차용 감산 연산을 지원하도록 상태 레지스터의 캐리-아웃에 따라 캐리-인을 지원한다.

이하 보다 상세히 설명하는 바와 같이, 몇몇 함수의 산술 논리 유닛(230)은 B포트에 대한 입력이 A-포트에 대한 입력으로부터 가산 또는 감산되는지를 제어하는 C-포트를 지원한다. 다중 산술과 이들 산술 논리 유닛 함수를 조합하면 산술 논리 유닛(230)의 각 섹션이 가산 또는 감산하는지를 C-포트에 대한 입력을 통해 제어할 수 있다. 연산의 기본 세트는 섹션이 가산 실행 중이면 "0"의 캐리-인을 공급하기 위한 하드웨어는 제15도와 관련하여 설명되어 있다.

다음은, 디지탈 영상/그래픽 프로세서(71)의 3-입력 산술 논리 유닛(230)에 의해 가능한 산술 함수의 전범위를 상세히 나타내고 있다. 대개의 알고리즘에서 위에 제시한 명령의 서브셋은 아주 적절하다. 설명을 보다 완벽하게 하기 위해 보다 상세한 설명을 추가한다.

이하의 설명에는 산술 논리 유닛(230) 용의 함수 코드를 작동시키는 방법에 대한 정보가 포함되어 있다. 함수 코드 비트 F7-F0에 대한 몇몇의 고찰은 산술 논리 유닛(230)이 여러 가지 연산용으로 사용될 수 있는 방법, 그리고 확장된 산술 논리 유닛 명령을 가장 잘 이용할 수 있는 방법을 이해하는데 도움이 된다. 디폴트 캐리-인은 C 포트에 대한 입력이 A와 B간의 가산 또는 감산의 선택을 제어하는 경우를 제외하고, F0, 즉 함수 코드의 최하위 비트와 동일하다. 모든 함수 코드를 반전시키면 비트는 연산 부호를 수정한다. 예를 들면, A+B를 특정하는 함수 코드 헥사 "66", 그리고 A

-B를 특정하는 헥사 "99"는 비트 와이즈 역이다. 이와 유사하게, 함수 코드 헥사 "65" (A+(B | C)) 및 헥사 "9A" (A-(B | C))는 비트 와이즈 역이다. 확장된 산술 논리 유닛 명령은 확장된 산술 논리 유닛 참(EALUT)과 확장된 산술 논리 유닛 거짓(EALUF)의 쌍에서 발생된다. 확장된 산술 논리 유닛 거짓 명령은 데이타 레지스터 D0의 비트 26-19에 저장된 확장된 산술 논리 유닛 코드 비트를 반전시킨다. 상술한 바와 같이, 본 발명은 일반적으로 가산과 감산 사이에서 선택을 한다. A+(B& C)용 함수 코드 헥사 "6A"의 4개의 최하위 비트를 반전시키면 함수가 A+(B | C)인 헥사 "65"가 발생된다. 이와 유사하게, A-(B& C)용 함수 코드 헥사 "95"의 4개의 최하위 비트를 반전시키면 A-(B | C)인 함수 코드 헥사 "9A"가 된다. 연산 B& C는 B의 비트를 "0"으로(C는 "0"임) 하고, 연산 B | C는 B의 비트를 "1"로 (C는 "0"임) 한다. 이것으로 C와 관련하여 대향하는 마스킹 연산을 달성한다. 이하 설명하는 바와 같이 부호 비트에 따라 함수 코드의 4개의 최하위 비트를 선택적으로 반전시킴에 의해 가산 또는 감산 전에 부호 확장을 수행할 수 있다.

산술 논리 유닛(230)을 사용하여 얻을 수 있는 모든 256개의 산술 함수는 다음과 같이 표시될 수 있다.

$$S=A\&F1(B,C)+F2(B,C)$$

여기서 S는 산술 논리 유닛의 결과이고, F1(B,C) 및 F2(B,C)는 표 17에 도시한 B 및 C의 16개의 가능한 부울 연산 중 임의의 것일 수 있다.

F1 코드	F2 코드	서브 함수	일반적 사용
00	00	0	제로항
AA	FF	올 1 = -1	올 1로 항을 설정
88	CC	B	B
22	33	-B-1	B 부정
A0	F0	C	C
0A	0F	-C-1	C 부정
80	C0	B&C	C가 0일 때 B의 비트를 0으로
2A	3F	-(B&C)-1	C가 0이고 부정일 때 B의 비트를 0으로
A8	FC	B|C	C가 1일 때 B의 비트를 1로
02	03	-(B|C)-1	C가 1이고 부정일 때 B의 비트를 1로
08	0C	B&~C	C가 1일 때 B의 비트를 0으로
A2	F3	(B&~C)-1	C가 1이고 부정일 때 B의 비트를 0으로
8A	CF	B|~C	C가 0일 때 B의 비트를 1로
20	30	-(B|~C)-1	C가 0이고 부정일 때 B의 비트를 1로
28	3C	(B&C)|((-B1)&C)	C=올 0이면 B선택, C=올 1이면 -B 선택
82	C3	(B&C)|((-B-1)&~C	C=올 1이면 B선택, C=올 0이면 -B 선택

[표 17]

제16도는 블럭 다이어 그램 형태로 산술 논리 유닛(230)을 나타낸 도면이다. 산술 유닛(491)은 식의 가산을 형성한다. 산술 유닛(491)은 비트 0 캐리-인 발생기로부터의 비트 0용 캐리 입력을 수신한다. AND 게이트(492)는 A AND F1(B,C)을 형성한다. 논리 유닛(493)은 표 17에 제시되어 있는 함수 코드로부터 서브 함수 F1(B,C)를 형성한다. 논리 유닛(494)은 표 17에 제시되어 있는 함수 코드로부터 서브 함수 F2(B,C)를 형성한다. 산술 논리 유닛(230)의 이러한 설명은 혼합된 부울 및 산술 연산 동안 부울 함수가 산술 기능 이전에 수행되는 것을 나타낸다. 제 13도에 도시한 함수 발생기, 제14도에 도시한 함수 수정자 및 제15도에 도시한 비트 0 캐리-인 발생기와 함께 제10도, 제11도 및 제12도에 도시한 비트 회로(400) 세트가 제16도에 도시한 산술 논리 유닛(230)의 양호한 실시예를 형성한다. 본 기술 분야

에 숙련된 자는 제16도에 도시한 산술 논리 유닛(230)을 실행하는 많은 다른 실시 가능한 방법이 있을 수 있음을 이해할 것이다.

제16도에 명확히 도시한 바와 같이, 서브 함수 F1(B,C) 및 F2(B,C)는 독립적이며, 산술 논리 유닛(230)의 하나의 연산을 위한 다른 서브 함수일 수 있다. 서브 함수 F2(B,C)는 B의 부정 및 C의 부정 양자를 포함한다. 따라서, B 또는 C 중 어느 하나가 그의 부(-)를 가산함으로써 A로부터 감산될 수 있다. 서브 함수 F1(B,C) 및 B2(B,C)용 코드에 의해 제11도 및 제12도에 도시한 산술 논리 유닛(230)용 함수 코드 비트 F7-F0를 유도할 수 있다. 산술 논리 유닛(230)용 함수 코드 비트 F7-F0는 대응하는 서브 함수 F1(B,C) 및 F2(B,C)용 코드는 배타적 OR이다. 서브 함수용 코드는 이러한 결과를 제공하도록 선택되므로, 이들 서브 함수는 동일한 연산용으로서 동일한 코드를 갖지 않는다.

표 17의 서브함수는 가장 일반적으로 유용한 표현 방법으로 되어 있다. 각 함수를 표시하거나 인수 분해하는 다른 방법들이 존재한다. 예컨대, 드모르간 법칙(DeMorgan's Law)을 적용함으로써, 함수 B| C는 (B& C)와 등가로 될 수 있다. X=-X-1이므로, (B& C)는 -(B& C)-1과 등가이고 B| C는 B|(-C-1)과 등가이다. 표 17의 부(-)의 형태 각각은 후속(trailling) "-1" 항을 갖는다. 상술한 바와 같이, 부의 수는 2의 보수이다. 이들은 1의 보수, 즉 -1을 형성하는 비트 와이즈 논리역과 등가이다. "1"의 캐리-인은 -1을 상쇄하고 2의 보수를 형성하도록 최하위 비트로 주입될 수 있다. 부의 서브 함수를 가진 가장 유용한 함수에 있어서 F2(B,C) 서브 함수만이 부(-)를 발생시킨다.

자주 마스킹 연산을 실행하는 것과 같이 표 17의 부울 서브 함수를 생각하는 것이 편리하다. 표 17에 표시한 바와 같이, 서브 함수 B& C는 C의 대응하는 비트가 "0" 인 경우 B의 입력 값은 "0" 로 하는 것으로 해석될 수 있다. 서브 함수 B| C는 C 입력이 "0" 일때 B의 입력값을 모든 비트에 대해 "1" 로 하는 것으로서 해석될 수 있다. 마스크 발생기(234) 및 확장 회로(238)가 멀티플렉서(233)를 통해 산술 논리 유닛(230)의 C-포트를 공급하기 때문에 대개의 경우에 C-포트는 서브 함수에 있어서 B 및 C항을 포함하는 마스크로서 사용되게 된다. 표 17은 C-포트에 대한 입력이 마스크로서 사용되는 것으로 가정하여 각 서브 함수의 식을 함수로 인수분해했다. 상기 식은A-입력이 산술식에서 부정될 수 없음을 나타낸다. 따라서 산술 논리 유닛(230)은 B 또는 C로부터 A를 감산할 수 없다. 한편, B 또는 C 중 어느 하나는 서브 함수 F1(B,C) 및 F2(B,C)가 B 및 C의 부정/역을 지원하기 때문에 A로부터 감산될 수 있다.

상기 식으로 대체될 때 표 17의 서브 함수는 산술 논리 유닛(230)이 수행할 수 있는 모든 256개의 가능한 산술 함수를 발생시킨다. 때때로, 상기 결과의 식에 있어서의 몇몇의 추가의 축소는 초기와 등가인, 보다 이해하기 쉬운 식을 발생시킨다. 이러한 식을 감축할 때, 몇몇 팁들이 도움이 될 수 있다. 기본 명령을 서브 함수 F2(B,C)가 표 17에 설명한 바와 같이 부의 B 또는 C항을 가질 때 가산을 위해 "0" 의 캐리-인 및 "1" 의 캐리-인에 디폴트를 설정한다. 이러한 캐리-인 주입은 이들 서브 함수의 식의 오른쪽 측에서 -1을 효과적으로 상쇄시킴에 의해 1의 보수(논리역)를 2의 보수로 하는 효과를 갖는다. A의 올 "1" 의 논리 AND는 A와 같다. 따라서 서브 함수 F1(B,C)는 식의 좌측에서 A를 얻기 위해 올 "1" 을 출력하도록 설정될 수 있다. 또한, 올 "1" 은 2의 보수 부호 2진수 마이너스 1(-1)과 동일하다.

아래의 예들은 임의의 가능한 산술 논리 유닛 함수와 그들의 대응하는 함수 코드를 얻기 위해 식 및 표 17의 서브 함수를 사용하는 방법을 나타낸다. 산술 함수 A+B는 A& (올 "1")+B로서 표시될 수 있다. 이것은 F(B,C)=올 "1" 및 F2(B,C)=B를 필요로 한다. 올 "1" 에 대한 F1 코드는 헥사 "AA" 이고, B에 대한 F2 코드는 헥사 "CC" 이다. 비트-와이즈 XOR 헥사 "AA" 및 헥사 "CC" 는 헥사 "66" 를 제공한다. 표 16은 헥사 "66" 가 A+B용 함수 코드임을 나타낸다.

산술 함수 A-B는 A&(올"1")+(-B-1)+1로서 표시될 수 있다. 이것은 캐리-인 주입과 함께, F1(B,C)=올"1"(F1 코드 헥사"AA")및"1"의 F2(B,C)=-B-1(F2 코드 헥사"33")을 의미한다."1"의 캐리-인이 부정을 포함하는 서브 함수 F2용 디폴트임을 알 수 있다. 헥사"AA"의 F1 코드를 XOR하며 헥사"33"의 F2 코드를 갖는 비트 와이즈는 헥사"99"를 제공한다. 표 16은 헥사"99"가 캐리-인이"1"이라고 가정하는 A-B용 함수 코드이다.

산술 함수 A+C는 A+B와 유사하게 얻어진다. 즉, A+C = A&(올"1") + (-C+1)+1 과 동일하다. 헥사"AA"의 F1 코드와 헥사"F0"의 F2 코드의 배타적 OR는 헥사"A5"즉 A+C용 함수 코드를 발생시킨다. 마찬가지로, A-C는 A&(올"1") + (-C-1) +1과 같다. 헥사"AA"의 F1 코드와 헥사"0F"의 F2 코드의 배타적 OR는 헥사"A5"즉 A-C용 함수 코드를 발생시킨다.

3개의 입력 산술 논리 유닛(230)은 제3 입력에 따라 2개의 입력 간에 마스킹 및/또는 조건부 함수를 제공함으로써 주요한 이점을 제공한다. 데이타 유닛(110)의 데이타 경로는 C-포트가 마스크 발생기(234)를 사용하는 마스크로서 또는 확장 회로(238)를 사용하는 조건부 제어 입력으로서 가장 유용하도록 한다. 산술 논리 유닛(230)은 항상 혼합된 부울 및 산술 연산에서의 산술 기능이 행해지기 전에 부울 함수를 수행한다. 따라서, 캐리는 비마스크된 비트로부터 제로(0) 또는 부울 함수로서 설정되는 하나 또는 그 이상의 비트로 리플할 수 있다. 다음의 예는 마스킹 및 조건부 연산에 유용하다.

함수 A+(B& C)는 A&(올"1")+(B& C)로 표시될 수 있다. F1(B,C)=올"1"(헥사"AA"의 F1 코드)를 선택하면, F2(B,C)=올"1"(헥사"AA"의 F1 코드)을 선택하면 F2(B,C)=B& C(헥사"C0"의 F2 코드)가 A+(B& C)를 제공한다. 헥사"AA"와 헥사"C0"의 비트-와이즈 배타적 OR은 표 16에 예시한 헥사"6A"의 산술 논리 유닛의 함수 코드를 제공한다. 이 함수는 비부호 수로부터 비트를 제거할 수 있다. 이하에 예시하는 바와 같이, 이 함수는 우측 이동과 가산 연산의 실행 시에 원통형 회전기(235) 및 마스크 발생기(234)와 결합될 수 있다. 이 때, C는 C가"0"인 모든 곳에서 B의 비트를 제로(0)로 하는 비트 마스크로서 작용한다. 마스크 발생기(234)는 우측 자리맞춤 1을 가진 마스크로 발생시킬 수 있으므로, 멀티플렉서 Cmux(233)를 통한 마스크 발생기(234)의 선택으로 A에 가산하기 전에 B의 최상위 비트 몇몇을 제로(0)로 할 수 있다. 이러한 함수의 또 다른 사용은 B를 A에 가산하는 조건부 가산이다. 멀티플렉서 Cmux(233)를 통한 확장 회로(238)의 선택으로 인해 Mflags 레지스터(211)의 비트에 따라 B를 A에 가산하는 제어를 할 수 있다. 다중 산술 동안 Mflags 레지스터(211) 내의 비트는 산술 논리 유닛(230)의 대응하는 섹션을 제어할 수 있다.

함수 A+(B | C)는 A&(올"1")+(B | C)로서 표시할 수 있다. F1(B,C)=올"1"(헥사"AA"의 F1 코드)을 선택하면, F2(B,C)=B | C("CF"의 F2 코드)가 이러한 식을 산출할 수 있다. 헥사"AA"및 헥사"C0"의 비트-와이즈 배타적 OR은 표 16에 예시되어 있는 바와 같이 헥사"65"의 함수 코드를 얻는다.

함수 A-(B& C)는 A&(올"1") +[-(B& -C)-1]+1로 표시될 수 있다."1"의 캐리-인 주입과 함께 F2(B,C)=-(B& C)-1(F2 코드 헥사"3F") 및 F1(B,C)=올"1"(F1 코드 헥사"AA")을 선택하면 이러한 식을 산출할 수 있다. 헥사"AA"및 헥사"3F"의 비트-와이즈 배타적 OR는 표 16에 예시한 바와 같이 함수 코드 헥사"95"를 발생시킨다. 이 함수는 A로부터의 감산 전에 C 입력에 의해 B 입력 내의 비트를 제거 또는 마스크 할 수 있다.

서브 함수가 F1(B,C)=0인 16개의 가능한 함수가 있다. 이들 함수들은 부정, 절대값, 비트 마스킹/및 또는 C-입력에 의해 13 입력의 부호 확장을 수행하도록 다른 하드웨어와 함께 사용된다. 서브 함수가 F1(B,C)=0이면 상수 논리 유닛 연산은 서브 함수 F2(B,C)로 주어진다.

함수 -(B& C)는 (A&"0") + (-(B& C))로서 표시될 수 있다. 이러한 식은 "1"의 캐리-인 주입과 함께 F1(F, C) = 0(F1 코드 헥사 "00") 및 F2(B,C) = -(B& C)-1(F2 코드 헥사 "3F")의 배타적 OR는 표 16에 도시한 바와 같이, 함수 코드 헥사 "3F"를 출력시킨다. 이 기능은 마스크 C에 의해 B 내에 비트를 마스크하고 그 양을 부정한다. 이 기능은 우측 이동과 부정 연산의 일부분으로서 사용될 수 있다.

여러 가지 함수가 유용한 방식의 상기 식의 합의 양자의 항의 마스킹을 지원 한다. 함수(A& C)+(B& C)는 F1(B,C)=C(F1 코드 헥사 "A0") 및 F2(B,C)=B& C(F2 코드 헥사 "C0")를 선택함으로써 달성될 수 있다. 헥사 "A0" 및 헥사 "F0"의 배타적 OR 는 표 16에 예시한 바와 같이 함수 코드 헥사 "60"을 발생시킨다. 이 함수는 가산전에 C가 "0"인 A 및 B 입력의 대응 비트를 효과적으로 제로(0)로 한다. 부울 함수는 가산 전에 인가되며 하나 또는 그 이상의 캐리는 제로(0)로 된 비트로 리플될 수 있음을 알 수 있다. 다중 산술을 이용할 때, 상기한 캐리는 비트로 리플될 수 있음을 알 수 있다. 다중 산술을 이용할 때, 상기한 캐리는 산술 논리 유닛(230)의 분할 섹션간의 경계(boundary)를 넣지 못한다. 이런 함수의 일반적인 사용은 하나의 레지스터에 저장되어 있는 작은 양을 합산하는 것이다. B-포트는 A포트로 향하는 수의 회전된 것을 수신하고, C 포트는 중첩하는 비트를 위한 마스크를 제공한다. 4개의 8비트수는 2개의 16비트수로 합산될 수 있으며 또한 2개의 16비트수는 단일 명령에서 하나의 32 비트 수로 합산될 수 있다.

유사한 함수(A& C)-(B& C)는 F1(B,C)=C(F1 코드 헥사 "A0") 및 F2(B,C)=-(B& C)-1을 선택하고 "1"의 캐리-인을 주입함으로써 달성된다. 헥사 "A0"와 헥사 "3F"의 배타적 OR은 표 16에 예시한 바와 같이, 함수 코드 헥사 "9F"를 발생시킨다. 이 함수는 A 및 B입력의 마스크로서 작용하는 C-포트 값을 갖는 부의 합을 발생시킬 수 있다.

함수(A& B)+B는 F1(B,C)=C(F1 코드 헥사 "A0") 및 F2(B,C)=B(F2 코드 헥사 "CC")를 선택함으로써 달성된다. 헥사 "A0" 및 헥사 "CC"의 배타적 OR는 함수 코드 헥사 "6C"를 제공한다. 이 함수는 A가 올 "1"이거나 혹은 올 "0"인지에 따라 조건부적으로 B를 중복시킬 수 있다.

제17도는 산술 논리 유닛(230) 다른 실시예를 블럭도의 형태로 나타낸 것이다. 제17도의 산술 논리 유닛(230)은 다음의 식을 형성한다.

$$S=F3(A,B,C)+F4(A,B,C)$$

여기서, S는 산술 논리 유닛의 결과이며, F3(A,B,C) 및 F4(A,B,C)는 A,B 및 C의 256개의 가능한 부울 함수의 임의의 것일 수 있다. 가산기(495)는 상기 식의 가산을 형성하며 비트 0 캐리-인 발생기(246)로부터 입력된 최하위 비트 캐리를 위한 입력을 포함한다. 부울 함수 발생기(496)는 입력 함수 신호에 의해 제어되는 함수 F3(A,B,C)를 형성한다. 이와 유사하게, 부울 함수 발생기(497)는 입력 함수 신호에 의해 제어되는 함수 F4(A,B,C)를 형성한다. 부울 함수 발생기(496, 497)는 3개의 입력의 256개의 가능한 부울 조합 세트로부터 A, B 및 C의 선택된 부울 조합을 독립적으로 형성한다. 이러한 구성으로부터 산술 논리 유닛(230)이 산술 조합을 형성하기 전에 부울 조합을 형성하는 것이 명백하다. 제12도의 회로는 이러한 결과를 달성하도록 수정될 수 있다, 제12도에 예시한 발생/킬 함수는 전파 함수에 사용되는 논리 트리(tree)의 일부분을 사용한다. 이것은 통과 게이트(451, 452, 453, 454, 461, 462)로 구성된다. 통과 게이트(451, 452, 453, 454, 461, 462)를 복제하는 상기 함수에 분리 논리 트리를 제공하고 N채널 MOSFET(475)를 제거하면 제17도의 구성이 된다. 이러한 구성에 있어서 발생 또는 킬항 중 하나는 전파 기간과 동시에 발생될 수 있다. 이러한 구성은 제16도에 도시한 것보다 한층 더 큰 융통성을 제공한다.

3개의 입력 산술 논리 유닛(230), 보조 데이타 경로 하드웨어, 및 2진수 시스템의 지식은 많은 유용한 기본적 함수를 형성하는데 사용될 수 있다. 디지탈 영상/그래픽 프로세서의 명령 세트는 일반적으로 마이크로 프로세서에서 보다 프로

그래머에 접근 가능한 많은 하드웨어를 만들 수 있다. 프로그래머가 보다 접근 가능한 하드웨어의 마스킹은 대부분의 다른 프로세서에 숨겨진 몇몇의 아키텍처의 양태가 노출된다. 이러한 명령 세트는 빌딩 블럭과 같이 기본적인 함수를 사용하는 익숙한 연산을 형성하는 것을 지원한다. 이것은 다른 프로세서 내에서 흔히 발견되는 하드웨어 기능 이상으로 프로그래머가 접근할 수 있는 큰 기능성을 부여한다. 디지탈 영상/그래픽 프로세서는 영상, 그래픽 및 다른 처리에 매우 유용한 하드웨어 기능을 갖고 있다. 하드웨어 능력 및 융통성의 결합으로 프로그래머는 대부분의 다른 아키텍처에서 많은 필요로 하는 것을 하나의 명령으로 실행할 수 있다. 다음은 몇몇의 키의 기본적 기능과 보다 복잡한 동작을 하기 위해 그들의 2 또는 그 이상 결합될 수 있는 방법을 설명한다.

이전의 섹션은 데이타 유닛(110)의 각 함수 블럭의 개개의 연산에 대한 설명이었다. 이 섹션은 이들 연산이 보다 복잡한 연산을 수행함에 있어 결합 관계로 사용될 수 있는 방법을 설명하고 있다. 원통형 회전기(235), 마스크 발생기(239) 및 3 입력 산술 논리 유닛(230)은 단일의 산술 논리 유닛의 명령 사이클에서 가산 또는 감산만으로 또는 가산 및 감산을 조합하여 좌측 이동, 비부호 우측 이동을 수행하도록 함께 동작할 수 있다. 어셈블리는 디지탈 영상/그래픽 프로세서(71,72, 73, 74)용 프로그램 코드를 만들어낸다. 이 어셈블리는 비부호(논리) 우측 이동용 부호 " > > u" , 산술(부호) 우측 이동용 " > > " 또는 " > > s" , 그리고 좌측 이동용 " < < " 를 지원한다. 이러한 이동 표기법은 회전, 마스크 발생 및 산술 논리 유닛 함수와 관련하여 적절한 명시 함수를 선택하는 효과적인 마이크로 함수이다. 또한 어셈블리는 원통형 회전(" \\"), 마스크 발생(" %" 및 " %!") 및 산술 논리 유닛 함수를 명시적으로 특정하는 것을 지원한다. 명시적 표기는 이동 표기법으로 표현할 수 없는 일반적인 함수를 특정할 때만 일반적으로 사용된다.

데이타 유닛(110)은 하나의 산술 논리 유닛 사이클에서 좌측 이동 동작을 수행한다. 이러한 좌측 이동 동작은 좌측 이동의 비트수에 의해 원통형 회전기(235)를 통해 원통형 회전기를 포함한다. 상술한 바와 같이, 이러한 회전동안 좌측으로 완전히 회전한 비트는 우측으로 순환되므로, 좌측 이동을 행하기 위해 제거될 필요가 있다. 회전된 출력은 산술 논리 유닛(230)의 B-포트로 송출된다. 마스크 발생기(239)는 이동량을 수신하고 상기 이동량과 동일한 복수의 우측 자리맞춤 1을 가진 마스크를 형성한다. 동일한 이동량은 멀티플렉서 Smux(231)을 통해 제2 입력 버스(202)로부터의 원통형 회전기(235) 및 멀티플렉서 Mmux(234)를 통해 제2 입력 버스(202)로 부터의 마스크 발생기(239)의 회전 제어 입력을 공급한다. 마스크 발생기(239)는 산술 논리 유닛(230)의 C-포트를 공급한다. 산술 논리 유닛(230)은 부울 함수 B&　C를 가진 마스크와 회전된 출력을 조합시킨다. 좌측 이동은 어셈블러에서 다음과 같이 표시된다.

```
Left_Shift = Input << Shift_Amount
```

이 연산은 명시 표기법과 등가이다.

```
Left_Shift = (Input₩₩Shift_Amount)&~%Shift_Amount
```

다음의 예는 4비트씩의 헥사 " 53FFFFA7" 의 좌측 이동을 나타낸다. 여러 단으로 표시하였지만 데이타 유닛(110)은 단일의 패스 산술 논리 유닛 사이클에서 이것을 수행한다. 2진 표기법에서 초기수는

```
0101  0011  1111  1111  1111  1111  1010  0111 이다.
```

원통형 회전기(235)로 4개의 위치를 회전시키면

0011 1111 1111 1111 1111 1010 0111 0101이 되고,

마스크 발생기(239)는 다음의 마스크를 형성한다.

0000 0000 0000 0000 0000 0000 0000 1111

산술 논리 유닛(230)은 논리 조합 B& C를 형성한다.

이것은 회전된 양의 비트를 마스킹하여 이들 비트를 " 0" 으로 하고 다른 비트들을 유지한다. 이렇게 함으로써 좌측 이동의 결과는

0011 1111 1111 1111 1111 1010 0111 0000이 된다.

상기 예의 좌측 이동은 산술 오버플로되는데, 그 이유는 몇몇의 비트들이 " 오버플로" 되었기 때문이다. 좌측 이동 동안, 산술 오버플로는 임의의 비트들이 완전히 이동된 경우 비부호 수로 인해 발생한다. 산술 오버플로는 최종 부호 비트가 초기 부호 비트와 다른 경우 부호 수로 인해 발생할 수 있다. 본 발명의 산술 논리 유닛(230)은 좌측 이동에 따른 산술 오버플로를 자동적으로 검출하지 못한다. 좌측 이동 오버플로는 좌측 이동량으로부터 LMO/RMO/LMBC/RMBC 회로(237)에 의해 발생된 초기수의 최좌측 비트 수정량을 감산함으로써 검출될 수 있다. 그 차이가 제로(0) 이하면 이동 동안 비트들의 오버플로는 없는 것이다. 그 차이가 제로(0) 보다 크면 이러한 차이는 오버플로된 비트수이다.

또, 어셈블러는 좌측 이동과 가산 연산 그리고 좌측 이동과 감산 연산을 수행하기 위해 데이타 유닛(110)을 제어한다. 어셈블러는 A+(B< n)함수를 번역하여 소망이 연산을 수행하도록 원통형 회전기(235), 마스크 발생기(239)및 산술 논리 유닛(230)을 제어한다. 좌측 이동과 가산 연산을 산술 논리 유닛(230)의 연산을 제외하고는 간단한 이동의 상기 예와 동일하게 행한다.

산술 논리 유닛은, 간단한 이동에서와 같이 논리 함수 B& C를 수행하는 대신에, 혼합된 산술 및 논리 함수 A+(B& C)를 실행한다. 좌측 이동과 가산 연산은 어셈블러 표기법에서 다음과 같이 표시된다.

```
LShift_Add = Input1 + Input2 << Shift_Amount
```

이 연산은 다음과 등가이다. 즉,

```
LShift_Add = Input1 + [(Input2₩₩Shift_Amount)&~%Shift_Amount]
```

다음의 예는 헥사 " 000000AA" 의 가산 후의 4비트에 의한 헥사 " 53FFFFA7" 의 좌측 이동을 나타낸다. 모든 이들의 단은 단일의 산술 논리 유닛 사이클만을 필요로한다. 2진 표기법에서 초기 입력 2는

0101 0011 1111 1111 1111 1111 1010 0111이다.

원통형 회전기(235)에서 4부분의 위치를 회전시키면

0011 1111 1111 1111 1111 1010 0111 0101이 된다.

마스크 발생기(239)는 마스크를 형성한다. 즉,

0000 0000 0000 0000 0000 0000 1111

산술 논리 유닛(230)은 좌측 이동 결과를 만들어 내는 논리 조합 B& C를 형성한다. 즉,

0011 1111 1111 1111 1111 1010 0111 0000

2진 표기법에서 다른 피연산자 입력 1은

0000 0000 0000 0000 0000 0000 1010 1010이다.

최종 합은, 0011 1111 1111 1111 1111 1011 0001 1010이다.

산술 논리 유닛(230)은 하나의 사이클로 논리 조합과 산술 조합을 형성하며, 상술한 좌측 이동 결과는 중간 결과로서 유효하지 않다는 것을 유념해 두어야 한다. 또한, 좌측 이동이 오버플로를 발생시키지 않더라도, 그 합이 오버플로될 수 있음을 유념해야 한다. 그 합의 오버플로는 산술 논리 유닛(230)의 최상위 비트로 부터의 캐리-아웃의 발생에 의해 검출된다. 이 조건이 검출되어 상태 레지스터(210)의 " V" 비트에 저장된다.

좌측 이동과 감산 연산은 또한, 단일의 산술 논리 유닛 사이클에서 원통형 회전기(235), 마스크 발생기(237), 및 산술 논리 유닛(230)에 의해 수행되는 함수 세트로 분해된다. 좌측 이동과 감산 연산은 산술 논리 유닛(230)의 함수에서 전술한 좌측 이동 연산 그리고 좌측 이동과 가산 연산과 다르다. 좌측 이동과 감산 산술 동안, 논리 유닛(230)은 혼합된 산술 및 논리 함수 A+(B | C)+1을 수행한다. 산술 논리 유닛(230)은 최하위 비트의 캐리 입력으로 " 1" 을 주입함으로써 " +1" 연산을 수행한다. 캐리-인의 주입은 비트 0 캐리-인 발생기(246)에서 발생된다. 본 발명의 대부분의 감산 연산은 최하위 비트에 " 1" 의 캐리-인을 사용함으로써 행해진다. 어셈블러 표기법으로 좌측 이동과 감산 연산을 표시하면 다음과 같다.

LShift_Sub = Input1 - Input2 << Shift_Amount

이 연산은,

LShift_Sub = Input1 - [(Input2₩₩Shift_Amount)&~%Shift_Amount]+1

과 등가이다.

다음의 예는 헥사 " 00000AA" 의 감산 후의 4비트에 의한 헥사 " 53FFFFA7" 의 좌측 이동을 나타낸다. 모든 이들 단은 하나의 산술 논리 유닛 사이클만을 필요로 함을 유념해야 한다. 2진 표기법에서 초기 입력 2는,

0101 0011 1111 1111 1111 1111 1010 0111이다. 원통형 회전기(235)에서 4부분 위치만큼 회전시키면, 0011 1111 1111 1111 1010 0111 0101이 된다.

마스크 발생기(239)는 마스크를 형성한다. 즉,

0000 0000 0000 0000 0000 0000 0000 1111이다.

논리 조합 B | C의 결과는 다음과 같다. 즉,

1100 0000 0000 0000 0000 0101 1000 1111이다.

2진 표기법에서 다른 피연산자 입력 1은,

0000 0000 0000 0000 0000 0000 1010 1010이다.

그 합 A+(B | C)는 1100 0000 0000 0000 0000 0110 0011 1001이다. 끝으로 최하위 비트 캐리-인으로 주입되는 " 1" 의 가산 결과는,

1100 0000 0000 0000 0000 0110 0011 1010이다.

산술 논리 유닛(230)은 하나의 사이클로 논리 조합과 산술 조합을 형성하며, 상술한 좌측 이동 결과 및 부분합은 모두 중간 결과로서 유효하지 못한 것임을 염두에 두어야 한다.

양호한 실시예의 어셈블리는 단일의 산술 논리 유닛 사이클에서 좌측으로부터 이동된 제로(0)로 비부호 우측 이동을 수행하도록 데이타 유닛(110)을 제어할 수 있다. 원통형 회전기(235)가 좌측 회전을 실행하므로, 최종적으로 우측 회전은 32-n의 회전량으로 형성될 수 있는데, 여기서 n은 오른쪽으로 회전하는 비트수이다. 제2 입력 버스(203) 상의 데이타의 5개의 최하위 비트만이 원통형 회전기(235) 및 마스크 발생기(239)에 의해 사용된다. 따라서, 회전량 32 및 0는 이동 연산 제어에 있어서 동등한 것이다. 어셈블러는 자동적으로 즉시 우측 이동량의 이동를 위해 32-n 연산을 한다. 양호한 실시예의 어셈블러는 프로그래머가 레지스터 기초 이동에 대해 32-n의 양을 형성하도록 요구한다.

우회전을 위한 조정이 행해지면 비부호 우측 이동은 산술 논리 유닛(230)이 다른 함수를 실행하는 것을 제외하고는 좌측 이동과 동일하게 작용한다. 이 연산은 원통형 회전기(235)를 통해 32 n 만큼의 회전을 포함한다. 이러한 순(net) 우회전의 결과로 워드의 최하위 비트로부터 최상위 비트로 순환되는 비트를 갖게 된다. 동일한 양(32-n)이 32-n 우측 자리맞춤 1을 발생시키는 마스크 발생기(239)를 제어한다.

마스크 발생기(239)는 제로(0)의 이동량이 올 " 1" 의 마스크를 발생시키도록 " !" 옵션으로 제어된다. 이 경우에는 어떤 비트도 제거되지 않는다. 산술 논리 유닛(230)은 원통형 회전기(235) 및 마스크 발생기(239)의 출력의 부울 조합을 형성한다.

비부호 우측 이동 연산의 예가 아래에 예시되어 있다. 비부호 우측 이동의 어셈블러 표기법은,

Unsigned_Right_Shift = Input >> u(32-Shift_Amount)

이다.

수행되는 함수를 명시적으로 나타내는 등가 연산은,

Unsigned_Right_Shift =

(Input₩₩(32-Shift_Amount))&%!(32-Shift_Amount)

이다.

상기 식에서, 마스크 연산자 " %!" 는 이동량이 제로(0)인 경우, 올 " 1" 마스크가 발생되는 것을 특정하는 것임을 유념해야 한다. 아래의 예는 4비트 위치 만큼 우측으로 헥사 " 53FFFFA7" 를 이동시킨 비부호 것을 나타낸다. 2진 형태의 초기 수는,

0101 0011 1111 1111 1111 1111 1010 0111이다

이 수가 32-4=28 위치 만큼 좌측으로 회전되면,

0111 0101 0011 1111 1111 1111 1111 1010으로 된다.

마스크 발생기(239)는 입력 32-4=28로부터 마스크를 형성한다.

즉, 0000 1111 1111 1111 1111 1111 1111 1111

끝으로, 산술 논리 유닛(230)은 부울 조합 B& C를 형성하며 그 결과는,

0000 0101 0011 1111 1111 1111 1111 1010 이다.

데이타 유닛(110)은 비부호 우측 이동과 가산 또는 비부호 우측 이동과 감산 연산 중 어느 하나를 수행한다. 양호한 실시예에서, 어셈블러는 함수식 A+B> > u(n)을 번역하여 비부호 우측 이동과 가산 연산을 수행하도록 원통형 회전기(235), 마스크 발생기(239) 및 산술 논리 유닛(230)을 제어하는 명령을 발생시킨다. 비부호 우측 이동과 가산은 산술 논리 유닛(230)이 함수 A+(B& C)를 실행하는 것을 제외하고 간단한 비부호 우측 이동의 전술한 예와 동일하게 작용한다. 양호한 실시예에서 어셈블러는 함수식 A-B> > u(n)을 번역하여 비부호 우측 이동 및 감산 연산을 수행하도록 원통형 회전기(235), 마스크 발생기(299) 및 산술 논리 유닛(230)을 제어하는 명령으로 한다. 비부호 우측 이동과 감산은 산술 논리 유닛(230)이 함수 A-(B|C)+1을 실행하는 것을 제외하고는 간단한 비부호 우측 이동의 전술한 예와 유사하게 작용한다. 좌측 이동과 감산에 따라 " +1" 연산은 비트 0 캐리-인 발생기(246)를 통해 최하위 비트로 " 1" 캐리-인을 주입하는 것을 포함한다.

양호한 실시예의 어셈블러는 하나의 산술 논리 유닛 사이클에서 좌측으로 이동되는 부호 비트를 갖는 부호 우측 이동을 수행하도록 데이타 유닛(110)을 제어할 수 있다. 이 어셈블러는 자동적으로 즉시 우측 이동량에 따라 상기 이동에 대한 32-n 연산을 한다. 데이타 유닛(110)은 원통형 회전기(235)의 입력으로서 부호 비트라고 하는 최상위 비트의 상태를 검출하는 하드웨어를 포함한다. 이 부호 비트는 함수 코드의 4개의 최하위 비트를 제어할 수 있다. 부호 우측 이동 연산은 원통형 회전기(235)에 대한 입력 부호에 따라 수행되는 산술 논리 유닛(230)의 함수를 제어하기 위해 상기 부호 검출 하드웨어를 이용한다. 이 연산은 다음에 기본 함수로서 설명될 수 있다. 원통형 회전기(235)는 32 마이너스 소망의 부호 우측 이동의 비트수(32-n)만큼 좌측으로 회전시킴으로써 순수한 우측 회전을 수행한다. 이동량(32-n)은 마스크 발생기(237)에 공급됨으로써, 32-n의 우측 자리맞춤 올 " 1" 을 발생시킨다. 이 마스크의 올 " 1" 은 우측으로 이동되는 수의 소망의 비트를 선택한다. 이 마스크의 올 " 0" 은 원통형 회전기(235)에 입력되는 최상위 비트와 동일한 부호 비트를 발생시킨다. 이어서, 산술 논리 유닛(230)은 원통형 회전기(235)로부터의 회전된 수와 마스크 발생기(237)로부터의 마스크를 결합시킨다. 산술 논리 유닛(230)에 의해 수행되는 부울 함수는 원통형 회전기(235)에 대한 입력에서의 부호 비트에 의존한다. 이 부호 비트가 " 0" 이면, 산술 논리 유닛(230)은 B& C를 수행하도록 함수 신호를 수신한다. 수정되지 않은 회전된 수가 선택되면, 마스크시에 " 0" 의 임의의 비트를 " 0" 로 한다. 따라서, 최종 결과의 최상위 비트는 원통형 회전기(235)에 대한 입력과 동일한 부호를 표시하는 " 0" 가 된다. 부호 비트가 " 1" 이면 산술 논리 유닛(230)이 B| C를 실행하기 위해 함수 신호를 수신한다. 이 함수는 마스크시에 " 0" 인 " 1" 의 임의의 비트로 되게 하면서 수정되지 않은 회전된 양을 선택한다. 함수 코드의 수정은 검출된 부호 비트가 " 0" 일때 4개의 최하위 비트를 반전시키는 것을 포함한다. 따라서, 결과의 최상위 비트는, 원통형 회전기(235)에 대한 입력과 같이 동일한 부호 표시를 나타내는 " 1" 이 된다.

비부호 우측 이동 연산의 2개의 예가 아래에 예시되어 있다. 부호 우측 이동은 우측 이동에 대한 디폴트 어셈블러 표기이다. 부호 우측 이동에 대한 2개의 허가된 어셈블러 표기는,

```
Signed_Right_Shift = Input >> s(32-Shift_Amount)
```

```
Signed_Right_Shift = Input >> (32-Shift_Amount)
```

이 연산은 부호 검출 하드웨어를 사용하기 때문에, 어셈블러의 양호한 실시예의 표기에 있어서 회전 및 마스킹과 관련하여 이러한 연산을 특정할 수 있는 특별한 방법은 없다. 양호한 실시예에서, 원통형 회전기(235)에 대한 입력 부호는 함수 코드 비트 F3-F0의 반전을 제어한다. 제1의 예는 부(-)의 수의 4위치 부호와 우측 이동, 즉 헥사 "ECFFFFA7" 을 나타낸다. 2진 표기에서의 초기수는

1110 1100 1111 1111 1111 1111 1010 0111 이다.

28(32-4) 위치만큼 좌회전시키면

0111 1110 1100 1111 1111 1111 1111 1010 이 된다.

마스크 발생기(237)는 이러한 마스크를 형성한다.

0000 1111 1111 1111 1111 1111 1111 1111

원통형 회전기(235)에 대한 입력의 최상위 비트가 "1" 이므로 산술 논리 유닛(230)은 부울 조합 B | C를 형성하며, 그 결과는

1111 1110 1100 1111 1111 1111 1111 1010 이 된다.

상기 예에서 올 "1" 은 이동된 결과의 최상위 비트로 이동되며, 초기수의 부호 비트를 정합한다. 제2예는 정(+)의 수의 4위치 부호 우측 이동, 즉 헥사 "5CFFFFA7" 을 나타낸다. 2진 표기의 초기수는,

0101 1100 1111 1111 1111 1111 1010 0111 이다.

28(32-4) 위치만큼 좌측으로 회전시키면

0111 0101 1100 1111 1111 1111 1111 1010 이 된다.

마스크 발생기(217)는 이러한 마스크를 형성한다.

0000 1111 1111 1111 1111 1111 1111 1111

원통형 회전기(235)에 대한 입력의 최상위 비트는 "0" 이므로, 산술 논리 유닛(230)은 함수 코드의 4 개의 최하위 비트의 반전에 의해 B& C의 부울 조합을 형성한다. 그 결과는

0000 1111 1111 1111 1111 1111 1111 1010 이다.

이러한 우측 이동시 올 " 0" 는 최상위 비트로 이동되고, 초기 수의 부호 비트를 정합시킨다는 것을 유념해야 한다.

데이타 유닛(110)은 부호 우측 이동과 가산 또는 좌측 이동과 감산을 수행할 수 있다. 본 발명의 양호한 실시예에 있어서, 어셈블러는 식 A+B > > (n) 또는 A+B> > s(n)을 명령어로 번역하는데, 이 명령어는 부호 우측 이동과 가산을 수행하도록 원통형 회전기(235), 마스크 발생기(239) 및 산술 논리 유닛(230)을 제어한다. 부호 우측 이동과 가산은 산술 논리 유닛(230)에 의해 수행된 기능을 제외하면 부호 우측 이동의 이전 예와 똑같이 동작한다. 부호 우측 이동 및 가산 연산에 있어서, 산술 논리 유닛(230)은 원통형 회전기(235)로의 입력의 부호 비트가 " 0" 인 경우에 함수 A+(B& C)를 수행한다. 이러한 부호 비트가 " 1" 이면, 산술 논리 유닛(230)은 함수 A+(B｜ C)를 수행한다. 본 발명의 양호한 실시예에 있어서, 어셈블러는 식 A-B > > s(n) 또는 A-B > > (n)을 명령어로 번역하는데, 이 명령어는 부호 우측 이동과 감산을 수행하도록 원통형 회전기(235), 마스크 발생기(239) 및 산술 논리 유닛(230)을 제어한다. 부호 우측 이동과 감산은 산술 논리 유닛(230)의 기능을 제외하면 단순 부호 우측 이동의 이전 예와 똑같이 동작한다. 부호 비트가 " 1" 이면, 산술 논리 유닛(230)은 함수 A-(B& C)+1을 수행한다. 부호 비트가 " 0" 이면, 산술 논리 유닛(230)은 다른 함수 A-(B｜ C)+1을 수행한다. 좌측 이동과 감산의 경우에 있어서, " +1" 연산은 비트 제로 캐리-인 발생기(246)를 통해 최하위 비트 내에 " 1" 캐리-인의 인젝션을 포함한다.

원통형 회전기(235), 마스크 발생기(239) 및 산술 논리 유닛(230)은 필드 추출을 단일 사이클 내에서 수행할 수 있다. 필드 추출은 소정의 임의 비트 위치에서 시작하는 워드 내에서 비트들의 필드를 취하고, 필드의 외부에 있는 비트들을 스트립 오프시키며, 필드를 우측 자리맞춤한다. 이러한 필드 추출은 필드를 우측 자리맞춤시키는데 필요한 비트들의 수만큼 워드를 좌측으로 회전시키고 필드의 사이즈 내에서 비트들의 수만큼 회전시킨 결과를 마스킹시키므로써 수행된다. 시프팅의 경우와는 달리, 비트 위치에 기초한 회전량, 및 필드 사이즈에 기초한 마스크 입력은 반드시 같은 크기일 필요는 없다. 양호한 실시예의 어셈블러는 필드 추출에 관한 다음 식을 채택한다:

Field_Extract = (Value₩₩(32-starting_bit))&%!Field size

" %!" 피연산자는 마스크 발생기(237)로 하여금 마스크를 형성하도록 하는데, 이 마스크는 0의 입력을 제외한 필드의 사이즈와 같은 크기의 복수의 우측 자리맞춤된 " 1" 들을 갖는다. 그 경우, 발생된 마스크의 모든 비트들은 " 1" 이기 때문에 비트들은 논리 AND 연산에 의해 마스크되지 않는다. 이러한 회전 및 마스킹은 필드 사이즈가 시작 비트 위치보다 큰 경우에 순환된 비트들을 생성할 수 있다. 이들 파라메타들은 특정 필드가 원시 워드의 끝을 지나 연장되는 변칙적인 경우를 특정한다. 데이타 유닛(110)은 이 경우에 하드웨어적 검사를 하지 않는다. 이러한 결과를 방지하는 것은 프로그래머의 책임이다. 다음의 예는 헥사 " 5CFFFFA7" 의 좌측으로부터 8 비트인 비트(24)에서 시작하는 4-비트 필드의 필드 추출을 증명하는 것이다. 2진수로는 다음과 같다:

0101 1100 1111 1111 1111 1111 1010 0111

필드를 우측 자리맞춤시키기 위해서 수는 32-24 또는 8 비트만큼 좌측으로 회전되어야 한다. 원통형 회전기(235)로부터의 출력은 다음과 같다:

1111 1111 1111 1111 1010 0111 0101 1100

마스크 발생기(237)은 4 비트의 필드 사이즈로 다음 마스크를 형성한다:

0000 0000 0000 0000 0000 0000 0000 1111

마지막으로, 산술 논리 유닛(230)은 부울 조합 B& C를 형성한다. 이는 추출된 필드를 다음과 같이 형성한다:

0000 0000 0000 0000 0000 0000 0000 1100

M플래그 레지스터(211)는 다양한 영상 및 그래픽 처리 연산에 유용하다. 이들 연산은 2가지 클래스로 구분된다. 제1 클래스의 M플래그 연산은 산술 논리 유닛(230)을 통한 단일 패스를 필요로 한다. 수는 M플래그 레지스터(211)에 로드되고 확장 회로(238), 멀티플렉서 Cmux(233), 및 산술 논리 유닛(230)의 C-포트를 통해 산술 논리 유닛(230)의 연산을 제어한다. 컬러 확장은 이들 단일 패스 연산들 중의 한 예이다. 제2 클래스의 M플래그 연산은 산술 논리 유닛(230)을 통한 2가지 패스들을 필요로 한다. 제1 패스 동안에, 특정 비트들은 산술 논리 유닛(230)의 0 결과들의 캐리에 기초하여 M플래그 레지스터(211) 내에 설정된다. 제2 패스 동안에는, M플래그 레지스터(211)의 내용은 확장 회로(238), 멀티플렉서 Cmux(233), 및 산술 논리 유닛(230)의 C-포트를 통해 산술 논리 유닛(230)의 연산을 제어한다. 이러한 2가지 패스 M플래그 연산들은 다중 산술을 이용할 때 특히 유용하다. 부합과 비교, 투명, 최소, 최대 및 포화 등의 많은 연산들은 제2 클래스로 구분된다.

기초 그래픽 연산은 픽셀 사이즈 크기로의 픽셀 형태 디스크립터 당 1 비트 (one bit per pixel shape descriptor)의 변환이다. 이는 종종 컬러 확장(color expansion)으로 불린다. 메모리 공간을 보존하기 위해서, 비트 맵된 텍스트 폰트들의 형태는 픽셀 당 1 비트의 형태로 종종 저장된다. 그 다음, 이들 형태는 디스플레이 메모리 내에 저장될 때 원하는 컬러(들)로 " 확장" 된다. 일반적으로, 형태 디스크립터 내의 " 1" 들은 " 1 컬러" 를 선택하고, 형태 디스크립터 내의 " 0" 들은 " 0 컬러" 를 선택한다. 보통 사용되는 다른 방법은 자리 세이버(place saver) 또는 투명 픽셀로서 작용하는 형태 디스크립터 내에 " 0" 들을 갖는다.

다음 예는 이러한 형태 디스크립터 데이타의 4 비트를 8 비트 픽셀로 변환시킨다. 이 예에서, 다중 산술 연산의 데이타 사이즈는 8비트이다. 따라서, 산술 논리 유닛(230)은 4 독립 8 비트 섹션 내에서 연산한다. 4 비트의 기술어 데이타 " 0110" 은 M플래그 레지스터(211)에 로드된다:

XXXXXXXX XXXXXXXX XXXXXXXX XXXX0110

" X" 로 리스트된 비트들은 컬러 확장 연산에 포함되어 있지 않은 돈케어 비트들이다. 확장 회로(238)는 M플래그 레지스터(211) 내에 있는 4개의 비트들을 다음과 같이, 8 비트 " 1" 들 및 " 0" 들의 블럭들로 확장시킨다:

00000000 11111111 11111111 00000000

1 컬러는 32 비트 데이타 워드 내의 4 픽셀들 각각에 대해 반복된 산술 논리 유닛(230)의 A-포트에 공급된다:

11110000 11110000 11110000 11110000

0 컬러는 4픽셀들 각각에 대해서도 반복된 산술 논리 유닛(230)의 B-포트에 공급된다:

10101010 10101010 10101010 10101010

산술 논리 유닛(230)은 다음을 산출하는 부울 조합 (A& C)|(B& C)를 형성한다:

10101010 11110000 11110000 10101010

컬러 확장은 보통 PixBlt 알고리즘으로 사용된다. 완전한 PixBlt를 수행하기 위해서, 데이타는 회전되어야 하고 이전의 데이타와 합병되어, 확장될 데이타 내의 비트들을 목적지 워드의 픽셀 얼라인먼트와 정렬시킨다. 원통형 회전기(235) 및 산술 논리 유닛(230)은 워드들을 M플래그 레지스터(211) 내에 정렬시킬 수 있다. 이 예에서는 형태 디스크립터 데이타가 적절히 정렬되어 예를 단순화시킬 수 있다는 것을 가정할 수 있다. 또한, M플래그 레지스터(211)가 비트들을 설정하고 비트들을 사용함에 따라 자체 회전 능력을 갖는다는 것을 주지해야 한다. 따라서, 32 비트 워드는 M플래그 레지스터(211) 내에 로드되고, 상기 명령이 8번 반복되어 32 확장 픽셀들을 생성한다.

상기 예에서와 같은 단순 컬러 확장은 결과를 2개의 고체 컬러들 중 하나로 만든다. 가끔은, 특히 사각형 박스들이 중첩될 수 있는 돌출된 텍스트 문자들의 경우에, 형태 디스크립터 내의 " 1" 들이 1 컬러로 확장되지만 자리 세이버 또는 투명 픽셀들로서 작용하는 " 0" 들을 갖는 것이 바람직할 수 있다. 목적지 픽셀 값은 그러한 투명 컬러를 이동시킬 때 변하지 않는다. 데이타 유닛(110)은 0 값 입력으로서 목적지의 원시 내용을 포함하는 레지스터를 단순히 사용함으로써 투명 컬러 확장을 수행할 수 있다. 이러한 예는 다음에 나타난다. 산술 논리 유닛(230)은 이전의 컬러 확장 예로서 동일한 기능을 수행한다. 단지 그 차이는 원시 목적지가 산술 논리 유닛(230)으로의 입력들 중 한 입력으로 된다는 것이다. 4 비트의 기술어 데이타 " 0110" 은 M플래그 레지스터(211) 내에 로드된다:

XXXXXXXX XXXXXXXX XXXXXXXX XXXX0110

확장 회로(238)는 M플래그 레지스터(211) 내에 있는 4개의 비트들을 다음과 같이, 8 비트 " 1" 들 및 " 0" 들의 블럭들로 확장시킨다:

00000000 11111111 11111111 00000000

1 컬러는 32 비트 데이타 워드 내의 4 픽셀들 각각에 대해 반복된 산출 논리 유닛(230)의 A-포트에 공급된다.

11110000 11110000 11110000 11110000

4 픽셀들을 포함하는 원시 목적지 데이타는 산슬 논리 유닛(230)의 B-포트에 공급된다:

11001100 10101010 11101110 11111111

산술 논리 유닛(230)은 다음을 산출하는 부울 조합 (A& C)｜(B& C)를 다시 형성한다:

11001100 11110000 11110000 11111111

결과가 M플래그 레지스터(211) 내의 " 1" 에 대응하는 픽셀들에 대한 1 컬러, 및 M플래그 레지스터(211) 내의 " 0" 에 대응하는 픽셀들에 대한 원시 픽셀값을 포함한다는 것을 주지해야 한다.

데이타 유닛(110)은 일련의 8 비트 크기를 고정 비교값으로의 정확한 부합에 기초하여 픽셀 마스크 당 1 비트를 생성할 수 있다. 이것에 관해서는 다음 예에서 도시된다. 비교값은 32 비트 워드 내에서 4번 반복된다. 산술 논리 유닛(230)은 반복된 비교값을 8 비트 크기를 4개 갖는 데이타 워드로부터 감산한다. 이러한 감산 중에, 산술 논리 유닛(230)은 8 비트 각각의 4개의 섹션들로 분할된다. 제7도에 도시된 제로 검출기(321, 322, 323 및 324)들은 M플래그 레지스터(211) 내에 저장될 데이타를 공급한다. 이 예는 M플래그 레지스터(211)를 회전시키므로써 누산을 증명하기 위해 행으로 된 2개의 명령들을 포함한다. 초기에 M플래그 레지스터 (211)는 돈케어 데이타를 저장한다:

XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX

비교 결과 제1 크기는 다음과 같다:

00000011 00001111 00000001 00000011

비교값은 " 00000011" 이다. 이는 32 비트 워드 내에서 다음과 같이 4번 반복된다:

00000011 00000011 00000011 00000011

산술 논리 유닛(230)은 비교값을 제1 크기로부터 감산한다. 최종 차이는 다음과 같다:

00000000 00001100 11111110 00000000

이것은 M플래그 레지스터(211) 내에 저장되어 있는 다음의 0 비교 " 1001" 을 형성한다. 이 예에서, M플래그 레지스터(211)는 0 결과를 저장하기 전에 사전-클리어된다. 따라서, M플래그 레지스터(211)는 다음과 같다:

00000000 00000000 00000000 00001001

비교 결과, 제2 크기는 다음과 같다:

00000111 11111100 00000011 00000000

동일한 비교값의 제2 감산 결과는 다음과 같다:

00000100 11111001 00000000 11111101

이는 4개의 자리들의 회전에 따라 M플래그 레지스터(211) 내에 저장되어 있는 새로운 0 비교 " 0010" 을 형성한다:

00000000 00000000 00000000 10010010

추가적인 비교는 M플래그 레지스터(211)가 32 비트를 저장할 때까지 동일한 형식으로 만들어질 수 있다. 그 다음, M플래그 레지스터(211)의 내용은 다른 레지스터로 이등되거나 메모리에 기입될 수 있다.

임계 검출은 픽셀 값을 고정 임계 값에 비교한다. 임계 검출은 고정 임계값보다 크거나 작았던 픽셀 값을 유효화시키는 각각의 픽셀에 대해 1 비트 값을 설정한다. 특정 응용에 따라, 동일한 경우는 크거나 작은 것으로 그룹화된다. 데이타 유닛(110)은 단일 산술 논리 유닛 사이클 내에서 비교 결과를 형성하도록 프로그램될 수 있다. 산술 논리 유닛(230)은 테스트될 크기와 고정 임계 값 사이의 차를 형성한다, 산술 논리 유닛(210)의 각각의 섹션으로부터의 캐리-아웃은 M플래그 레지스터(211) 내에 세이브된다. 테스트될 크기 I가 이로부터 감산된 고정 임계값 T를 포함하면, 캐리-아웃은 I가 T보다 크거나 같은 경우에만 발생된다. 상술한 바와 같이, 산술 논리 유닛(230)은 2의 보수 가산에 의해 감산을 수행하고, 이러한 상황 하에서 캐리-아웃은 빌려오지 않음(not-borrow)을 나타낸다. 다음은 임계 값이 " 00000111" 인 4개의 8 비트 크기에 대한 프로세스의 예이다. 4개의 8 비트 크기 I가 테스트된다:

00001100 00000001 00000110 00000111

32 비트 워드 내에서 4번 반복된 임계 값 T는 다음과 같다:

00000111 00000111 00000111 00000111

그 차이는 다음과 같다:

00000101 11111010 11111111 00000000

이는 다음의 캐리-아웃 " 1001" 을 생성한다. M플래그 레지스터(211) 내에서 이것에 대한 결과는 다음과 같다.

XXXXXXXX XXXXXXXX XXXXXXXX XXXX1001

부합 검출의 경우에 있어서와 같이, 이러한 단일 명령은 32 비트가 형성될 때까지 M플래그 레지스터 회전에 의해 새로운 데이타에 대해 반복될 수 있다.

2개의 비부호(unsigned) 수들을 가산하면, 캐리-아웃은 결과의 비트들 수로 표시될 수 있는 것보다 결과가 크다는 것을 나타낸다. 이러한 캐리-아웃은 결과의 정밀도인 최상위 비트를 나타낸다. 따라서, M플래그 레지스터(211) 내로 캐리-아웃들의 세이빙은 정밀도를 유지하기 위해 사용될 수 있다. 이들 캐리-아웃 비트들은 정밀도를 유지하기 위해 이후의 가산시에 세이브될 수 있다. 특히, 다중 산술에 이용될 때, 정밀도를 보다 적은 비트들로 제한함으로써 동일한 프로세스가 보다 적은 산술 논리 유닛 사이클 동안에 수행될 수 있도록 종종 인에이블시킨다.

제2 타입의 M플래그 연산은 M플래그 레지스터(211) 내의 비트들을 설정하면서 M플래그 레지스터(211) 내에 저장된 비트들을 채택함으로써 산술 논리 유닛(230)의 연산을 제어한다. 다중 산술 연산은 M플래그 레지스터(211)의 확장들과의 조합시에 이용되어 다중 병렬 바이트 또는 반-워드 연산들을 수행할 수 있다. 또한, M플래그 레지스터(211) 내에 비트들을 설정하고 M플래그 레지스터(211)를 산술 논리 유닛(230)으로 확장하는 것은 여러 가지 다양한 방식에 사용될 수 있는 역 공간 변환(inverse space conversion)이다.

다음의 예는 투명 기능을 수행하기 위해 확장을 이용하는 명령이 뒤따르는 8비트 다중 산술 명령의 조합을 나타내기 위한 것이다. 투명은 사각형이 아닌 형태들의 사각형 PixBlt들을 수행할 때 보통 사용된다. 투명 픽셀들은 목적지에 영향을 끼치지 않으므로써 원시 목적지가 전체적으로 투명한 자리 세이버 픽셀들로서 사용된다. 투명으로 인해, 투명 코드와 동일하지 않은 발생지 내의 픽셀들만이 목적지에서 대체된다. 제1 명령에 있어서, 투명 컬러 코드는 발생지로부터 감산되고 M플래그 레지스터(211)는 동등한 제로(zero)에 기초하여 설정된다. 주어진 8 비트 크기가 투명 코드와 부합되면, 대응하는 " 1" 은 M플래그 레지스터(211) 내에 설정된다. 제2 명령은 확장 회로(238)을 사용하여 M플래그 레지스터(211)를 확장시켜 발생지 또는 목적지의 픽셀x 픽셀 단위(pixel by pixel basis)의 선택을 제어한다. 산술 논리 유닛(230)은 이러한 선택을 위해 함수 (A& C)｜(B&　C)를 수행한다. 이러한 부울 함수는 비트 단위로 수행되는 반면, M플래그 레지스터(211)는 픽셀 사이즈 8로 확장되어 픽셀들 사이에서 선택한다. 픽셀 발생지는 다음과 같다 :

00000011 01110011 00000011 00000001

투명 코드 TC는 " 00000011" 이다. 32 비트 워드를 채우기 위해 4번 반복하면, 다음과 같이 된다:

00000011 00000011 00000011 00000011

그 차이 SRC-TC는 다음과 같다:

00000000 01110000 00000000 11111110

이는 제로 검출 비트 " 1010" 을 생성한다. 따라서, M플래그 레지스터(211)는 다음을 저장한다:

XXXXXXXX XXXXXXXX XXXXXXXX XXXX1010

제2 명령에서, 확장 회로(238)는 M플래그 레지스터(211)를 다음과 같이 확장한다:

11111111 00000000 11111111 00000000

원시 목적지 DEST는 다음과 같다:

11110001 00110011 01110111 11111111

원시 발생지 SRC는 산술 논리 유닛(230)용 제3 입력을 형성한다. 산술 논리 유닛(230)은 그 다음에 부울 조합 (DEST& @MF)|(SRC& @MF)를 다음과 같이 형성한다:

11110001 00010011 01110111 00000001

최종 결과는 발생지가 투명하지 않은 경우의 발생지 상태를 갖고, 그렇지 않으면 목적지의 상태를 갖는다는 것을 주지해야 한다. 이는 투명 기능에 관한 것이다.

데이타 유닛(110)은 M플래그 레지스터(211) 및 2개의 산술 논리 유닛 사이클을 이용하여 최대 및 최소 함수들을 수행할 수 있다. 최대 함수는 2개의 비부호 픽셀 값들 중 더 큰 것을 결과로서 취한다. 최소 함수는 2개의 비부호 픽셀 값들 중 더 작은 것을 결과로서 취한다. 이들 연산 중에, 제1 명령은 다중 감산을 수행함으로써, 캐리-아웃들에 기초하여 M플래그 레지스터(211)를 설정한다. 따라서, 상태 설정에 있어서, 산술 논리 유닛(230)은 OP1-OP2를 형성한다. 이러한 제1 명령만이 M플래그 레지스터(211)를 설정하고, 최종 차이는 없어진다. 제2 명령으로 최대 함수를 수행하면, 산술 논리 유닛(230)은 연산 (OP1& @MF)|(OP2& @MF)를 수행한다. 이는 개별 픽셀들 중 최대를 형성한다. 제1피연산자 OP1을 다음과 같이 정한다:

00000001 11111110 00000011 00000100

제2 피연산자 OP2는 다음과 같다:

00000011 00000111 00000111 00000011

그 차이 OP1 - OP2는 다음과 같다:

11111110 11110111 11111100 00000000

M플래그 레지스터(211)를 다음과 같이 설정하는 캐리 - 아웃(빌리지 않음) " 0101" 을 생성한다.

XXXXXXXX XXXXXXXX XXXXXXXX XXXX0101

제2 명령에 있어서, M플래그 레지스터(211) 내의 4개의 최하위 비트들은 확장 회로(238)을 통해 확장되어 다음을 생성한다:

00000000 11111111 00000000 11111111

산술 논리 유닛(230)은 부울 함수 (OP1& @MF)｜(OP2& @MF)를 수행한다. 이는 다음과 같은 결과를 생성한다 :

00000011 11111110 00000111 00000100

결과의 각각의 8 비트 섹션은 OP1 및 OP2의 대응하는 섹션들 중 더 큰 것의 상태를 갖는다. 이는 최대 함수에 관한 것이다. 최소 함수는 제2 명령에서 산술 논리 유닛(230)이 부울 함수 (OP1& @MF)｜(OP2& @MF)를 수행한다는 것을 제외하고는 상기 최대 함수와 유사하게 동작한다. 이러한 부울 함수는 각각의 8 비트 섹션에 대해 더 큰 크기보다는 더 작은 크기를 선택한다.

또한, 데이타 유닛(110)은 포화되는 가산 기능(add - with - saturate function)을 수행할 수 있다. 포화되는 가산 기능은 오버플로가 발생되지 않으면 정규 가산처럼 동작한다. 이 경우, 포화되는 가산 기능은 결과를 올 " 1" 로 클램프시킨다. 포화되는 가산 기능은 보통 그래픽 및 영상 처리에 사용되어, 작은 정수 결과를 최상위 수에서 하위 수로 다시 오버플로되는 것으로부터 방지한다. 다음의 예는 4개의 8 비트 픽셀들에 대해 다중 산술을 이용하여 포화되는 가산 기능을 2개의 명령으로 형성하는 것을 도시한다. 먼저, 가산은 M플래그 레지스터(211)에 저장된 캐리 - 아웃들을 이용하여 일어난다. " 1" 의 캐리 - 아웃은 오버플로를 나타내므로, 그함은 포화된 값인 올 " 1" 들로 설정된다. 그 다음, 확장 회로(238)는 M플래그 레지스터(211)를 확장시켜 합 또는 포화된 값의 선택을 제어한다. 제1 피연산자 OP1은 다음과 같다:

00000001 11111001 00000011 00111111

제2 피연산자 OP2는 다음과 같다:

11111111 00001011 00000111 01111111

산술 논리 유닛(230)은 합 OP1+OP2 = RESULT를 형성하여 다음을 생성한다:

00000000 00000100 00001010 10111110

이는 " 1100" 의 캐리-아웃에 대응한다. 이들은 다음과 같이 M플래그 레지스터 (211)에 저장된다:

XXXXXXXX XXXXXXXX XXXXXXXX XXXX1100

제2 명령에서, 확장 회로(238)는 M플래그 레지스터(211)의 4개의 최하위 비트를 다음과 같이 확장한다:

11111111 11111111 00000000 00000000

산술 논리 유닛(230)은 부울 함수 RESULT | @MF를 수행하여 다음을 형성한다:

11111111 11111111 00001010 10111110

합이 오버플로되지 않을 때 제2 명령의 결과는 합과 같고, 합이 오버플로될 때는 " 11111111" 과 같다는 것을 주지해야 한다.

마찬가지로, 데이타 유닛(110)은 포화되는 감산 기능을 수행할 수 있다. 포화되는 감산 기능은 언더플로가 발생하지 않으면 정규 감산처럼 동작한다. 그 경우, 포화되는 감산 기능은 결과를 올 " 0" 들로 클램프시킨다. 포화되는 감산 기능은 또한 그래픽 및 영상 처리에 보통 사용된다. 데이타 유닛(110)은 상술한 포화되는 가산 기능과 유사하게 포화되는 감산 기능을 수행한다. 먼저, 감산은 M플래그 레지스터(211)에 저장된 캐리-아웃들을 이용하여 일어난다. " 0" 의 캐리-아웃은 빌림(borrow)을 나타내므로 언더플로이다. 그 경우, 차이는 포화된 값인 올 " 0" 들로 설정된다. 확장 회로(238)은 M플래그 레지스터(211)를 확장시켜 차이 또는 포화된 값의 선택을 제어한다. 이러한 제2 명령 동안에, 산술 논리 유닛(230)은 부울 함수 RESULT& @MF를 수행한다. 이는 대응하는 캐리-아웃이 " 0" 인 경우에 조합을 " 0" 으로 만들어, 차이를 올 " 0" 들로 포화시킨다. 반면에, 대응하는 캐리-아웃이 " 1" 인 경우에는 부울 조합이 RESULT와 같다.

제18도는 디지탈 영상/그래픽 프로세서(71, 72, 73 및 74)에 대한 명령 워드의 포맷을 도시한다. 명령 워드는 제18도에 도시된 바와 같이, 일반적으로 2개의 병렬 섹션들로 분할되어 있는 64 비트들을 포함한다. 명령 워드의 최상위 25 비트(비트 63-39)들은 데이타 유닛(110)에 의해 수행된 연산의 형태를 특정한다. 명령 워드의 최하위 39 비트(비트 38-0)들은 데이타 유닛(110)에 의해 병렬로 수행된 데이타 전송을 특정한다. 데이타 유닛(110)의 연산에 대해서는 5개의 포맷 A, B, C, D 및 E들이 있다. 데이타 전송 포맷(1 내지 10)들에 대한 10가지 형태들이 있다. 명령 워드는 데

이타 전송을 특정하는 다른 대안으로서 32 비트 즉시 값(immediate value)을 특정한다. 명령 워드는 32 비트 즉시 값을 특정할 때 상술한 2개의 섹션들로 분할되지 않는데, 이는 일반적인 법칙에 예외적이다. 많은 명령들은 데이타 유닛(110)을 이용하지 않는 연산들을 수행한다. 이들 명령은 병렬 데이타 전송 동작을 가능케 하거나, 병렬 데이타 전송 동작은 명령에 따라 방해받을 수도 있다. 다른 특징에 있어서, 데이타 유닛(110)에 대해 특정된 연산들은 데이타 전송에 특정된 연산들과는 무관하다.

명령 워드 대안들은 다음과 같이 요약된다. 데이타 유닛(110)의 연산은 단일 산술 논리 유닛 연산 또는 단일 승산 연산일 수 있고, 또는 각각은 병렬로 수행될 수 있다. 데이타 유닛(110)의 모든 연산들은 명령 워드 내의 필드에 기초하여 제약될 수 있다. 병렬 데이타 전송은 데이타 포트 유닛(140)의 로컬 데이타 포트(local port;141) 및 글로벌 데이타 포트(global port;145)를 통해 메모리로 및/또는 이로부터 수행될 수 있다. 2개의 데이타 전송 동작들은 명령 워드 내에서 독립적으로 특정된다. 12개의 어드레싱 모드들은 레지스터 또는 오프셋 인덱스의 선택에 따라, 각각의 메모리 액세스에 대해 지원된다. 데이타 유닛(110) 내에서 내부 레지스터간 전송은 글로벌 데이타 포트(145)를 통한 메모리 액세스 대신에 명령 워드로 특정될 수 있다. 데이타 유닛(110)의 연산이 발생지 또는 목적지로서 비-데이타 유닛 레지스터를 이용할 때, 명령 워드 중 소정의 병렬 데이타 전송 섹션은 추가 레지스터 정보를 특정하고, 글로벌 포트 발생지 데이타 버스 GSRC(105) 및 글로벌 포트 목적지 데이타 버스 GDST(107)은 데이타를 데이타 유닛(110)에 대해 전송한다.

명령 워드 중 보통 로컬 버스 데이타 전송을 특정하는 부분은 다른 용도를 갖는다. 다른 용도는 조건부 데이타 유닛(110) 연산 및/또는 글로벌 메모리 액세스 또는 레지스터간 이동을 가능케 하는 것이다. 제한된 조건부 발생지 선택은 데이타 유닛(110)의 연산에서 지원된다. 데이타 유닛(110)의 결과는 조건부로 세이브되거나 소거될 수 있으며, 유리하게는 브랜치를 갖지 않고도 연산을 조건부로 수행할 수 있다. 상태 레지스터의 각각의 개별 비트의 갱신은 또한 조건적으로 선택될 수 있다. 메모리로의 조건부 저장은 2개의 레지스터들 간에서 선택된다. 메모리로부터의 조건부 로드는 데이타를 로드 또는 소거시킨다. 조건부 레지스터간 이동은 데이타를 목적지에 기입하거나, 소거시킨다.

제18도에서 명령 워드의 형태에 대한 설명, 및 5개의 데이타 유닛 연산 포맷들의 여러 비트들과 필드들에 대한 해석과 용어 설명은 다음에 기술된다. 비트들과 필드들은 명령 워드 뿐만 아니라 회로도 정의하는데, 이 회로는 특정된 논리 관계에 따라 명령 워드를 디코드한다. 이 회로는 특정 연산 또는 제시된 연산들을 수행하기 위해서 명령 워드의 특정 비트 또는 필드 또는 논리 조합에 응답한다. 따라서, 이 분야에서는, 비트, 필드, 포맷 및 연산들에 대한 특정은 본 실시예의 중요하면서도 유리한 특징들을 정의하고 명령 워드를 디코드 또는 구현하기 위한 대응하는 논리 회로를 특정한다. 이 회로는 프로그램가능 논리 어레이(PAL), 또는 현재 알려지거나 이후에 고안된 다른 회로 형태에 숙련된 자들이 이 명세서를 통해 간단히 구현할 수 있는 것이다. 정당한 연산 조합에 대한 설명은 명령 워드 포맷의 설명을 가능케 한다.

데이타 유닛 포맷 A는 비트 63=" 1" 과 비트 44=" 0" 으로 인식된다. 데이타 유닛 포맷 A는 5 비트 즉시 필드를 비용하여 기초적인 산술 논리 유닛 연산을 특정한다. " 클래스" 필드(비트 62-60)는 산술 논리 유닛(230)과 관련하여 데이타 유닛(110) 내에서 데이타 루팅(data routing)을 지정한다. 표 18은 데이타 유닛 포맷 A, B 및 C에 대한 " 클래스" 필드에 대응하는 데이타 루팅에 관한 정의를 보여준다.

클래스 필드 666 210	입력 A	입력 B	입력 C	maskgen	회전
000	src2/im	src1	@MF	-	0
001	dstc	src1	src2/im	-	D0 (4-0)
010	dstc	src1	mask	src2/im	0
011	dstc	src1	mask	src2/im	src2/im
100	src2/im	src1	mask	D0 (4-0)	D0 (4-0)
101	src2/im	src1	@MF	-	D0 (4-0)
110	dstc	src1	src2/im	-	0
111	src1	Hex "1"	src2/im	-	src2/im

[표 18]

표 18에서, " 입력 A" 는 입력 A 버스(241)용 Amux(232)에 의해 선택된 발생지이다. 발생지 " src2/im" 은 데이타 유닛 포맷 A에서는 " immed" 필드 (비트 43-39)의 5 비트 즉시 값이거나, 데이타 유닛 포맷 B에서는 " src2" 필드 (비트 41-39)에 의해 지정된 데이타 레지스터(200)이거나, 데이타 유닛 포맷 C에서는 " 32-비트 즉시" 필드(비트 31-0)의 32 비트 즉시 값이다. 발생지 " dstc" 는 산술 논리 유닛(230) 결과의 목적지에 대한 보고 데이타 레지스터(200)이다. 이러한 보조 데이타 레지스터(200)는 " 0110" 과 같은 상부 4개의 비트들로 레지스터 지정되므로, 데이타 레지스터(200)들 중 하나를 특정하며, " dst" 필드 (비트 50-48)로 특정된 하부 3개의 비트들로 레지스터 지정된다. 보조 레지스터들에는 목적지의 레지스터 뱅크를 특정하기 위한 " Adstbnk" 필드(비트 21-18) 및 입력 B의 레지스터 뱅크를 특정하기 위한 " Aslbank" 필드(비트 9-6)를 사용하는 전송 포맷(6 및 10)들이 사용된다. 이것은 장거리 목적지로서 알려져 있는데, 그 이유는 목적지가 데이타 레지스터(200)들 중에 있는 것이 아니기 때문이다. 따라서, 1개의 발생지 및 목적지는 레지스터 번호가 같은 서로 다른 레지스터 뱅크들을 가질 수 있다. 표 19는 " Adstbnk" 필드 내에 특정된 레지스터 뱅크에 기초한 여러 가지 다른 디지탈 영상/그래픽 프로세서 레지스터들에 대한 보조 레지스터들을 도시한다. 소정의 다른 전송 포맷에 있어서, 이러한 원시 레지스터가 " dst" 필드로 특정된 레지스터 번호를 갖는 데이타 레지스터(200)인 것을 주지해야 한다.

보조 데이타 레지스터

Adstbnk	D0	D1	D2	D3	D4	D5	D6	D7
0 0 0 0	A0	A1	A2	A3	A4	--	A6	A7
0 0 0 1	A8	A9	A10	A11	A12	--	A14	A15
0 0 1 0	X0	X1	X2	--	--	--	--	--
0 0 1 1	X8	X9	X10	--	--	--	--	--
0 1 0 0	D0	D1	D2	D3	D4	D5	D6	D7
0 1 0 1	--	SR	MF	--	--	--	--	--
0 1 1 1	CALL	BR	IPS	IPRS	INTEN	INTFLG	COMM	LCTL
1 1 0 0	LC0	LC1	LC2	--	LR0	LR1	LR2	--
1 1 0 1	LRSE0	LRSE1	LRSE2	--	LRS0	LRS1	LRS2	--
1 1 1 0	LS0	LS1	LS2	--	LE0	LE1	LE2	--
1 1 1 1	--	--	--	--	TAG0	TAG1	TAG2	TAG3

[표 19]

표 19에서, " - -" 는 예약 레지스터를 나타낸다. 표 19에는 레지스터 뱅크 " 0110" , " 1000" , " 1001" , " 1010" 또는 " 1011" 이 리스트되지 않는다. 이들 뱅크 내의 모든 레지스터들은 예약되거나 모방 기능들에 할당되며, 보통은 장거리 목적지로 사용하지 않는다.

표 18에서, " 입력 B" 는 입력 B 버스(242)를 공급하는 원통형 회전기(235)용 발생지이다. " src1" 로 지정된 " 입력 B" 발생지는 데이타 유닛 포맷 A 및 B에서 " src1' 필드(비트 47-45)로 표시되거나, " slbank" 필드 (비트 38-36)의 레지스터 뱅크 및 " src1" 필드(비트 48-45)의 레지스터 번호로 표시된 데이타 레지스터(200)인데, 이는 데이타 포맷 C에서는 데이타 유닛(110) 내에 있는 소정의 64 하부 어드레스가능 레지스터들일 수 있다. " 입력 B" 에 대한 " Hex 1" 발생지는 버퍼(236)로 부터의 " 1" 과 같은 32 비트 정수이다. 표 18에서, " 입력 C" 는 입력 C 버스(243)용 Cmux(233)에 의해 선택된 발생지이다.

" 입력 C" 발생지 " @MF" 는 상태 레지스터(210)의 " M사이즈" 필드(비트 5-3)에 따라 확장 회로(238)에 의해 확장된 다중 플래그 레지스터(211)로부터 1개 이상의 비트들이다. 상태 레지스터(210)의 " M사이즈" 필드의 정의에 대해서는 표 2를 참조하라, " src2/im" 발생지는 " 입력 A" 발생지와 관련하여 이미 기술되어 있다. " mask" 발생지는 마스크 발생기(239)의 출력이다. 표 18에서, 마스크 발생기(239)용 Mmux(234)에 의해 선택된 발생지이다. 이러한 발생지는 이미 기술된 " src2/im" 이거나 " D0 (4-0)" 일 수 있는데, 이는 데이타 레지스터 D0의 " DBR" 필드(비트 4-0)의 디폴트 원통형 회전량이다. 표 18에서, " 회전" 은 원통형 회전기(235)의 회전량을 제어하기 위해 Smux(231)에 의해 선택된 발생지이다. 이러한 발생지는 회전이 없다는 것을 나타내는 " 0" 이거나, 데이타 레지스터 D0의 " DBR" 필드(비트 4-0)의 디폴트 원통형 회전량을 나타내는 D0 (4-0)" 이거나, 이미 기술된 " src2/im" 일 수 있다.

" ari" 비트(비트 59)는 데이타 유닛(110)의 산술 논리 유닛(230)이 산술 연산 또는 부울 논리 연산에 사용되는 지를 지정한다. " ari" 비트가 " 1" 이면, 산술 연산이 발생하고, " 0" 이면 부울 논리 연산이 발생한다.

데이타 유닛 포맷 A는 산술 논리 유닛(230)의 연산의 명령 워드 명세(instruction word specification)를 허가한다. " 8-비트 ALU 코드" 필드(비트 58-51)는 산술 논리 유닛(230)에 의해 수행된 연산을 지정한다. 이러한 필드는 " ari" 비트가 " 1" 인 경우에 산술 연산을 지정한다. 이러한 경우에는, " 8-비트 ALU코드" 비트(57, 55, 53 및 51)들은 표 6에 의한 " 8-비트 ALU 코드" 비트(58, 56, 54 및 52)들로 구성된 " FMOD" 필드에 의해 변형된 바와 같이, 표 4에 의한 산술 연산을 지정한다 " ari" 비트가 " 0" 이면, 이는 부울 연산이고, " 8-비트 ALU 코드" 필드는 표 13에 의한 함수 코드 비트 F7-F0들로 번역된다. 이들 인코딩에 대한 상세는 데이타 유닛(110)의 설명과 관련하여 이미 기술되었다.

데이타 유닛 포맷 A는 2개의 발생지들과 산술 논리 유닛(230)용 목적지를 지정한다. " dst" 필드(비트 50-48)는 레지스터를 산술 논리 유닛(230)용 목적지로서 지정한다. " dst" 필드는 레지스터 번호에 의해 데이타 레지스터(200)들 중 한 레지스터로 인용될 수 있거나, " dst" 필드의 레지스터 번호는 전송 포맷에 따라 장거리 레지스터를 특정시키기 위해 레지스터 뱅크와 관련하여 사용될 수 있다. " src 1" 필드(비트 47-45)는 레지스터를 산술 논리 유닛(230)용 제1 발생지로서 지정한다. 이는 데이타 레지스터(200)들 중 하나일 수 있거나, 전송 포맷에 따라 장거리 레지스터를 특정시키기 위해 레지스터 뱅크와 관련하여 사용될 수 있다. " immed" 필드(비트 43-39)는 산술 논리 유닛(230)용 제2 발생지로서 사용된 5 비트 즉시 값을 지정한다. 사용시에, 이러한 5비트 즉시 값은32비트들로 확장된 0이다. 레지스터 뱅크의 사용에 대해서는 전송 포맷에 대한 설명과 관련하여 이후에 설명한다.

목적지 레지스터에 결과의 저장은 " cond." 필드 내에 표기된 조건이 진실인 경우에만 발생한다. " cond." 필드(비트 35-32)는 조건부 연산에 대한 조건을 지정한다. 이러한 " cond." 필드는 명령 워드 중에서 일반적으로 전송 포맷에 사용된 부분에 해당된다는 것을 주지해야 한다. 전송 포맷(7, 8, 9 및 10)들은 이러한 필드를 포함한다. 따라서, 산술 논리 유닛(230)의 결과에 대한 조건부 저장은 이들 전송 포맷들이 사용될 때에만 발생한다. 양호한 실시예에 있어서, " cond." 필드는 표 20에서 후술되는 바와 같이 디코드된다.

조건 필드 비트 3 3 3 3 5 4 3 2	기호	조건 설명	비교된 상태 비트
0 0 0 0	u	비조건부	---
0 0 0 1	p	포지티브	~N&~Z
0 0 1 0	ls	낮거나 같음	~C | Z
0 0 1 1	hi	그보다 높음	C&~Z
0 1 0 0	lt	그보다 적음	(N&~V) | (~N&V)
0 1 0 1	le	적거나 같음	(N&~V) | (~N&V) | Z
0 1 1 0	ge	크거나 같음	(N&V) | (~N&~V)
0 1 1 1	gt	그보다 큼	(N&V&~Z) | (~N&~V&~Z)
1 0 0 0	hs, c	그보다 낮음, 캐리	C
1 0 0 1	lo, nc	높거나 같음, 캐리 없음	~C
1 0 1 0	eq, z	같음, 0	Z
1 0 1 1	ne, nz	같지 않음, 0 아님	~Z
1 1 0 0	v	오버플로	V
1 1 0 1	nv	오버플로 아님	~V
1 1 1 0	n	네가티브	N
1 1 1 1	nn	비-네가티브	~N

[표 20]

조건들은 상태 레지스터(210)에 관하여 검출된다. 상술한 바와 같이, 상태 레지스터(210)는 산술 논리 유닛(230)의 출력의 조건에 관련한 몇몇 비트들을 저장한다. 이들 조건들은 네가티브, 캐리, 오버플로 및 0을 포함한다. 상태 레지스터(210)와 관련된 산술 논리 유닛(230)의 조건부 연산에 대해서는 데이타 유닛(110)의 설명과 관련하여 이미 기술되었다.

데이타 유닛 포맷 B는 비트 63=" 1" , 비트 44=" 0" 으로 인식된다. 데이타 유닛 포맷 B는 산술 논리 유닛(230)의 제2 발생지를 위해 특정된 레지스터를 이용하여 기초적인 산술 논리 연산을 특정한다. " class" 필드는 표 18과 관련하여 이미 기술된 데이타 유닛(110) 내에서의 데이타 루팅을 지정한다. " ari" 비트는 데이타 유닛(110)의 산술 논리 유닛(230)이 산술 연산 또는 부울 논리 연산에 사용되는 지를 지정한다. " 8 비트 ALU 코드" 필드는 상술한 방식으로 산술 논리 유닛(230)에 의해 수행된 연산을 지정한다. " src2" 필드(비트 41-39)는 산술 논리 유닛(230)용 제2 발생지로서 데이타 레지스터(200)들 중 하나를 지정한다. 데이타 유닛 포맷 B에 있어서, 산술 논리 유닛(230)용 제2 발생지는 " src2" 필드 내에 지정된 데이타 레지스터이다. 소정의 데이타 전송 포맷들은 제1 발생지에 대한 레지스터들의 뱅크들의 지정, 및 산술 논리 유닛(230)의 지정을 허가한다. 다른 관점에서 보면, 데이타 유닛 포맷 B는 데이타 유닛 포맷 A와 동일하다.

데이타 유닛 포맷 C는 비트 63=" 1" , 비트 44=" 1" , 및 비트 43=" 1" 로 인식된다. 데이타 유닛 포맷 C는 32 비트 즉시 필드를 이용하여 기초적인 산술 논리 유닛 연산을 특정한다. " 클래스" 필드는 표 18과 관련하여 이미 기술한 바와 같이, 데이타 유닛(11) 내에서의 데이타 루팅을 지정한다. " ari" 비트는 데이타 유닛(110)의 산술 논리 유닛(230)이 산술 논리 연산 또는 부울 논리 연산에 이용되는 지를 지정한다. " 8 비트 ALU 코드" 필드는 상술한 바와 같이, 산술 논리 유닛(230)에 의해 수행된 연산을 지정한다. 제1 발생지는 " src1" 필드에 의해 지정된 데이타 레지스터이다. 제2 발생지는 " 32-비트 imm." 필드(비트 31-0)의 32 비트 즉시 값이다. 이러한 데이타 유닛 포맷은 병렬 데이타 전송을 특정할 만한 여지가 없기 때문에, 아무것도 허가되지 않는다. " dstbank" 필드(비트 42-39)는 데이타 유닛(110)

내에서 레지스터들의 탱크를 지정한다. " dstbank" 필드에는 산술 논리 유닛(230)용 목적지로서 데이타 유닛(110)의 소정의 64 레지스터들을 지정하기 위해 " dst" 필드(비트 50 - 48)가 사용된다. " slbnk" 필드(비트 38 - 36)는 데이타 유닛(110) 내에서 레지스터들의 뱅크를 지정한다. 이러한 지정은 데이타 유닛(110)의 레지스터들의 하반부에 한정되고 산술 논리 유닛(230)용 제1 발생지로서 데이타 유닛(110) 내의 소정의 64 하반부 레지스터들을 지정하기 위해 " src1" 필드를 이용한다. 연산들은 후술하는 방식으로 " cond." 필드(비트 35 - 32)에 기초하여 조건부로 행해질 수 있다.

데이타 유닛 포맷 D는 비트 63=" 1" , 비트 44=" 0" 을 포함하고, " 클래스" 필드는 " 000" , 비트 59=" 1" (부울 논리 연산과 반대로서 산술 연산을 보통 선택함)이며, " 8 비트 ALU 코드" 의 비트(57, 55, 53 및 51)들은 올 " 0" 이다. 데이타 유닛 포맷 D는 비 - 산술 논리 유닛 연산들을 특정한다. " 연산" 필드(비트 43 - 39)는 비 - 산술 논리 유닛 연산을 지정한다. 양호한 실시예에서, 이러한 " 연산" 필드는 나중에 표 21에 도시되는 바와 같이, 디코드된다.

연산 필드 4 4 4 4 3 3 2 1 0 9	비-ALU 연산
0 0 0 0 0	연산 없음
0 0 0 0 1	아이들(idle)
0 0 0 1 0	글로벌 인터럽트를 인에이블
0 0 0 1 1	글로벌 인터럽트를 디스에이블
0 0 1 0 0	명령 페칭의 동기화를 로크시킴
0 0 1 0 1	명령 페칭의 동기화를 로크 해제시킴
0 0 1 1 0	예약
0 0 1 1 1	D 레지스터들을 우측 1만큼 회전시킴
0 1 0 0 0	널(null)
0 1 0 0 1	명령 실행을 정지시킴
0 1 0 1 0	예약
0 1 0 1 1	예약
0 1 1 0 0	에뮬레이터 인터럽트로 진행
0 1 1 0 1	에뮬레이터 인터럽트 1을 발생
0 1 1 1 0	에뮬레이터 인터럽트 2를 발생
0 1 1 1 1	예약
1 X X X X	예약

[표 21]

비 - 산술 논리 유닛 명령 널, 명령 실행 정지, 에뮬레이터 인터럽트로의 진행, 에뮬레이터 인터럽트 1의 발생, 및 에뮬레이터 인터럽트 2의 발생은 병렬 데이타 전송을 방해한다. 명령 워드 내에 특정된 소정의 병렬 데이타 전송은 무시된다. 다른 비 - 산술 논리 유닛 명령들은 병렬 데이타 전송을 허가한다.

데이타 유닛 포맷 E는 " 011" 로 되는 비트 63 - 61로 인식된다. 데이타 유닛 포맷 E는 병렬 산술 논리 유닛 및 승산 연산들을 특정한다. 이들 연산들은 이러한 포맷에서 특정된 6개의 피연산자들 때문에, " 6 피연산자 연산" 들로 인용된다. 양호한 실시예에서, " 연산" 필드(비트 60 - 57)는 표 22에서 나중에 도시되는 연산들을 특정한다. 심볼 " ‖ " 는 리스트된 연산들이 데이타 유닛(110) 내에서 병렬로 발생되는 것을 나타낸다. 16개의 연산들 중 11개만이 정의된다는 것을 주지해야 한다.

연산 필드 비트 6 5 5 5 0 9 8 7	6 피연산자 연산
0 0 0 0	MPYS ‖ ADD
0 0 0 1	MPYS ‖ SUB
0 0 1 0	MPYS ‖ EALUT
0 0 1 1	MPYS ‖ EALUF
0 1 0 0	MPYS ‖ ADD
0 1 0 1	MPYS ‖ SUB
0 1 1 0	MPYS ‖ EALUT
0 1 1 1	MPYS ‖ EALUF
1 0 0 0	EALU ‖ ROTATE
1 0 0 0	EALU% ‖ ROTATE
1 0 1 0	DIVI
1 0 1 1	예약
1 1 0 0	예약
1 1 0 1	예약
1 1 1 0	예약
1 1 1 1	예약

[표 22]

이들 연산에 대한 기호는 상기와 같이 정의된다. 검토해 보면: MPYS ‖ ADD는 병렬 부호 승산 및 가산을 지정하고; MPYS‖ SUB는 병렬 부호 승산 및 감산을 지정하며; MPYS ‖EALUT는 병렬 부호 승산 및 확장된 산술 논리 유닛 참(true) 연산을 지정하고; MPYS ‖EALUF는 병렬 부호 승산 및 확장된 산술 논리 유닛 거짓(false) 연산을 지정하며; MPYU ‖ ADD는 병렬 비부호 승산 및 가산을 지정하고; MPYU ‖SUB는 병렬 비부호 승산 및 감산을 지정하며; MPYU ‖ EALUT는 병렬 비부호 승산 및 확장된 산술 논리 유닛 참 연산을 지정하고; MPYU ‖ EALUF는 병렬 비 부호 승산 및 확장된 산술 논리 유닛 거짓 연산을 지정하며; EALU ‖ ROTATE는 원통형 회전기(235)의 출력이 분리 저장되어 있는 확장된 산술 논리 유닛 연산을 지정하고; EALU% ‖ ROTATE는 원통형 회전기(235)의 출력이 분리 저장되어 있는 마스크 발생기(239)에 의해 발생된 마스크를 사용하는 확장된 산술 논리 유닛 연산을 지정하며; DIVI는 제산에 사용된 제산 반복 연산을 지정한다. MPYx ‖ EALUT 명령에서의 산술 논리 유닛 연산은 데이타 레지스터 D0의 " EALU" 필드(비트 19-26)에 의해 선택되며, " A" 비트(비트 27)는 산술 연산, 또는 " FMOD" 필드(비트 31-28)에 의해 변형된 논리 연산을 선택한다. 이들 필드들의 코딩에 대해서는 이미 기술되었다. MPYx ‖ EALUF 명령에서의 산술 논리 유닛 연산은 " EALU" 필드 비트들의 감지가 반전되는 것을 제외하고는 똑같이 선택된다. EALU 및 EALU% 명령들에 대한 산술 논리 유닛 연산들은 똑같이 선택된다. 이들 연산들은 산술 논리 유닛 연산을 특정하기 위해 데이타 유닛(110)의 데이타 레지스터 D0의 일부를 채택한다. 데이타 레지스터 D0는 원하는 확장 산술 논리 유닛 연산 코드를 이용하여 프리-로드된다. DIVI 연산은 나중에 상세히 설명된다. 소정의 데이타 전송 포맷은 데이타 유닛(110)의 연산을 통해 병렬로 특정될 수 있다.

6 피연산자들은 데이타 유닛 포맷 E 내에서 특정된다. 4개의 발생지들 및 2개의 목적지들이 존재한다. " SRC3" 필드(비트 56-54)는 제3 발생지로서 데이타 레지스터(200)들 중 하나를 지정한다. 이는 승산 연산이 특정되는 경우에 승산기(220)용 제1 입력이고, 그렇지 않으면 원통형 회전기(235)의 원통 회전량이다. " dst2" 필드(비트 53-51)는 제2 목적지로서 데이타 레지스터(200)들 중 하나를 지정한다. 명령이 승산 연산을 특정하면, " dst2" 는 승산기(220)용 목적지이다. 그렇지 않으면, " dst2" 는 원통형 회전기(235)의 출력에 대한 목적지를 특정한다. " dst1" 필드(비트 50-48)는 산술 논리 유닛(230)용 목적지로서 데이타 레지스터 (200)들 중 하나를 지정한다. " src1" 필드(비트 47-45)는 산술 논리 유닛(230)용 제1 입력으로서 레지스터를 지정한다. 이러한 명령이 " Aslbank" 필드(비트 9-6)를 포함하는 전송 포맷(6 또는 10)을 포함하면, 이러한 레지스터 발생지는 " Aslbank" 필드가 레지스터 뱅크를 지정하고 "

src1" 필드가 레지스터 번호를 지정하는 데이타 유닛(110) 내의 소정의 레지스터일 수 있다. 그 경우, 이러한 데이타는 원통형 회전기(235)에 의해 회전될 수 없다. 이는 장거리 산술 논리 유닛 연산이라 불린다. 다른 전송 포맷에 있어서, " src1" 필드는 레지스터 번호에 의해 데이타 레지스터(200)들 중 하나를 지정한다. 전송 포맷(7, 8, 9 및 10)들은 레지스터 발생지가 상태 레지스터(210)의 " N" 비트에 기초하여 한 쌍의 데이타 레지스터(200)들로부터 조건부 선택되게 한다. 상태 레지스터(211)의 " N" 비트(비트 31)가 " 1" 이면, 지정된 데이타 레지스터는 산술 논리 유닛(230)용 제1 발생지로서 선택된다. " N" 비트가 " 0" 이면, 1 작은 데이타 레지스터(data register one less)가 선택된다. 이러한 옵션이 이용되면, " src1" 필드의 레지스터 수는 홀수이어야 한다. " src3" 필드(비트 56-54)는 승산기(220)용 제2 입력으로서 데이타 레지스터(200)들 중 하나를 지정한다. " src4" 필드(비트 44-42)는 승산기(220)용 제2 입력으로서 데이타 레지스터(200)들 중 하나를 지정한다.

표 23은 데이타 유닛 포맷 E에서 지원된 소정의 연산들에 대한 데이타 경로 접속을 도시한다. 입력 C는 멀티플렉서 Cmux(233)에 의해 선택된 입력 C 버스(243)에 공급된 신호이다. Maskgen은 멀티플렉서 Mmux(234)에 의해 선택된 마스크 발생기(239)에 공급된 신호이다. Rotate는 멀티플렉서 Smux(231)에 의해 선택된 원통형 회전기(235)의 제어 입력에 공급된 신호이다. 곱 좌측 이동은 신호로서 승산 이동 멀티플렉서 MSmux(225)에 의해 선택된 곱 좌측 이동기(224)의 제어 입력에 공급된다. DIVI 연산의 특별한 경우가 나중에 설명된다.

6 피연산자 연산	입력 C	maskgen	rotate	곱 좌측 이동
MPYS || ADD	-	-	0	0
MPYS || SUB	-	-	0	0
MPYS || EALUT	mask	D0(4-0)	D0(4-0)	D0(9-8)
MPYS || EALUF	mask	D0(4-0)	D0(4-0)	D0(9-8)
MPYS || ADD	-	-	0	0
MPYS || SUB	-	-	0	0
MPYS || EALUT	mask	D0(4-0)	D0(4-0)	D0(9-8)
MPYS || EALUF	mask	D0(4-0)	D0(4-0)	D0(9-8)
EALU	src4	-	src3	-
EALU%	mask	src4	src1	-

[표 23]

표 23에 리스트된 6 피연산자 명령들 전체에 있어서, 버스(201) 상에서 승산기(220)의 제1 입력은 " src3" 필드(비트 56-54)에 의해 지정된 레지스터이고, 버스(202) 상에서 승산기(220)의 제2 입력은 " src4" 필드(비트 44-42)에 의해 지정된 레지스터이며, 원통형 회전기(235)에 대한 입력은 " src1" 필드(비트 41-39)에 의해 지정된 레지스터이고, 입력 A 버스(241)에 대한 입력은 " src2" 필드(비트 47-45)에 의해 지정된 레지스터이다. 또한, 원통형 회전기(235)의 결과가 멀티플렉서 Bmux(227)을 통해 " dst2" 필드(비트 53-51)에 의해 지정된 레지스터에 세이브되는 대신에, 승산기(220)이 EALU 및 EALU% 명령에 사용되지 않는다는 것을 주지해야 한다.

DIVI 연산은 산술 논리 유닛(230)을 사용하고 승산기(220)는 사용하지 않는다. DIVI 연산은 비부호 제산을 위한 내부 루프에 사용될 수 있다. 부호 제산은 추가적인 셋-업으로 행하여도 좋고, 클린-업은 몫의 부호를 조정하기 위한 명령을 사용하여 행하여도 좋다. 컴퓨터로 구현하는 데 있어서 제산이 4가지 기본적인 산술 연산(가산, 감산, 승산 및 제산)들 중에서 가장 어렵다는 것은 본 분야에서 잘 알려져 있다.

DIVI 명령은 일단 적절히 셋업하면, 실행 파이프라인 스테이지 당 원하는 몫의 1 디지트를 계산하기 위해 데이타 유닛(110)의 하드웨어를 채택한다. DIVI 데이타 유닛 명령에는 조건부 데이타 전송(결과적으로 조건부 데이타 유닛 연산)을 지원하는 데이타 전송 포맷만이 사용될 수 있다. 이들 데이타 전송 포맷(7, 8, 9 및 10)들은 나중에 상세히 기술된다. 멀티플렉서 Amux(232)는 입력 A 버스(241)를 통해 산술 논리 유닛(230)에 공급하기 위해 산술 논리 유닛 제1 입력 버스(205) 상에서 " src2" 필드에 의해 지정된 데이타 레지스터(200b)로부터 데이타를 선택한다. 멀티플렉서 Imux(222)는 승산기 제2 입력 버스(202)에 공급하기 위해 상수 Hex " 1" 을 선택하고, 멀티플렉서 Smux(231)은 이러한 Hex " 1" 을 승산기 제2 입력 버스(202) 상에서 선택하여 회전 버스(244)에 공급한다. 데이타 레지스터(200)들 중" src1" 필드에 의해 지정된 데이타 레지스터로부터의 데이타는 원통형 회전기(235)에 공급된다. 이러한 레지스터는 단지 데이타 레지스터(D7, D5, D3 또는 D1)일 수 있고, 상태 레지스터(210)의 " N" 비트(비트 31)에 기초하여 멀티플렉서(215)에 의해 선택된 조건부 레지스터 발생지이다. 상태 레지스터(210)의 " N" 비트는 " 0" 이면, " src1" 필드에 의해 지정된 데이타 레지스터(200a)가 선택된다. 이러한 레지스터 선택은 반대 감지만 제외하고는, 산술 논리 유닛(230)을 채택하는 다른 명령으로 조건부 레지스터 선택을 제공하는데 사용된 동일한 하드웨어를 양호하게 이용한다. 이러한 레지스터 선택은 멀티플렉서를 통해 달성되거나, 레지스터의 특정화 동안에 레지스터 필드의 최하위 비트 대신에 상태 레지스터(210)의 " N" 비트의 반전을 이용함으로써 달성될 수도 있다. 상태 레지스터(210)의 " N" 비트가 " 1" 이면, " src1" 필드에 의해 지정된 레지스터보다 작은 데이타 레지스터(200c)가 선택된다. 원통형 회전기(235)는 이러한 데이타를 1 비트씩 좌측으로 회전시키고 입력 B 버스(241)를 통해 산술 논리 유닛(230)에 결과를 공급한다. 또한, 원통형 회전기(235)의 출력은 멀티플렉서 Bmux(227)을 통해 데이타 레지스터(200a)에 세이브되며, (회전 이전에) 다중 플래그 레지스터(211)의 비트 31은 원통형 회전기(235)의 출력의 비트 0을 대신한다. 목적지 레지스터는 " src1" 필드에 의해 지정된 레지스터이다. 멀티플렉서 Mmux(234)는 마스크 발생기(239)에 공급하기 위해 승산기 제2 입력 버스(202) 상에서 상수 Hex " 1" 을 선택한다. 멀티플렉서 Cmux(233)는 마스크 발생기(239)로부터 출력을 선택하여 입력 C 버스(243)를 통해 산술 논리 유닛(230)에 공급한다. 비트 0 캐리-인 발생기(246)는 (회전 이전에) 다중 플래그 레지스터(211)의 비트 31을 산술 논리 유닛(230)의 캐리-인 입력에 공급한다.

DIVI 명령 동안에, 산술 논리 유닛(230)은 Hex " A6" 의 함수 코드 비트 F7-F0을 수신한다. 이로 인해, 산술 논리 유닛(230)은 입력 A 버스(241) 및 입력 B 버스(242) 상의 입력들을 가산하고 0 확장된 결과를 좌측 이동시킨다. 이러한 좌측 이동은 Hex " 1" 입력에 응답하여 마스크 발생기(239)에 의해 공급된 마스크 때문에 1 비트씩 이루어진다. 이러한 기능은 A+B< 0< 으로 기호화된다. 산술 논리 유닛(230)의 결과는 " dst1" 필드에 의해 지정된 데이타 레지스터(200c)에 저장된다. 다중 플래그 레지스터(211)는 1 비트씩 회전되고, 다중 플래그 레지스터(211)의 최하위 비트(비트 0)는 산술 논리 유닛(230)에 의해 생성된 결과에 따라 설정된다. 이러한 동일 비트는 상태 레지스터(210)의 " N" 비트(비트 31)에 저장된다. OR 게이트(247)는 다중 플래그 레지스터(211)에 저장된 비트를 형성하고, 산술 논리 유닛(230)의 C_{out} 로부터 상태 레지스터(210)는 원통형 회전기(235)의 입력의 비트 31로 OR된다. 다른 상태 레지스터(210) 비트 " C" , " V" 및 " Z" 는 정상적으로 설정된다. 데이타 레지스터(200a) 내의 데이타가 X이고 데이타 레지스터(200b) 내의 데이타가 Y이고 데이타 레지스터(200c) 내의 데이타가 Z이면, DIVI 명령은 X=X< < 1 및 Z=X[n]Z+Y이다. " n" 기호는 " N" 상태 레지스터 비트에 기초한 레지스터 발생지 선택을 나타낸다.

DIVI 명령은 조건부 감산 및 이동 제산 알고리즘의 반복을 수행하도록 동작한다. 이러한 명령은 16 비트 젯수에 의해 제산된 32 비트 피젯수에 사용되어 16 비트 몫 및 16 비트 나머지를 생성할 수 있거나, 32 비트 젯수에 의해 제산된 64 비트 피젯수에 사용되어 32 비트 몫 및 32 비트 나머지를 생성할 수 있다. 64 비트 피젯수 경우에 있어서, 피젯수의 32 최상위 비트들은 초기에 데이타 레지스터(200a)에 저장되고, 32 최하위 비트들은 초기에 다중 플래그 레지스터(211)에 저장된다. 데이타 레지스터(200b)는 젯수의 역수를 저장한다. 제산 연산의 제1 반복에 있어서, DIVI 명령은 무조건적으로 실행되거나 상태 레지스터(210)의 " N" 비트는 " 0" 으로 설정된다. 원통형 회전기(235)로부터의 회전된 수는 데이타 레지스터(200a)에 저장된다. 원통형 회전기(235) 및 다중 플래그 레지스터(211)의 회전은 효과적으로 64 비트 피젯수를 한 자리 이동시킨다. 다중 플래그 레지스터 (211)의 최상위 비트가 64 피젯수의 다음 최상위 비트이고 산술 논리 유닛(230)의 캐리-인 입력에 적절히 공급된다는 것을 주지해야 한다. 데이타 레지스터(200a)에 저장된 양은 피젯수/실행 나머지(numerator/running remainder)로 불린다. 시험 감산의 결과는 데이타 레지스터(200c)에 저장된다.

시험 감산의 결과에 대해서는 2가지 경우가 있다. 초기 피젯수의 최상위 비트가 "1" 이거나 부(-) 젯수의 가산이 캐리를 발생하면, 대응하는 몫 비트는 "1" 이다. 이는 다중 플래그 레지스터(211)의 제1 비트 및 상태 레지스터(210)의 "N" 비트에 저장된다. 다음 시험 감산에 있어서, 멀티플렉서(215)는 상태 레지스터(210)의 "N" 비트 내의 "1" 에 의한 다음 반복을 위해 B 입력에 대한 데이타 레지스터(200c)를 선택한다. 따라서, 다음 시험 감산은 이전의 결과로부터 취해진다. OR 게이트(247)가 "0" 을 발생하면, 대응하는 몫 비트는 "0" 이다. 따라서, 다음 시험 감산은 좌측으로 한 자리 이동된 데이타 레지스터(200a)에 저장된 이전의 피젯수/실행 나머지로부터 취해진다. 이러한 반복은 DIVI의 32 사이클 동안 계속되어, 각각의 사이클 동안에 1 비트의 몫을 형성한다. 그 다음, 32 비트의 몫은 다중 플래그 레지스터(211) 내에서 완전히 형성된다. 32 비트 나머지는 상태 레지스터(210)의 "N" 비트의 상태에 따라 데이타 레지스터(200a) 또는 데이타 레지스터(200c)에서 나타난다.

32 비트 대 16 비트 제산에 대한 프로세스는 유사하다. 부 젯수는 데이타 레지스터(200b)에 저장되기 전에 16 자리 좌측으로 이동된다. 전체 피젯수는 데이타 레지스터(200a)에 저장된다. DIVI 명령은 16번만 반복되고, 이에 따라 몫은 다중 플래그 레지스터(211)의 16 최하위 비트들로 형성되며, 나머지는 상태 레지스터(210)의 "N" 비트의 상태에 따라 데이타 레지스터(200a) 또는 데이타 레지스터(200c)의 16 최상위 비트들로 형성된다.

이러한 기술은 데이타 유닛(110)에 이미 사용 가능한 하드웨어를 사용하여, 많은 마이크로프로세서 연산들의 오버헤드를 감소시킨다. DIVI 명령은 근본적으로 비부호 제산의 1 비트를 형성한다. 추가적인 소프트웨어는 부호 제산을 지원하기 위해 사용될 수 있다. 4개의 제산 서브루틴들은 비부호 반 워드(32 비트/16 비트)제산, 비부호 워드(64 비트/32 비트) 제산, 부호 반 워드(32 비트/16 비트) 제산 및 부호 워드(64 비트/32 비트) 제산을 위해 기록될 수 있다. 4개의 서브루틴들 각각은 다음의 3가지 위상들을 갖는다: 제산 준비; 단일 명령 루프 내의 제산 반복; 및 제산 랩-업(wrap-up). 루프 커널 내에는 제로 오버헤드 루핑 및 단일 64 비트 DIVI 명령을 이용하는 것이 양호하다.

각각의 제산 서브루틴의 제1 부분은 제산 준비이다. 이러한 제1 부분은 0의 젯수에 대한 테스트를 포함한다. 젯수가 "0" 이면, 제산 서브루틴은 취소되고 에러 조건은 주목된다. 다음에, 피젯수 및 젯수에 대하여 부호 비트가 설정된다. 부호 제산 서브루틴에 있어서, 몫의 부호는 피젯수 및 젯수의 부호 비트들의 OR로서 설정된다. 그 다음, 부호 제산에 있어서, 피젯수 또는 젯수가 부(-)이면, 이들은 정(+)의 수를 얻기 위해 부정된다. 피젯수는 선택된 홀수 데이타 레지스터와 다중 플래그 레지스터(211) 사이에 스핏(spit)된다. 워드 제산에 있어서, 피젯수의 상부 32 비트들은 선택된 데이타 레지스터에 저장되고, 피젯수의 하부 32 비트들은 다중 플래그 레지스터(211)에 저장된다. 반 워드 제산에 있어서, 피젯수의 전체 32 비트들은 선택된 데이타 레지스터에 저장된다. 반 워드 제산에 있어서, 다중 플래그 레지스터(211)의 미사용 하부 비트들은 제로 필(zero fill)된다. 반 워드 제산에 있어서, 젯수는 하부 비트들이 제로 필되어 있는 데이타 레지스터의 상부 16 비트들에 저장된다. 젯수는 산술 논리 유닛(230)이 가산에 의한 감산을 수행할 수 있도록 부정되어야 한다. 서브루틴은 몫이 오버플로되는 경우를 결정하기 위해 피젯수와 젯수의 최상위 비트들의 절대값들을 비교할 수 있다.

각각의 서브루틴의 중심은 단일 DIVI 명령을 포함하는 루프이다. 제로 오버 헤드 1 명령 루프를 초기화시키기 위해서는 레지스터 어드레스 LSRE2-LSRE0들 중 하나에 기록하는 것이 유리하다. 16번의 반복은 반 워드 몫들에 필요하고 32번의 반복은 워드 몫들에 필요하다. 루프 논리(720)가 0으로 감소되기 때문에, 루프 카운터는 원하는 수의 반복들보다 적은 수로 로드되어야 한다. 또한, 루프 논리 초기화 다음에 지연 슬롯들 내의 DIVI 명령을 2번까지 반복할 수 있게 한다. 이러한 루프 내의 단일 명령은 이미 기술된 DIVI 명령이다.

각각의 제산 서브루틴은 재산 랩-업으로 완성된다. 제산 랩-업은 다음 단계들을 포함한다. 몫은 다중 플래그 레지스터(211)로부터 데이타 레지스터로 이동된다. 몫의 부호가 부(-)이면, "1"은 데이타 레지스터 내의 몫에 가산되어 "1"의 보수 표현으로부터 2의 보수 표현으로 변환된다. 나머지가 필요하면, 상태 레지스터(210)의 "N" 비트에 기초하여 선택된다.

데이타 전송 포맷에 대한 설명, 및 제18도의 명령 워드의 병렬 데이타 전송 포맷의 여러 비트들과 필드들에 대한 설명에 대해서는 다음에 설명한다. 데이타 유닛 포멧의 비트들과 필드들의 설명과 관련하여 이미 기술된 바와 같이, 이들 비트들과 필드들은 명령 워드뿐만 아니라 명령 워드의 실행을 인에이블시키는 회로도 정의한다.

전송 포맷 1은 "00"이 아닌 비트 38-37,"000"이 아닌 비트 30-28, 및 "00"이 아닌 비트 16-15로 인식된다.
전송 포맷 1은 이중 병렬 데이타 전송 포맷으로 불린다. 전송 포맷 1은 메모리(20)의 2개의 독립적인 액세스들을 허가하는데, 그 하나는 글로벌 액세스이고, 다른 하나는 디지탈 영상/그래픽 프로세서에 대응하는 메모리 섹션들로 국한되는 로컬 액세스이다. "Lmode" 필드(비트 38-35)는 로컬 전송 모드로 인용되는데, 이는 어드레스 유닛(120)의 로컬 어드레스 유닛이 동작하는 방법을 특정한다. 이러한 필드는 표 24에 도시된 바와 같이 양호하게 디코드된다.

Lmode 필드 3 3 3 3 8 7 6 5	관계식	연산 설명
0 0 X X		연산 없음
0 1 0 0	*(An++=Xm)	변형있는 인덱스 레지스터의 사후-가산
0 1 0 1	*(An--=Xm)	변형있는 인덱스 레지스터의 사후-감산
0 1 1 0	*(An++=Imm)	변형있는 오프셋의 사후-가산
0 1 1 1	*(An--=Imm)	변형있는 오프셋의 사후-감산
1 0 0 0	*(An+Xm)	인덱스 레지스터의 사전-가산
1 0 0 1	*(An-Xm)	인덱스 레지스터의 사전-감산
1 0 1 0	*(An+Imm)	오프셋의 사전-가산
1 0 1 1	*(An-Imm)	오프셋의 사전-감산
1 1 0 0	*(An+=Xm)	변형있는 인덱스 레지스터의 사전-가산
1 1 0 1	*(An-=Xm)	변형있는 인덱스 레지스터의 사전-감산
1 1 1 0	*(An+=Imm)	변형있는 오프셋의 사전-가산
1 1 1 1	*(An-=Imm)	변형있는 오프셋의 사전-감산

[표 24]

"d" 필드(비트 34-32)는 로컬 버스 전송의 발생지 또는 목적지로 될 데이타 레지스터 D0-D7들 중 하나를 지정한다. "e" 비트(비트 31)는 "1"인 경우에는 확장된 부호를 지정하고, 그렇지 않고 "0"이면 로컬 데이타 전송을 위해 확장된 0을 지정한다. 이는 로컬 "siz" 필드(비트 30-29)가 전체 32 비트 워드 사이즈보다 작은 것을 나타날 때 레지스터 전송을 위해 메모리 내에서 동작 가능하다. "e" (비트 31) = "1"과, 다르게는 아무런 의미가 없는 "L" (비트 21) = "0"의 조합은 로컬 어드레스 유닛 산술 연산을 나타낸다. 로컬 "siz" 필드 (비트 30-29)는 표 25에 도시된 바와 같이, 양호하게 코드된다.

사이즈 필드 3 2 0 9	데이타 워드 사이즈
0 0	바이트 8 비트
0 1	반 워드 16 비트
1 0	전 워드 32 비트
1 1	예약

[표 25]

" s" 비트(비트 28)는 로컬 어드레스 인덱스 스케일링에 적용되는 스케일링 모드를 설정한다. " s" 비트가 " 1" 이면, 인덱스 레지스터 또는 명령 특정 오프셋으로부터 재호출될 수 있는 어드레스 계산에서의 인덱스는 " siz" 필드에 의해 표시된 사이즈로 스케일된다. " s" 비트가 " 0" 이면, 스케일링은 발생되지 않는다. 상술한 바와 같이, 이러한 인덱스 스케일링은 인덱스 스케일러(614)에서 발생한다. 선택된 데이타 사이즈가 8 비트(1바이트(byte))이면, " s" 비트의 상태와 관계없이 스케일링은 발생하지 않는다. 이 경우에서만, " s" 비트는 추가 오프셋 비트로서 사용될 수 있다. " Lmode" 필드가 오프셋을 지정하면, 이러한 " s" 비트는 오프셋의 최상위 비트로 되어 " Lim/x" 필드의 3 비트 오프셋 인덱스를 4 비트들로 변환시킨다. " La" 필드(비트 27-25)는 로컬 데이타 전송을 위해 어드레스 유닛(120)의 로컬 어드레스 유닛(620) 내의 어드레스 레지스터를 지정한다. " L" 비트(비트 21)는 로컬 데이타 전송이 메모리 레지스터간(L=" 1") 데이타 로드 전송이거나, 레지스터 메모리간 (L=" 0") 데이타 저장 전송인지를 나타낸다. " Lim/x" 필드(비트 2-0)는 " Lmode" 필드의 코딩에 따라 인덱스 레지스터의 레지스터 번호 또는 3 비트 오프셋을 특정한다.

글로벌 데이타 전송 연산은 로컬 데이타 전송의 코딩과 유사한 형태로 코드된다. " L" 비트(비트 17)는 글로벌 로드/저장 선택이다. 이러한 비트는 글로벌 데이타 전송이 로드로도 알려진 메모리 레지스터간(" L" =" 1") 전송인지 또는, 저장으로도 알려진 레지스터 메모리간(" L" =" 0") 전송인지를 결정한다. " Gmode" 필드(비트 16-13)는 로컬 전송 모드가 " Lmode" 필드에 의해 정의되는 방식으로 글로벌 전송 모드를 정의한다. 이러한 필드는 표 26에 도시된 바와 같이 양호하게 디코드된다.

Gmode 필드 1 1 1 1 6 5 4 3	관계식	연산 설명
0 0 X X		연산 없음
0 1 0 0	*(An++=Xm)	변형있는 인덱스 레지스터의 사후-가산
0 1 0 1	*(An--=Xm)	변형있는 인덱스 레지스터의 사후-감산
0 1 1 0	*(An++=Imm)	변형있는 오프셋의 사후-가산
0 1 1 1	*(An--=Imm)	변형있는 오프셋의 사후-감산
1 0 0 0	*(An+Xm)	인덱스 레지스터의 사전-가산
1 0 0 1	*(An-Xm)	인덱스 레지스터의 사전-감산
1 0 1 0	*(An+Imm)	오프셋의 사전-가산
1 0 1 1	*(An-Imm)	오프셋의 사전-감산
1 1 0 0	*(An+=Xm)	변형있는 인덱스 레지스터의 사전-가산
1 1 0 1	*(An-=Xm)	변형있는 인덱스 레지스터의 사전-감산
1 1 1 0	*(An+=Imm)	변형있는 오프셋의 사전-가산
1 1 1 1	*(An-=Imm)	변형있는 오프셋의 사전-감산

[표 26]

" reg" 필드(비트 12-10)는 레지스터를 식별한다. " reg" 필드는 저장의 경우에는 원시 레지스터의 수를 지정하거나, 로드의 경우에는 목적지 레지스터의 수를 지정한다. " Obank" 필드(비트 20-18)는 3개의 비트를 포함하고, 하부 64 레지스터들 내에 있는 레지스터들의 뱅크를 식별한다. 이들 레지스터들은 레지스터 뱅크 수들을 " OXXX" 형태로 갖는다. 3 비트 " Obank" 필드는 3 비트 " reg" 필드와 조합되어, 글로벌 데이타 전송을 위해 하부 64 레지스터들에서 소정의 레지스터를 데이타 발생지 또는 목적지로서 지정한다. " e" 비트(비트 9)는 " 1" 인 경우에는 확장된 부호를 지정하고, 그렇지 않고 " 0" 이면 글로벌 데이타 전송을 위해 확장된 0을 지정한다. 이는 글로벌 " siz" 필드(비트 8-7)가 전

체 32 비트 워드 사이즈보다 작은 것을 나타날 때 레지스터 전송을 위해 메모리 내에서 동작 가능하다. 이러한 " e" 비트는 데이타 사이즈가 32 비트인 경우에는 무시된다. " e" (비트9) = " 1" 과, " L" (비트 17) = " 0" 의 조합은 글로벌 어드레스 유닛 산술 연산을 나타낸다. 글로벌 " siz" 필드 (비트 8-7)는 표 27에 도시된 바와 같이, 양호하게 코드된다.

사이즈 필드		
8	7	데이타 워드 사이즈
0	0	바이트 8 비트
0	1	반 워드 16 비트
1	0	전 워드 32 비트
1	1	예약

[표 27]

" s" 비트(비트 6)는 글로벌 어드레스 인덱스 스케일링에 적용되는 스케일링 모드를 설정한다. " s" 비트가 " 1" 이면, 인덱스 레지스터 또는 명령 특정 오프셋으로부터 재호출될 수 있는 어드레스 계산에서의 인덱스는 " siz" 필드에 의해 표시된 사이즈로 스케일된다. " s" 비트가 " 0" 이면 스케일링은 발생하지 않는다. " siz" 필드가 8 비트의 데이타 사이즈를 지정하는 경우에는, 스케일링은 " s" 비트의 상태와 관계없이 발생하지 않는다. " Gmode" 필드가 오프셋을 지정하면, 이러한 " s" 비트는 오프셋의 최상위 비트로 되어 " Gim/x" 필드의 3 비트 오프셋 인덱스를 4 비트들로 변환시킨다. " Ga" 필드(비트 5-3)는 로컬 버스 전송을 위해 어드레스 유닛(120)의 글로벌 어드레스 유닛(610) 내의 어드레스 레지스터를 지정한다. " Gim/x" 필드(비트 24-22)는 " Gmode" 필드의 코딩에 따라 인덱스 레지스터의 레지스터 번호 또는 3 비트 오프셋을 특정한다. " Ga" 필드(비트 5-3)는 글로벌 데이타 전송의 메모리 어드레스를 계산할 때에 사용된 어드레스 레지스터의 레지스터 번호를 특정한다.

데이타 전송 포맷 2는 " 00" 이 아닌 비트 38-37, " 000" 인 비트 30-28, 및 " 00" 이 아닌 비트 16-15로 인식된다. 데이타 전송 포맷 2는 XY 패치 포맷으로 불린다. 데이타 전송 포맷 2는 어드레스 유닛(120)의 글로벌 및 로컬 어드레스 유닛 둘 다로부터 어드레스들을 멀티플렉싱하는 XY 패치 방식으로 메모리(20)의 어드레싱을 허가한다. " o" 비트(비트 34)는 외부 XY 패치 검출을 인에이블시킨다. " o" 비트가 " 1" 로 설정되면, 비트 " a" 및 " n" 들에 의해 특정된 연산들은 특정된 어드레스가 XY 패치 외부에 있는 경우에 수행된다. 그렇지 않고, " o" 비트가 " 0" 으로 설정되면, 연산들은 어드레스가 XY 패치 내부에 있는 경우에 수행된다. " a" 비트(비트 33)는 XY 패치 메모리 액세스 모드를 특정한다. " a" 비트가 " 1" 로 설정되면, 메모리 액세스는 어드레스가 XY 패치 내부 또는 외부에 있든지 간에 상관없이 수행된다. " a" 비트가 " 0" 으로 설정되면, 어드레스가 패치 외부(" o" 비트가 " 1" 인 경우) 또는 내부(" o" 비트가 " 0" 인 경우)에 있는 경우에, 메모리 액세스는 방해된다. " n" 비트(비트 32)는 XY 패치 인터럽트 모드를 특정한다. " n" 비트가 " 1" 로 설정되면, 어드레스가 패치 외부(" o" 비트가 " 1" 인 경우) 또는 내부(" o" 비트가 " 0" 인 경우)에 있는 경우에, XY 패치용의 인터럽트 플래그 레지스터 비트가 " 1" 로 설정된다. " n" 비트가 " 0" 으로 설정되면, XY 패치 인터럽트 요청 플래그는 설정되지 않는다.

다른 필드들은 상술한 동일 방식으로 정의된다. " Lmode" 필드는 표 24에 도시된 바와 같이 로컬 어드레스 계산 모드를 특정한다. 이러한 로컬 어드레스 계산은 " La" 필드에 의해 지정된 로컬 어드레스 레지스터와, 3 비트 비부호 오프셋, 또는 " Lim/x" 필드에 의해 지정된 로컬 인덱스 레지스터 중 어느 하나를 포함한다. " Gmode" 필드는 글로벌 어드레스 계산을 특정한다, 글로벌 비부호 3 비트 오프셋, 또는 " Gim/X" 에 의해 표시된 글로벌 인덱스 레지스터는 글로벌 어드레스를 형성하기 위해 " Ga" 필드에 의해 특정된 어드레스 레지스터와 조합된다. 4 비트 " 뱅크" 필드(비트 21-1

8)는 데이타 레지스터 뱅크를 식별하고, XY 패치 액세스를 위한 데이타 발생지 또는 목적지로서 소정의 레지스터를 지정하기 위해 레지스터 번호를 식별하는 3 비트 " reg" 필드와 조합된다. " L" 비트는 로드/저장 선택이다. 이러한 비트는 XY 패치 액세스가 로드로서 또한 알려진, 메모리 레지스터간(" L" =" 1")전송이거나, 저장으로서 또한 알려진, 레지스터 메모리간(" L" =" 0") 전송인지를 결정한다. " e" 비트는 " 1" 인 경우에는 확장된 부호를 지정하고, 그렇지 않고 " 0" 이면 제로 확장된다. 이는 " siz" 필드가 전체 32 비트 워드 사이즈보다 작은 것을 나타날 때 로드 연산(메모리 레지스터간 데이타 전송)으로 동작 가능하다. 이러한 " e" 비트는 데이타 사이즈가 32 비트인 경우에는 무시된다. " e" = " 1" 과 " L" = " 0" 의 조합은 패치된 어드레스 유닛 산술 연산을 나타낸다. " s" 비트는 글로벌 어드레스 인덱스 스케일링에 적용되는 스케일링 모드를 설정한다. " s" 비트가 " 1" 이면, 메모리로부터 재호출된 데이타가 " siz" 필드에 의해 표시된 사이즈로 스케일된다. " s" 비트가 " 0" 이면, 스케일링은 발생되지 않는다. 선택된 데이타 사이즈가 8 비트(1 바이트)이면, " s" 비트의 상태와 관계없이 스케일링은 발생하지 않는다. 이 경우에서만, " s" 비트는 3 비트 " Gim/x" 오프셋 인덱스를 4 비트들로 변환시키는 오프셋의 최상위 비트로서 사용된다.

데이타 전송 포맷 3은 " 00" 이 아닌 비트 38-37, " 0" 인 비트 24, 및 " 0000" 인 비트 16-13으로 인식된다. 데이타 전송 포맷 3은 이동 및 로컬 데이타 전송 포맷으로 불린다. 데이타 전송 포맷 3은 글로벌 포트 발생지 데이타 버스 GSRC(105) 및 글로벌 포트 목적지 데이타 버스 GDST(107)을 이용하여 레지스터간 이동과 병행으로 로컬 데이타 포트를 통해 데이타 레지스터(200)들 중 하나의 로드 또는 저장을 허가한다. 로컬 데이타 포트 연산은 상술한 방식으로 " Lmode" , " d" , " e" , " siz" , " s" , " La" , " L" 및 " Lim/x" 필드들에 의해 정의된다. 레지스터간 이동은 " srcbank" 필드(비트 9-6)에 의해 표시된 뱅크 및 " src" 필드(비트 12-10)에 의해 표시된 레지스터 번호에 의해 정의된 레지스터로부터, " dstbank" 필드(비트 21-18)에 의해 표시된 뱅크 및 " dst" 필드(비트 5-3)에 의해 표시된 레지스터 번호에 의해 정의된 레지스터까지이다.

데이타 전송 포맷 3은 디지탈 영상/그래픽 프로세서 상대적 어드레싱을 지원한다. " Lrm" 필드(비트 23-22)는 어드레싱 연산의 형태를 나타낸다. 이는 표 28에서와 같이 설정된다.

Lrm 필드 8	7	어드레싱 모드
0	0	정규 어드레싱
0	1	예약
1	0	데이타 메모리 베이스 어드레스 DBA
1	1	파라메타 메모리 베이스 어드레스 PBA

[표 28]

DBA의 명세는 로컬 어드레스 유닛(620)으로 하여금 대응하는 메모리의 베이스 어드레스를 생성하게 한다. 마찬가지로, PBA의 명세는 로컬 어드레스 유닛(620)으로 하여금 대응하는 파라메타 메모리의 베이스 어드레스를 생성하게 한다. 이러한 방식으로 생성된 베이스 어드레스는 인덱스 레지스터에 저장된 인덱스, 또는 표 24에 도시된 " Lmode" 필드로 특정된 소정의 어드레스 생성 연산들에서의 오프셋 필드와 조합될 수 있다.

또한, 이러한 데이타 전송 포맷은 명령 워드 생성을 지원한다. 레지스터간 이동의 목적지가 글로벌 어드레스 유닛(A15)의 제로 값 어드레스 레지스터이면, 명령 워드 디코딩 회로는 명령 워드 전송을 지정된 프로세서로 초기화시킨다. 이러한 명령 워드는 특별한 명령 워드 신호가 수반되는 글로벌 데이타 포트를 통해 크로스바(50)로 전송된다. 이로 인해, 인터프로세서 통신에서는 예를 들어, 소정의 디지탈 영상/그래픽 프로세서(71, 72, 73 및 74)들이 다른 프로세서에 인터럽트를 발생시킬 수 있게 되어 있다. 이러한 프로세스에 대해서는 상술되어 있다.

데이타 전송 포맷 4는 " 00" 이 아닌 비트 38-37, " 0" 인 비트 24, 및 " 0001" 인 비트 16-13으로 인식된다. 데이타 전송 포맷 4는 필드 이동 및 로컬 데이타 전송 포맷으로 불린다. 데이타 전송 포맷 4는 글로벌 포트 발생지 데이타 버스 GSRC(105) 및 글로벌 포트 목적지 데이타 버스 GDST(107)을 이용하여 레지스터간 필드 이동과 병행으로 로컬 데이타 포트를 통해 데이타 레지스터(200)들 중 하나의 로드 또는 저장을 허가한다. 로컬 데이타 포트 연산은 상술한 방식으로 " Lmode" , " d" , " e" (비트 31), " siz" (비트 30-29), " s" , " La" , " L" 및 " Lim/x" 필드들에 의해 정의된다.

레지스터간 필드 이동은 " src" 필드(비트 12-10)에 의해 표시된 레지스터 번호에 의해 정의된 레지스터로부터, " dstbank" 필드(비트 21-18)에 의해 표시된 뱅크 및 " dst" 필드(비트 5-3)에 의해 표시된 레지스터 번호에 의해 정의된 레지스터까지이다. " D" 비트(비트 6)는 " D" = " 1" 인 경우에 필드 이동이 필드 중복 이동이거나, " D" = " 0" 인 경우에 필드 추출 이동인지를 나타낸다. 필드 중복 이동에 있어서, 원시 레지스터의 최하위 8 비트들은 " siz" 필드(비트 8-7)가 바이트 사이즈를 나타내는 경우에 목적지 레지스터 내에서 4번 반복되고, 발생지 레지스터의 최하위 16 비트들은 " siz" 필드(비트 8-7)가 반 워드 사이즈를 나타내는 경우에 목적지 레지스터 내에서 2번 반복된다. " siz" 필드가 워드 사이즈를 나타내는 경우에, 원시 레지스터의 전체 32 비트들은 " D" 비트의 상태에 상관없이 중복되지 않고 목적지 레지스터에 전송된다. 필드 추출 이동에 있어서, " itm" 필드(비트 23-22)는 원시 레지스터로부터 추출될 작은 끝(little endian) 아이템 수를 나타낸다. 또한, 추출된 특별한 비트들은 " siz" 필드에 따른다. " siz" 필드(비트 8-7)의 데이타 사이즈가 바이트이면, " itm" 은 바람직한 바이트를 나타내는 0, 1, 2, 또는 3일 수 있다 " siz" 필드(비트 8-7)의 데이타 사이즈가 반 워드이면, " itm" 은 바람직한 반 워드를 나타내는 0 또는 1일 수 있다. " itm" 필드는 " siz" 필드(비트 8-7)가 워드인 경우에 무시된다. 원시 레지스터로부터 추출된 필드는 " e" 비트(비트 9)가 " 1" 인 경우에 확장된 부호이고, " e" 비트(비트 9)가 " 0" 인 경우에는 확장된 0이다. " e" 필드는 필드 중복 이동 중에 무시된다.

데이타 전송 포맷 5는 " 00" 이 아닌 비트 38-37, " 1" 인 비트 24, 및 " 00" 인 비트 16-15로 인식된다. 데이타 전송 포맷 5는 로컬 롱(long) 오프셋 데이타 전송 포맷으로 불린다. 데이타 전송 포맷 5는 글로벌 데이타 전송이 가능하지 않기 때문에 로컬 어드레스 유닛 내에 구성된 어드레스를 이용하여 글로벌 포트 메모리 액세스를 허가한다. 로컬 데이타 포트 연산은 상술한 방식으로 " Lmode" , " d" , " e" , " siz" , " s" , " La" 및 " L" 필드들에 의해 정의된다. 레지스터 발생지 또는 목적지는 " bank" 필드(비트 21-18) 내에 지정된 레지스터들의 뱅크 내에 있는 " reg" 필드(비트 34-32) 내에 지정된 레지스터 수에 대응한다. " Local Long Offset/x" 필드(비트 14-0)는 15 비트 로컬 어드레스 오프셋을 특정하거나, 3개의 최하위 비트들은 " Lmode" 필드에 의해 설정된 인덱스 레지스터를 특정한다. 프로그래머는 데이타 전송 포맷 5가 소정의 데이타 유닛 레지스터를 저장용 발생지 또는 로드용 목적지로서 허가하기 때문에, " Local Long Offset" 필드보다는 인덱스 레지스터를 이용하는 데이타 전송 포맷을 사용하기를 원한다. " Lmode" 필드는 이러한 필드가 오프셋 값 또는 인덱스 레지스터 번호를 포함하는 지를 나타낸다. 선택된 데이타 사이즈가 8 비트(1 바이트)이면, " s" 비트의 상태와 관계없이 스케일링은 발생하지 않는다. 이 경우에서만, " s" 비트는 15 비트 " Local Long Offset" 필드를 16 비트들로 변환시키는 오프셋의 최상위 비트로 된다. " Lrm" 필드(비트 23-22)는 표 28에 리스트되어 있는 정규 어드레스 연산, 데이타 메모리 베이스 어드레스 연산 또는 파라메타 메모리 베이스 연산을 특정한다.

데이타 전송 포맷 6은 " 00" 인 비트 38-37, " 00" 이 아닌 비트 16-15, 및 " 0" 인 비트 2로 인식된다. 데이타 전송 포맷 6은 글로벌 롱(long) 오프셋 데이타 전송 포맷으로 불린다. 데이타 전송 포맷 6은 어드레스 계산이 글로벌 어드레스 유닛 내에서 발생하는 것을 제외하고는, 데이타 전송 포맷 5와 유사하다. " bank" , " L" , " Gmode" , " reg" , " e" , " siz" , " s" 및 " Ga" 필드에 대해서는 상기 정의되어 있다. " Global Long Offset/x" 필드(비트 36-22)는 " Gmode" 필드에 따라 글로벌 오프셋 어드레스 또는 인덱스 레지스터를 특정한다. 이는 상술한 " Local Long Offset/x" 필드와 유사하다. " Grm" 필드(비트 1-0)는 어드레싱 연산의 형태를 나타낸다. 이는 표 29에 도시되어 있다.

Grm	필드	
1	0	어드레싱 모드
0	0	정규 어드레싱
0	1	예약
1	0	데이타 메모리 베이스 어드레스 DBA
1	1	파라메타 메모리 베이스 어드레스 PBA

[표 29]

이는 어드레스 계산이 글로벌 어드레스 유닛(610)에서 발생되는 것을 제외하고는 상술한 " Lrm" 필드와 동일한 형태로 동작한다.

데이타 전송 포맷 7은 " 00" 이 아닌 비트 38 - 37, " 0" 인 비트 24, 및 " 001" 인 비트 16 - 14로 인식된다. 데이타 전송 포맷 7은 비 - 데이타 레지스터 데이타 유닛 연산 및 로컬 데이타 전송 포맷으로 불린다. 데이타 전송 포맷 7은 산술 논리 유닛(230)용 발생지 및 산술 논리 유닛(230)용 목적지가 디지탈 영상/그래픽 프로세서(71) 상의 소정의 레지스터일 경우에 데이타 유닛 연산과 병행하여 로컬 포트 메모리 액세스를 허가한다. 로컬 데이타 포트 연산은 상술한 방식으로 " Lmode" , " d" , " e" , " siz" , " s" , " La" , " Lrm" , " L" 및 " Lim/x" 필드들에 의해 정의된다. " Adstbnk" 필드(비트 21 - 18)는 서술 논리 유닛 목적지에 대한 레지스터들의 뱅크를 특정한다. 이러한 필드는 데이타 유닛 포맷 A, B 및 C 내의 " dst" 필드, 및 데이타 유닛 포맷 D 내의 " dst1" 필드와 조합하여 레지스터 발생지를 특정한다. " Aslbank" 필드는 제1 산술 논리 유닛 발생지에 대한 레지스터들의 뱅크를 특정한다. 이는 데이타 유닛 포맷 A, B, C 및 D 내의 " src1" 필드와 조합하여 레지스터 발생지를 특정한다. 이들 데이타 유닛 연산들은 제1 발생지 및 목적지가 데이타 유닛(110)의 데이타 레지스터(200)일 필요는 없기 때문에 장거리 산술 논리 유닛 연산으로 불린다.

데이타 전송 포맷 8은 " 00" 인 비트 38 - 37, " 0" 인 비트 24, 및 " 0000" 인 비트 16 - 13으로 인식된다. 데이타 전송 포맷 8은 조건부 데이타 유닛 연산 및 조건부 이동 전송 포맷으로 불린다. 데이타 전송 포맷 8은 산술 논리 유닛(230)용 제1 발생지의 조건부 선택 및 산술 논리 유닛(230)의 결과의 조건부 저장을 허가한다. 조건부 산술 논리 유닛 연산들은 " cond." , " c" , " r" , " g" 및 " NCVZ" 필드들에 의해 정의된다.

" cond." 필드(비트 35 - 32)는 조건부 레지스터 발생지들로부터의 산술 논리유닛 연산, 및 산술 논리 유닛 결과의 조건부 저장을 정의한다. 이러한 필드는 표 20에 정의되어 있다. 이들 조건들은 상태 레지스터(210)의 " N" , " C" , " V" 및 " Z" 비트들에 기초하여 평가된다.

특정된 조건은 조건부 레지스터 발생지, 산술 논리 유닛(230) 결과의 조건부 저장 또는 조건부 레지스터간 이동을 결정할 수 있다. " c" 비트(비트 31)는 조건부 발생지 선택을 결정한다. " c" 비트가 " 0" 이면, 산술 논리 유닛(230)에 대한 제 1 발생지는 명령 워드의 데이타 유닛 포맷부의 " src1" 필드(비트 47 - 45)에 기초하여 무조건적으로 선택된다. " c" 비트가 " 1" 이면, 레지스터 발생지는 홀수 레지스터쌍과 짝수 레지스터쌍 사이에서 선택된다. 이 경우에, " src1" 필드가 홀수 번호 데이타 레지스터(200)을 특정해야 한다는 것을 주지해야 한다. 조건이 참이면, 특정된 레지스터는 산술 논리 유닛(230)용 제1 발생지로서 선택된다. 조건이 거짓이면, 특정된 데이타 레지스터보다 1 작은 대응하는 짝수 데이타 레지스터는 발생지로서 선택된다. 양호한 실시예는 상태 레지스터(210)의 " N" 비트에 기초하여 조건부 발생지 선택을 지원한다. 상태 레지스터(210)의 " N" 필드가 " 1" 이면, 지정된 데이타 레지스터는 산술 논리 유닛(230)용 제1 발생지로서 선택된다. 상태 레지스터(210)의 " N" 필드가 " 0" 이면, 1 작은 데이타 레지스터가 선택된다. 이러한 선택은 멀티플렉서에 의해, 또는 레지스터 번호의 최하위 비트 대신에 상태 레지스터(210)의 " N" 필드를 사용함으로써 행해질 수 있다. 양호한 실시예가 상태 레지스터(210)의 " N" 비트에 기초하여 조건부 발생지 선택만을 지원하며,

상태 레지스터(210)의 " C" , " V" 및 " Z" 비트들에 기초하여 조건부 발생지 선택을 제공할 수 있다.

데이타 전송 포맷 8은 산술 논리 유닛(230) 결과의 조건부 저장을 지원한다. " r" 비트(비트 30)는 결과의 저장이 조건적인 지를 나타낸다. " r" 비트가 " 1" 이면, 결과의 저장은 " cond " 필드의 조건에 기초하여 조건적으로 된다. " r" 비트가 " 0" 이면, 결과의 저장은 무조건적으로 된다. 조건적 결과 연산에 있어서, 상태 레지스터(210)의 상태 비트들은 무조건적으로 설정된다. 따라서, 이들 비트들은 결과가 저장되지 않은 경우에도 설정될 수 있다.

또한, 데이타 전송 포맷 8은 조건부 레지스터간 이동 연산을 허가한다. 조건은 조건부 데이타 유닛 연산을 특정하는 동일한 " cond." 필드에 의해 정의된다. 이동의 레지스터 데이타 발생지는 " srcbank" 필드(비트 9-6)에 의해 표시된 뱅크에 의해, 그리고 " src" 필드(비트 12-10)에 의해 표시된 레지스터 번호에 의해 정의된다. 레지스터 데이타 목적지는 " dstbank" 필드(비트 21-18)에 의해 표시된 뱅크에 의해, 그리고 " dst" 필드(비트 5-3)에 의해 표시된 레지스터 번호에 의해 정의된다. " g" 비트(비트 29)는 데이타 이동이 조건적인 지를 나타낸다 " g" 비트가 " 1" 이면, 데이타 이동은 " cond." 필드 내에 특정된 조건에 기초하여 조건적으로 된다. " g" 비트가 " 0" 이면, 데이타 이동은 무조건적으로 된다. 글로벌 어드레스 유닛의 제로 값 어드레스 레지스터(A15)의 목적지는 상술한 바와 같이 공통 워드 기록 연산을 발생한다. 따라서, 데이타 전송 포맷 8은 조건부 공통 워드 발생을 허가한다.

" N C V Z" 필드(비트 28-25)는 명령의 실행 중에 변경으로부터 보호되는 상태 비트들을 나타낸다. 상태 레지스터의 조건들은 다음과 같다: N 부정; C 캐리; V 오버플로; 및 Z 제로. 이들 비트들 중 하나 이상이 " 1" 로 설정되면, 상태 레지스터 내의 대응하는 조건 비트 또는 비트들은 명령의 실행 중에 변형으로부터 보호된다. 그렇지 않으면, 상태 레지스터(210)의 상태 비트들이 산술 논리 유닛(230)의 결과에 따라 정상적으로 설정된다.

데이타 전송 포맷 9는 " 00" 인 비트 38-37, " 0" 인 비트 24, 및 " 0001" 인 비트 16-13으로 인식된다. 데이타 전송 포맷 9는 조건부 데이타 유닛 연산 및 조건부 필드 이동 전송 포맷으로 불린다. 데이타 전송 포맷 9는 산술 논리 유닛(230)용 제1 발생지의 조건부 선택, 및 산술 논리 유닛(230)의 결과의 조건부 저장을 데이타 전송 포맷 8에서와 동일한 방식으로 허가한다. 조건부 산술 논리 유닛 연산들은 데이타 전송 포맷 8의 설명에서 주지된 바와 같이, " cond." , " c" , " r" 및 " N C V Z" 필드들에 의해 정의된다.

또한, 데이타 전송 포맷 9는 조건부 레지스터간 필드 이동을 지원한다. 조건은 조건부 데이타 유닛 연산을 특정하는 동일한 " cond." 필드에 의해 정의된다. 필드 이동의 발생지는 데이타 레지스터(200)들 중 하나이어야 한다. " src" 필드(비트 12-10)는 특별한 데이타 레지스터를 특정한다. 레지스터간 이동의 목적지는 " dstbank" 필드(비트 21-18)의 레지스터 뱅크에 의해, 그리고 " dst" 필드(비트 5-3)에 의해 표시된 레지스터 번호에 의해 정의된다. " g" (비트 29), " itm" (비트 23-22), " e" (비트 9), " siz" (비트 8-7) 및 " D" (비트 6) 필드들은 조건부 필드 이동의 파라메타들을 정의한다. " g" 비트는 " g" = " 0" 인 경우에 필드 이동은 무조건적이고, " g" = " 1" 인 경우에 필드 이동은 조건적이라는 것을 결정한다. " D" 비트는 필드 이동이 " D" =" 1" 인 경우에 필드 중복 이동인지 또는 " D" = " 0" 인 경우에 필드 추출 이동인 지를 나타낸다. 이들 옵션들에 대해서는 이미 기술되었다. 필드 추출 이동에 있어서, " itm" 필드(비트 23-22)는 " siz" 필드에 의해 특정된 데이타 사이즈에 기초하여 원시 레지스터로부터 추출될 작은 끝 아이템 수를 나타낸다. 원시 레지스터로부터 추출된 필드는 " e" 비트(비트 9)가 " 1" 인 경우에 확장된 부호이고, " e" 비트(비트 9)가 " 0" 인 경우에는 확장된 제로이다. " e" 필드는 필드 중복 이동 중에는 무시된다.

데이타 전송 포맷 10은 " 00" 인 비트 38-37, " 00" 이 아닌 비트 16-15, 및 " 1" 인 비트 2로 인식된다. 데이타 전송 포맷 10은 조건부 데이타 유닛 연산 및 조건부 글로벌 데이타 전송 포맷으로 불린다. 데이타 전송 포맷 10은 산술 논리 유닛(230)용 제1 발생지의 조건부 선택, 및 산술 논리 유닛(230)의 결과의 조건부 저장을 허가한다. 조건부 산술

논리 유닛 연산들은 데이타 전송 포맷 8의 설명에서 주지된 바와 같이, " cond." , " c" , " r" 및 " N C V Z" 필드들에 의해 정의된다.

또한, 데이타 전송 포맷 10은 글로벌 어드레스 유닛(610)을 통한 조건부 메모리 액세스를 지원한다. 조건부 메모리 액세스는 상술한 바와 같이, " g" , " Gim/x" , " bank" , " L" , " Gmode" , " reg" , " e" , " siz" , " s" , " Ga" 및 " G rm" 필드들에 의해 특정된다. " g" 비트(비트 29)는 데이타 이동이 이미 기술한 방식으로 조건적인 지를 나타낸다. " Gim/x" 필드는 " Gmode" 필드의 상태에 따라 인덱스 레지스터 번호 또는 오프셋 필드를 특정한다. " bank" 필드는 레지스터 뱅크를 특정하고, " reg" 필드는 글로벌 메모리 액세스의 레지스터 발생지 또는 목적지의 레지스터 번호를 특정한다. " L" 은 로드 연산(메모리 레지스터간 전송)을 " 1" 로 나타내고, 저장 연산(레지스터 메모리간 전송)을 " 0" 으로 나타낸다. " Gmode" 필드는 표 26에 도시된 글로벌 데이타 유닛(610)의 연산을 나타낸다. " e" 비트는 로드 연산에 대한 부호 또는 제로 확장을 나타낸다. " L" 필드의 " 0" 및 " e" 필드의 " 1" 은 어드레스 산술 연산을 생성한다. " siz" 필드는 표 27에 도시된 데이타 사이즈를 특정한다. " s" 비트는 인덱스가 상술한 바와 같이 데이타 사이즈로 스케일 되는지를 나타낸다. " Ga" 필드는 어드레스 계산에 사용된 어드레스 레지스터를 특정한다. " Grm" 필드는 표 29에 도시된 어드레싱 연산의 형태를 나타낸다.

데이타 전송 포맷 11은 " 00" 인 비트 38-37, " 0" 인 비트 24, 및 " 001" 인 비트 16-14로 인식된다. 데이타 전송 포맷 11은 조건부 비-데이타 레지스터 데이타 유닛 포맷으로 불린다. 데이타 전송 포맷 11은 메모리 액세스를 허가하지 않는다. 그 대신, 데이타 전송 포맷 11은 디지탈 영상/그래픽 프로세서(71) 내의 소정의 레지스터로서 산술 논리 유닛(230)의 1개의 발생지 및 목적지를 갖는 조건부 데이타 유닛 연산을 허가한다. 이들은 장거리 산술 논리 유닛 연산으로 불린다. " Aslbank" 필드(비트 9-6)는 명령의 데이타 유닛 포맷으로 된 " src1" 필드(비트 47-45)와 조합하여 제1 산술 논리 유닛 발생지를 정의하는 레지스터들의 뱅크를 특정한다. 따라서, 이러한 발생지는 디지탈 영상/그래픽 프로세서(71) 내의 소정의 레지스터일 수 있다. " Adstbnk" 필드(비트 21-18)는 데이타 유닛 포맷 A, B 및 C로 된 " dst" 필드(비트 50 48), 및 데이타 유닛 포맷 E로 된 " dst1" 필드(비트50-48)와 조합하여 산술 논리 유닛 목적지를 정의하는 레지스터들의 뱅크를 특정한다. 조건부 산술 논리 유닛 연산은 데이타 전송 포맷 8의 설명에 주지된 바와 같이 " cond." , " c" , " r" 및 " N C V Z" 필드들에 의해 정의된다.

" R" 비트(비트 0)는 리세트 비트이다. 이러한 " R" 비트는 단지 리세트시에 사용된다. " R" 비트는 스택 포인터 레지스터(A14)가 디지탈 영상/그래픽 프로세서 (71)의 리세트시에 초기화되는 지를 결정한다. 이러한 " R" 비트는 명령을 통해 사용자들에게 이용가능하지 않고 더 이상 설명되지 않는다.

단일 명령 내에서 가능한 상당수의 연산들로 인해, 단일 명령의 1개 이상의 연산이 동일한 목적지 데이타 레지스터(200)를 특정하는 것이 가능하다. 이러한 경우, 고정된 순서의 우선 순위는 공통적으로 특정된 목적지 레지스터 내의 결과를 세이브하는 연산을 결정한다. 이러한 고정된 순서의 우선 순위는 최상위 우선 순위로부터 최하위 우선 순위까지의 순서로 표 30에 도시된다.

우선 순위 랭크	연산
최상위	글로벌 어드레스 유닛 데이타 전송
중간 순위	로컬 어드레스 유닛 데이타 전송
최하위	데이타 유닛
	승산/ALU ⇒ 승산
	회전/ALU ⇒ ALU

[표 30]

따라서, 글로벌 어드레스 유닛 데이타 전송은 최상위 우선 순위를 갖고, 데이타 유닛 연산은 최하위 우선 순위를 갖는다. 1개 이상의 데이타 유닛 연산들이 단일 명령 중에 발생하기 때문에, 그러한 연산들에 대한 또 다른 우선 순위 랭크가 존재한다. 승산 연산 및 산술 논리 유닛 연산이 동일한 목적지 레지스터를 갖기 때문에, 승산 연산의 결과만이 저장된다. 이 경우, 상태 비트들은 중지된 산술 논리 유닛 연산에 의해 변경되지 않는다. 산술 논리 유닛 연산의 결과의 저장이 글로벌 또는 로컬 어드레스 유닛 데이타 전송과의 충돌 때문에 중지되면, 상태 비트들은 정상적으로 설정된다는 것을 주지해야 한다. 원통형 회전 결과 및 산술 논리 유닛 연산이 동일한 목적지를 갖기 때문에, 산술 논리 유닛 연산의 결과만이 저장된다. 이 경우, 상태 비트들은 완전한 산술 논리 유닛 연산을 위해 정상적으로 설정된다.

멀티프로세서 집적 회로(101)는 제19도에서 컬러 팩시밀리 장치의 부품으로서 도시된다. 모뎀(1301)은 송신 및 수신을 위해 전화선에 양방향 결합된다. 모뎀(1301)은 또한 버퍼(1302)와 통신하는데, 이 버퍼는 또한 영상 시스템 버스에 결합된다. 모뎀(1301)은 전화선을 통해 팩시밀리 신호를 수신한다. 모뎀(1301)은 이들 신호를 복조시키는데, 그 다음 이들 신호는 버퍼(1302)에 임시 저장된다. 전송 제어기(80)는 디지탈 영상/그래픽 프로세서(71)에 의한 처리를 위해 데이타를 데이타 메모리(22, 23 및 24)에 전송함으로써 버퍼(1302)로서 동작한다. 디지탈 영상/그래픽 프로세서(71)가 유입하는 데이타를 좇아갈 수 없는 경우에는, 전송 제어기(80)는 또한 버퍼(1302)로부터의 데이타를 메모리(9)로 전송할 수 있다. 디지탈 영상/그래픽 프로세서(71)는 유입하는 팩시밀리의 영상 데이타를 처리한다. 이는 영상 압축 해제, 잡음 감소, 에러 교정, 컬러 베이스 교정 등을 포함한다. 일단 처리되면, 전송 제어기(80)는 영상 데이타를 데이타 메모리(22, 23 및 24)들로 부터 비디오 랜덤 액세스 메모리(VRAM;1303)로 전송한다. 프린터 제어기(1304)는 프레임 제어기(90)의 제어 하에 영상 데이타를 재호출하여 이를 컬러 프린터(1305)로 공급하며, 이 프린터는 하드 카피를 형성한다.

제19도의 장치는 또한 컬러 팩시밀리를 송신할 수 있다. 영상화 장치(3)는 원고 서류를 스캔한다. 영상화 장치(3)는 프레임 제어기(90)의 제어 하에 동작하는 영상 캡처 제어기(4)로 원시 영상 데이타를 공급한다. 이러한 영상 데이타는 비디오 랜덤 액세스 메모리(1303)에 저장된다. 제19도에 도시된 실시예는 분리 비디오 랜덤 액세스 메모리들을 사용하는 제1도의 실시예에 비해, 영상 캡처 및 영상 디스플레이 둘 다를 위해 비디오 랜덤 액세스 메모리(1303)를 공유한다. 전송 제어기 (80)는 이러한 영상 데이타를 데이타 메모리(22, 23 및 24)들에 전송한다. 그 다음, 디지탈 영상/그래픽 프로세서(71)는 영상 압축, 에러 교정 중복, 컬러 베이스 교정 등을 위해 영상 데이타를 처리한다. 처리된 데이타는 팩시밀리 전송을 지원하도록 필요에 따라 전송 제어기(80)에 의해 버퍼(1302)에 전송된다. 상대적 데이타율에 따라, 전송 제어기(80)는 버퍼(1302)로의 전송 이전에 데이타를 메모리(9)에 임시 저장한다. 버퍼(1302) 내의 이러한 영상 데이타는 모뎀(1301)에 의해 변조되어 전화선을 통해 전송된다.

동일 시스템 내에 영상화 장치 및 컬러 프린터가 존재함으로써 이 시스템이 컬러 복사기로도 동작할 수 있게 한다. 이 경우, 데이타 압축 및 압축 해제는 필요하지 않을 수 있다. 그러나, 디지탈 영상/그래픽 프로세서(71)는 잡음 감소 및 컬러 베이스 교정에 계속 유용하다. 또한, 복사본이 원고보다 다양한 채색을 갖도록 디지탈 영상/그래픽 프로세시(71)는 고의적인 변환(deliberately shift) 컬러들로 프로그램되게 할 수 있다. 위장 채색(false coloring)으로 알려진 이러한 기술은 사용 가능한 프린터 컬러들의 동적 범위에 데이타의 동적 범위를 일치시키는데 유용하다.

(57) 청구의 범위

청구항 1.

제1다중비트 입력 신호를 수신하는 제1 입력(A);

제 2 다중비트 입력 신호를 수신하는 제 2 입력(B);

제3 다중비트 입력 신호를 수신하는 제3 입력(C);

명령원(21, 26, 31, 36; source of instruction);

상기 명령원으로부터 명령을 수신하고, 각각의 수신된 명령에 대응하여 상기 산술 인에이블 신호 및 함수 코드 비트 세트를 생성하는 명령 디코드 논리(250); 및

상기 제1, 제2 및 제3 입력에 접속되고, 상기 제1, 제2 및 제3 다중비트 입력 신호의 혼합된 산술 및 부울 조합(Boolean combination)인 다중비트 출력 신호를 출력에서 형성하는 조합 유닛(230) - 상기 산술 논리 유닛은 상기 명령 디코드 논리에 더 접속됨으로써, 상기 함수 코드 비트 세트를 수신하여, 상기 명령 디코드 논리로부터 수신된 함수 코드 비트 세트에 대응하여 상기 제1, 제2 및 제3 입력 신호의 선택된 산술, 부울 또는 혼합된 산술 및 부울 함수를 수행함 - 을 포함하는 형태의 데이타 처리 장치로서,

상기 산술 논리 유닛은

캐리 입력, 캐리 출력 및 결과 출력을 각각 가지는 복수의 단일 비트 회로 - 상기 복수의 단일 비트 회로는 다음의 단일 비트 회로의 상기 캐리 입력에 접속된 각각의 단일 비트 회로의 상기 캐리 출력과 직렬 연속으로 배치됨 - 를 포함하며,

각각의 단일 비트 회로는

제1 입력 신호(A_i)의 대응 비트를 수신하는 제1 비트 입력, 제2 입력 신호(B_i)의 대응 비트를 수신하는 제2 비트 입력 및 제3 입력 신호(C_i)의 대응 비트를 수신하는 제3 비트 입력과, 각각의 함수 코드 비트($\overline{F7-F0}$)를 수신하기 위한 복수의 함수 신호 입력을 가지는 부울 함수 발생기(403) - 상기 각각의 함수 코드 비트($\overline{F7-F0}$)가 상기 부울 함수 발생기의 조합을 정의하고, 상기 부울 함수 발생기는 다음과 같이 제1, 제2 및 제3 입력 신호의 상기 대응 비트의 선택된 조합에 대응하는 전파 신호(propagate signal)(P_i), 발생 신호(generate signal)(Gi) 및 킬 신호(kill signal)(Ki)를 생성함

$$P_i = F0\&(\sim A_i\&\sim B_i\&\sim C_i) \mid F1\&(A_i\&\sim B_i\&\sim C_i) \mid F2\&(\sim A_i\&\ B_i\&\sim C_i)$$
$$\mid F3\&(A_i\&\ B_i\&\sim C_i) \quad \mid F4\&(\sim A_i\&\sim B_i\&\ C_i) \mid F5\&(A_i\&\sim B_i\&\ C_i)$$
$$\mid F6\&(\sim A_i\&\ B_i\&\ C_i) \quad \mid F7\&(A_i\&\ B_i\&\ C_i)$$

$$G_i = A_i\&[(F0\&\sim F1\&\sim B_i\&\sim C_i) \mid (F2\&\sim F3\&\ B_i\&\sim C_i)$$
$$\mid (F4\&\sim F5\&\sim B_i\&\ C_i) \mid (F6\&\sim F7\&\ B_i\&\ C_i)]$$

$$K_i = \sim G_i\&\sim P_i$$

(여기서, P_i는 대응 비트에 대한 전파 신호이고, G_i는 대응 비트에 대한 발생 신호이고, K_i는 대응 비트에 대한 킬 신호이고, F0 F7은 함수 코드 비트이고, " & " 는 논리곱(logical AND)을 나타내고, " | " 는 논리합(logical OR)을 나타내고, " " 는 논리 부정(logical negation)을 나타냄) - 와,

상기 캐리 입력에 접속되고 또한, 상기 전파 신호(P_i)를 수신하기 위한 상기 부울 함수 발생기(403)에 접속된 결과 논리 회로(401) - 상기 결과 논리 회로는 상기 캐리 입력과 상기 전파 신호의 역의 배타적 부정 논리합(exclusive NOR)(431, 433)으로부터 비트 결과 신호를 형성함 - 와,

상기 캐리 입력에 접속되고 또한, 상기 전파 신호, 상기 발생 신호 및 상기 킬 신호를 수신하기 위하여 상기 부울 함수 발생기에 접속되며, 산술 인에이블 신호(A_{en})를 수신하는 캐리-아웃 논리 회로(402) - 상기 캐리-아웃 논리 회로는 다음과 같이 상기 캐리 출력에서 캐리-아웃 신호(C_{out})를 생성함

$$C_{out} = (P_i \ \& \ C_{in}) \mid (G_i \ \& \ A_{en}) \mid \sim(K_i \ \& \ A_{en})$$

(여기서, C_{out} 은 캐리 출력 신호이고, P_i는 대응 비트에 대한 전파 신호이고, G_i는 대응 비트에 대한 발생 신호이고, K_i는 대응 비트에 대한 킬 신호이고, A_{en} 은 산술 인에이블 신호이고, " & " 은 논리곱을 나타내고, " 1" 은 논리합을 나타내고, " " 은 논리 부정을 나타냄) -

를 포함하는 것을 특징으로 하는 데이타 처리 장치.

청구항 2.

제1항에 있어서,

상기 복수의 단일 비트 회로의 각각의 상기 부울 함수 발생기(403)는 2진 브랜칭 통과 게이트(binary branching pass gate)(451-458, 461-464, 471-473) 층의 논리 트리(logic tree)를 포함하고,

각각의 통과 게이트는 제1 통과 라인 및 제2 통과 라인과 제어 라인 및 반전 제어 라인을 가지며 - 적어도 3개의 입력 신호(A_i, B_i, C_i)의 대응 비트 신호는 각각의 제어 라인에 접속됨 - , 상기 제1 통과 라인을 제2 통과 라인에 선택적으로 결합함으로써 상기 제어 라인 및 상기 반전 제어 라인에서 수신된 신호에 따라서 신호를 전달하고,

상기 통과 게이트(451-458)의 하층의 제2 통과 라인은 상기 통과 게이트 (471-473)의 상층의 상기 제1 통과 라인에서 발생된 상기 적어도 3개의 입력 신호의 논리 조합을 제어하는 각각의 함수 코드 비트(F0-F7)를 수신하는 것을 특징으로 하는 데이타 처리 장치.

청구항 3.

제2항에 있어서,

상기 복수의 비트 회로의 각각에 포함된 상기 부울 함수 발생기(403)의 상기 논리 트리는

논리 트리 출력 노드에서 함께 접속된 제1 통과 라인을 가지며, 각각의 제어 라인에서 상기 제1 입력 신호(A_i)의 상기 대응 비트 신호와 상기 제1 입력 신호의 상기 대응 비트 신호의 역(474)을 수신하는 제1 통과 게이트 쌍(471, 472)과,

상기 제1 쌍의 제1 통과 게이트(471)의 상기 제2 통과 라인에 함께 접속된 제1 통과 라인을 가지며, 각각의 제어 라인에서 상기 제2 입력 신호(B_i)의 상기 대응 비트 신호와 상기 제2 입력 신호의 상기 대응 비트 신호의 역(465)을 수신하는 제2 통과 게이트 쌍(461, 462)과,

상기 제1 쌍의 제2 통과 게이트(472)의 상기 제2 통과 라인에 함께 접속된 제1 통과 라인을 가지며, 각각의 제어 라인에서 상기 제2 입력 신호(B_i)의 상기 대응 비트 신호와 상기 제2 입력 신호의 상기 대응 비트 신호의 역(465)을 수신하는 제3 통과 게이트 쌍(463, 464)과,

상기 제2 쌍의 제1 통과 게이트(461)의 상기 제2 통과 라인에 함께 접속된 제1 통과 라인을 가지며, 각각의 제어 라인에서 상기 제3 입력 신호(C_i)의 상기 대응 비트 신호와 상기 제3 입력 신호의 상기 대응 비트 신호의 역(459)을 수신하며, 각각의 제2 통과 라인에서 대응하는 제어 신호 쌍($\overline{F4}$-$\overline{F0}$)을 수신하는 제4 통과 상기 제2 쌍의 제2 통과 게이트(462)의 상기 제2 통과 라인에 함께 접속된 제1 통과 라인을 가지며, 각각의 제어 라인에서 상기 제3 입력 신호(C_i)의 상기 대응 비트 신호와 상기 제3 입력 신호의 상기 대응 비트 신호의 역(459)을 수신하며, 각각의 제2 통과 라인에서 대응하는 제어 신호 쌍($F6$-$F2$)을 수신하는 제5 통과 게이트 쌍(453, 454)과,

상기 제3 쌍의 제1 통과 게이트(463)의 상기 제2 통과 라인에 함께 접속된 제1 통과 라인을 가지며, 각각의 제어 라인에서 상기 제3 입력 신호(C_i)의 상기 대응 비트 신호와 상기 제3 입력 신호의 상기 대응 비트 신호의 역(459)을 수신하며, 각각의 제2 통과 라인에서 대응하는 제어 신호 쌍($F5$-$F1$)을 수신하는 제6 통과 게이트 쌍(455, 456)과,

상기 제3 쌍의 제2 통과 게이트(464)의 상기 제2 통과 라인에 함께 접속된 제1 통과 라인을 가지며, 각각의 제어 라인에서 상기 제3 입력 신호(C_i)의 상기 대응 비트 신호와 상기 제3 입력 신호의 상기 대응 비트 신호의 역(459)을 수신하며, 각각의 제2 통과 라인에서 대응하는 제어 신호 쌍($F7$-$F3$)을 수신하는 제7 통과 게이트 쌍(457, 458)을 포함하는 것을 특징으로 하는 데이타 처리 장치.

청구항 4.

제1항에 있어서,

상기 명령 디코드 논리(250)는 수신된 명령에 응답하여 상기 함수 코드 비트세트($\overline{F0}$-$\overline{F7}$)를 공급하여, 상기 산술 논리 유닛(230)이 상기 제1, 제2 및 제3 입력 신호 중 하나에 반응하지 않으므로써 나머지 입력 신호의 두 입력 함수를 수행하는 것을 특징으로 하는 데이타 처리 장치.

청구항 5.

제1항에 있어서,

상기 명령원(21, 26, 31, 36)은 각 함수 코드 비트($\overline{F0}$-$\overline{F7}$)에 대응하는 개별적인 비트를 갖는 함수 신호 필드(8-비트 ALU 코드)를 포함하는 적어도 하나의 함수 신호 명령을 공급하고,

상기 명령 디코드 논리(250)는 수신된 함수 신호 명령에 응답하여 상기 함수 신호 필드(8-비트 ALU 코드)의 상기 대응하는 개별적인 비트와 동등한 함수 코드 비트($F0$-$F7$)를 공급하는 것을 특징으로 하는 데이타 처리 장치.

청구항 6.

제1항에 있어서,

상기 명령원(21, 26, 31, 36)은 각 함수 코드 비트(57, 55, 53, 51)에 대응하는 개별적인 비트를 갖는 함수 신호 필드(8-비트 ALU 코드) 및 수정 비트 세트(58, 56, 54, 52)를 갖는 수정 필드를 포함하는 적어도 하나의 함수 신호 수정 명령을 공급하고,

상기 명령 디코드 논리(250)는 수신된 함수 신호 수정 명령에 응답하여 상기 수정 비트 세트(58, 56, 54, 52)에 대응하여 수정된 상기 함수 신호 필드(비트(57, 55, 53, 51))의 상기 대응하는 개별적인 비트와 동등한 함수 코드 비트($\overline{F0}$-$\overline{F7}$)를 공급하는 것을 특징으로 하는 데이타 처리 장치.

청구항 7.

제6항에 있어서,

상기 명령 디코드 논리(250)는 수신된 함수 신호 수정 명령에 응답하여 상기 제1, 제2 또는 제3 입력 신호 중 적어도 하나를 수정하는 것을 특징으로 하는 데이타 처리 장치.

청구항 8.

제1항에 있어서,

상기 명령 디코드 논리(250)에 접속되어, 각 함수 코드 비트($\overline{F0}$-$\overline{F7}$)에 대응하는 개별적인 비트를 갖는 함수 신호 필드(EALU 비트 19-26)를 포함하는 특수 함수 데이타 레지스터(D0)를 더 포함하고,

상기 명령원(21, 26, 31, 36)은 적어도 하나의 확장된 산술 논리 유닛 명령을 공급하고,

상기 명령 디코드 논리(250)는 수신된 확장된 산술 논리 유닛 명령에 응답하여 상기 특수 함수 데이타 레지스터(D0)의 상기 함수 신호 필드(EALU 비트 19-26)의 상기 대응하는 개별적인 비트와 동등한 함수 코드 비트($\overline{F0}$-$\overline{F7}$)를 공급하는 것을 특징으로 하는 데이타 처리 장치.

청구항 9.

제8항에 있어서,

상기 특수 함수 데이타 레지스터(D0)는 수정 비트 세트를 갖는 수정 필드 (FMOD 비트 28-31)를 더 포함하고,

상기 명령 디코드 논리(250)는 수신된 함수 신호 수정 명령에 응답하며 상기 수정 비트 세트(FMOD 비트 28-31)에 대응하여 수정된 상기 함수 신호 필드의 상기 대응하는 개별적인 비트와 동등한 함수 코드 비트(EALU 비트 19-26)를 공급하는 것을 특징으로 하는 데이타 처리 장치.

청구항 10.

제8항에 있어서,

상기 명령원(21, 26, 31, 36)을 적어도 하나의 확장된 산술 논리 유닛 거짓 명령(extended arithmetic logic unit false Instruction)을 공급하고,

상기 명령 디코드 논리(250)는 수신된 확장된 산술 논리 유닛 거짓 명령에 응답하여 상기 특수 함수 데이타 레지스터(D0)의 상기 함수 신호 필드(EALU 비트 19-26)의 상기 대응하는 개별적인 비트의 각각의 역과 동등한 함수 코드 비트(F0-F7)를 공급하는 것을 특징으로 하는 데이타 처리 장치.

청구항 11.

제1항에 있어서,

상기 명령원(21, 26, 31, 36)은 적어도 하나의 부호 의존 명령(sign dependent instruction)을 공급하고,

상기 명령 디코드 논리(250)는 수신된 부호 의존 명령에 응답하여, 상기 제 1, 제2 및 제3 다중비트 입력 신호(버스 206 비트 31) 중 선정된 입력 신호의 최상위 비트가 제1 상태이면 제1 함수 코드 비트 세트를 공급하고, 상기 선정된 입력 신호의 상기 선정된 비트가 제2 상태이면 제2 함수 코드 비트 세트를 공급하는 것을 특징으로 하는 데이타 처리 장치.

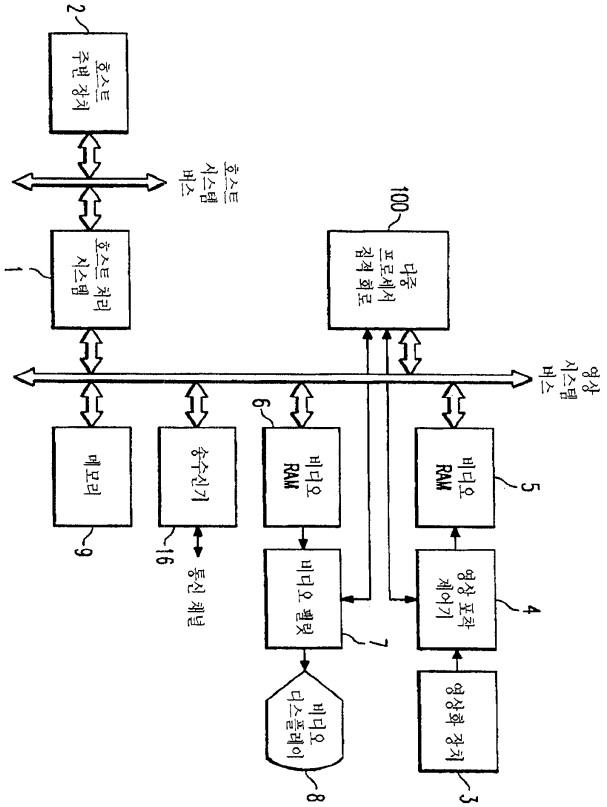
청구항 12.

제11항에 있어서,

상기 명령 디코드 논리(250)는 상기 제1 함수 신호 세트의 선택된 서브셋을 반전시킴으로써(551) 상기 제2 함수 코드 비트 세트를 공급하는 것을 특징으로 하는 데이타 처리 장치.

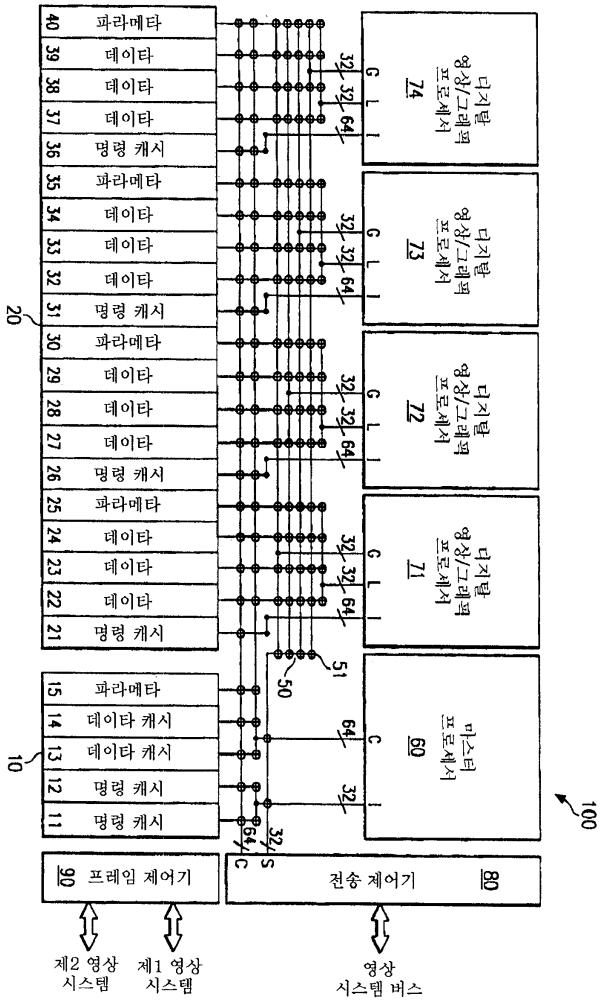
도면

도면 1

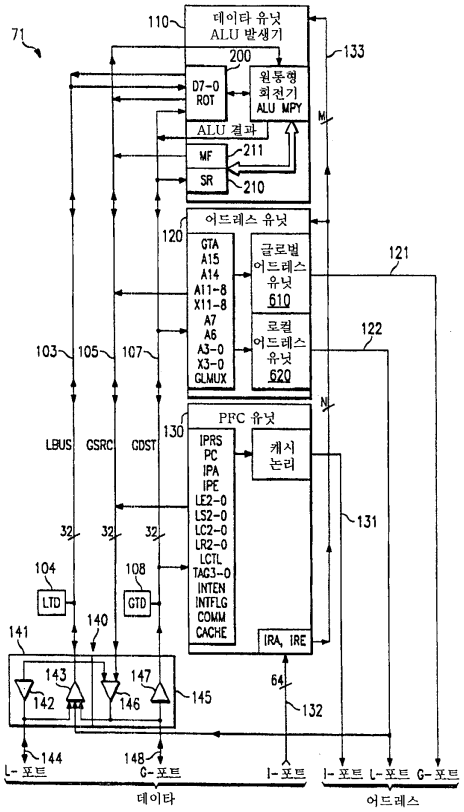

호스트 주변 장치
2
호스트 시스템 버스
호스트 처리 시스템
1
100
다중 프로세서 검색 회로
영상 시스템 버스
메모리
9
송수신기
16
통신 채널
비디오 RAM
6
비디오 믹싱
7
비디오 디스플레이
8
비디오 RAM
5
영상 포착 제어기
4
영상화 장치
3

p

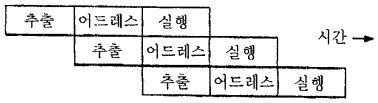
도면 2


40 파라메타
39 데이타
38 데이타
37 데이타
36 명령 캐시
35 파라메타
34 데이타
33 데이타
32 데이타
31 명령 캐시
30 파라메타
29 데이타
28 데이타
27 데이타
26 명령 캐시
25 파라메타
24 데이타
23 데이타
22 데이타
21 명령 캐시
20
15 파라메타
14 데이타 캐시
13 데이타 캐시
12 명령 캐시
11 명령 캐시
10
디지탈 영상/그래픽 프로세서 74
디지탈 영상/그래픽 프로세서 73
디지탈 영상/그래픽 프로세서 72
디지탈 영상/그래픽 프로세서 71
마스터 프로세서 60
100
90 프레임 제어기
전송 제어기 80
제2 영상 시스템
제1 영상 시스템
영상 시스템 버스
32
64
50
51
G
L
I
C
S

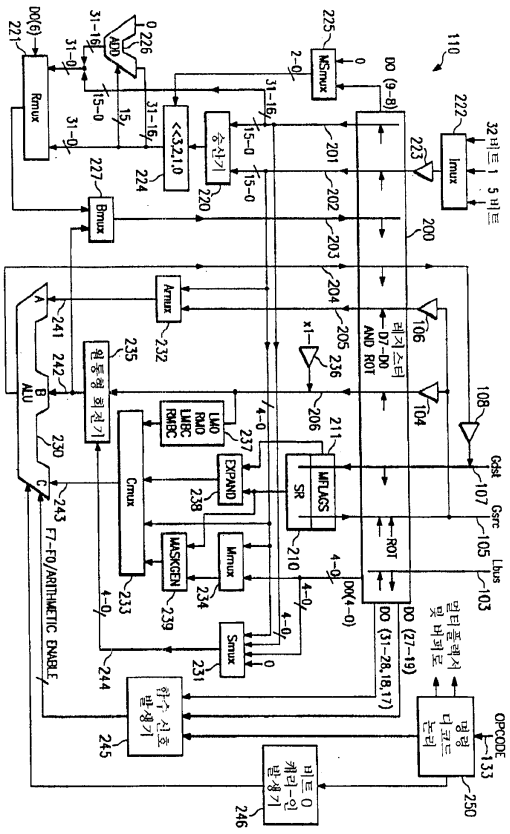
도면 3


71
110
데이타 유닛
ALU 발생기
200
133
D7-0
ROT
원통형
회전기
ALU MPY
ALU 결과
MF
211
SR
210
어드레스 유닛
120
GTA
A15
A14
A11-8
X11-8
A7
A6
A3-0
X3-0
GLMUX
글로벌
어드레스
유닛
610
121
로컬
어드레스
유닛
620
122
103
105
107
PFC 유닛
130
IPRS
PC
IPA
IPE
LE2-0
LS2-0
LC2-0
LR2-0
LCTL
TAG3-0
INTEN
INTFLG
COMM
CACHE
캐시
논리
131
IRA, IRE
LBUS
GSRC
GDST
32
32
32
104
LTD
108
GTD
140
141
143
142
147
146
145
64
132
144
148
L-포트
G-포트
I-포트
I-포트 L-포트 G-포트
데이타
어드레스

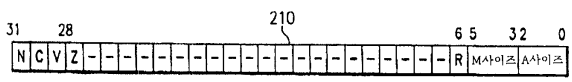
도면 4


추출
어드레스
실행
시간
추출
어드레스
실행
추출
어드레스
실행

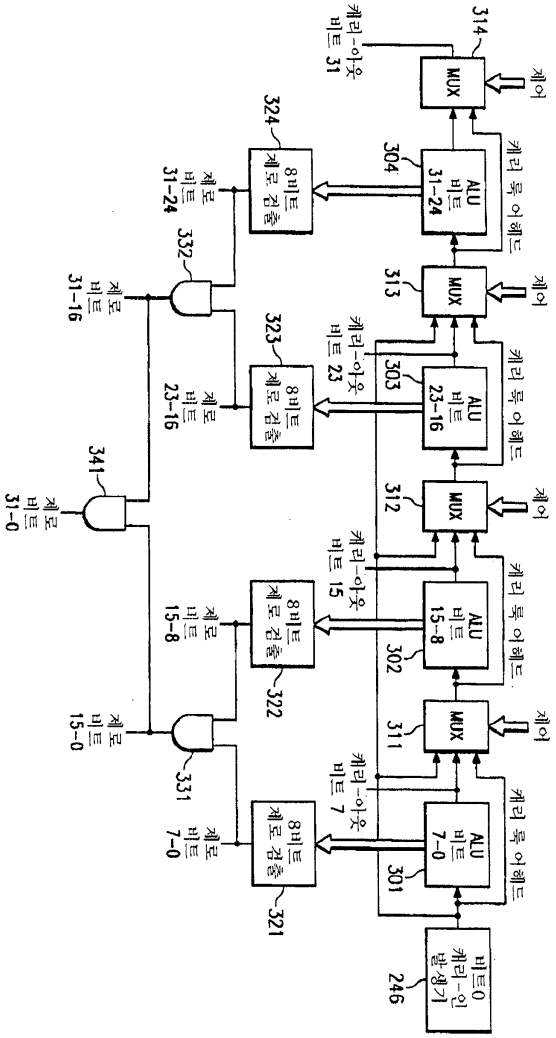
도면 5


Rmux
Bmux
Imux
MSmux
ADD
<<3,2,1,0
승산기
Amux
원통형 회전기
Cmux
EXPAND
MASKGEN
Mmux
Smux
MFLAGS
SR
레지스터
D7-D0
AND ROT
ROT
Gdst
Gsrc
Lbus
OPCODE
멀티플렉서 및 버퍼로
명령 디코드 논리
비트 0 캐리-인 발생기
F7-F0/ARITHMETIC ENABLE
ALU
32비트
5비트

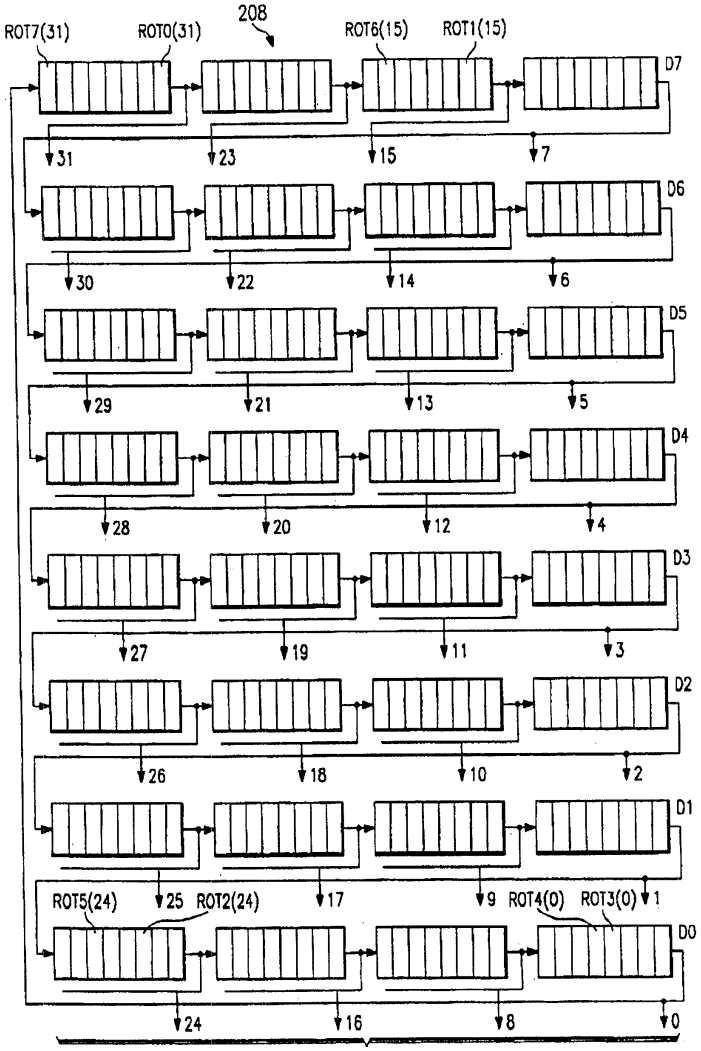
도면 6


31
28
210
6 5
32
0
N
C
V
Z
R
M사이즈
A사이즈

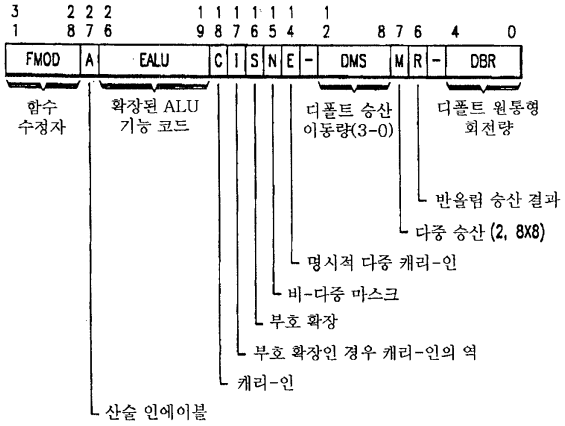
도면 7


314
MUX
제어
캐리-아웃 비트 31
304
ALU 비트 31-24
캐리 룩 어헤드
324
8비트 제로 검출
제로 비트 31-24
313
MUX
제어
303
ALU 비트 23-16
캐리 룩 어헤드
캐리-아웃 비트 23
323
8비트 제로 검출
제로 비트 23-16
332
제로 비트 31-16
312
MUX
제어
302
ALU 비트 15-8
캐리 룩 어헤드
캐리-아웃 비트 15
322
8비트 제로 검출
제로 비트 15-8
341
제로 비트 31-0
311
MUX
제어
301
ALU 비트 7-0
캐리 룩 어헤드
캐리-아웃 비트 7
321
8비트 제로 검출
제로 비트 7-0
331
제로 비트 15-0
246
비트 0 캐리-인 발생기

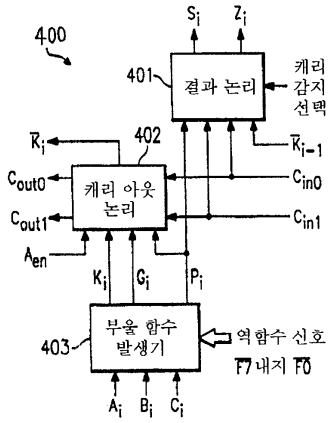
도면 8


208
ROT7(31)
ROT0(31)
ROT6(15)
ROT1(15)
D7
31
23
15
7
D6
30
22
14
6
D5
29
21
13
5
D4
28
20
12
4
D3
27
19
11
3
D2
26
18
10
2
D1
ROT5(24)
25
ROT2(24)
17
9
ROT4(0)
ROT3(0)
1
D0
24
16
8
0
회전 레지스터 비트 수

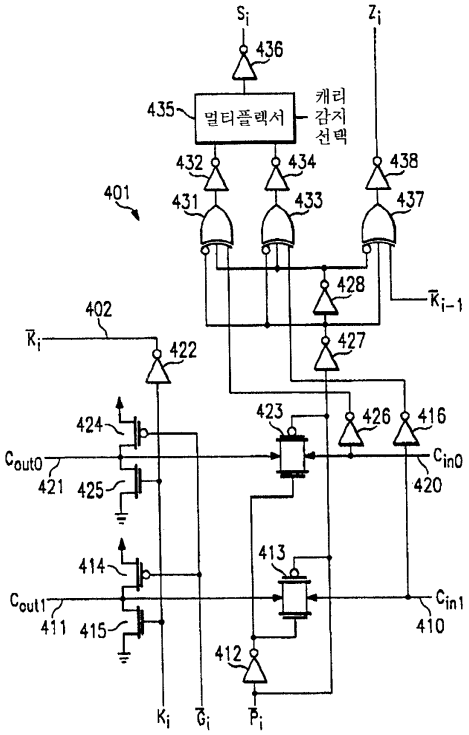
도면 9


31 28 27 26 19 18 17 16 15 14 12 8 7 6 4 0
FMOD
A
EALU
C
I
S
N
E
-
DMS
M
R
-
DBR
함수 수정자
확장된 ALU 기능 코드
디폴트 승산 이동량(3-0)
디폴트 원통형 회전량
반올림 승산 결과
다중 승산 (2, 8X8)
명시적 다중 캐리-인
비-다중 마스크
부호 확장
부호 확장인 경우 캐리-인의 역
캐리-인
산술 인에이블

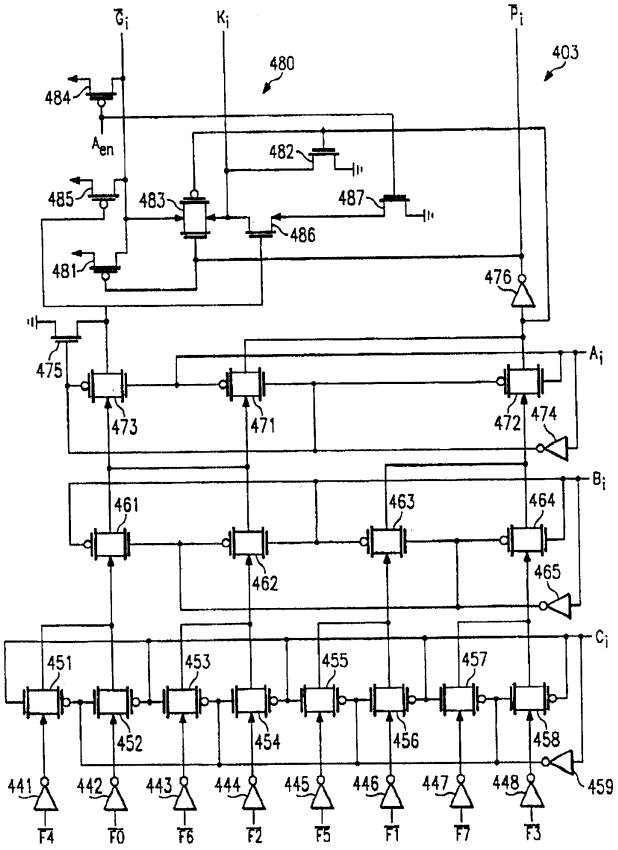
도면 10


400
401
결과 논리
S_i
Z_i
캐리 감지 선택
402
캐리 아웃 논리
$\overline{K}_i$
$\overline{K}_{i-1}$
C_{out0}
C_{out1}
C_{in0}
C_{in1}
A_{en}
K_i
G_i
P_i
403
부울 함수 발생기
역함수 신호
$\overline{F7}$ 내지 $\overline{F0}$
A_i
B_i
C_i

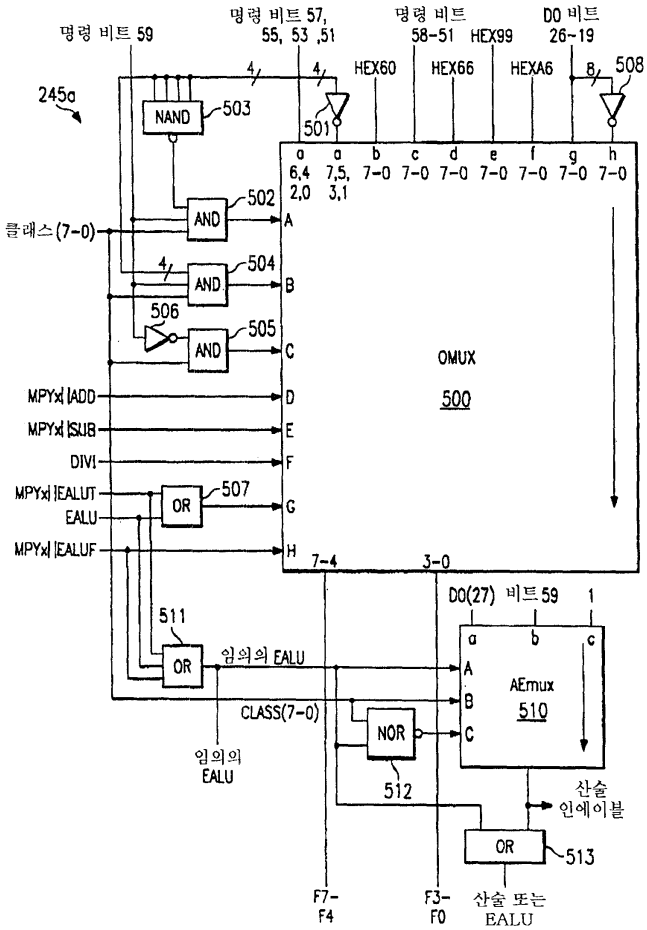
도면 11


S_i
Z_i
436
435
멀티플렉서
캐리 감지 선택
432
434
438
431
433
437
401
428
$\overline{K}_{i-1}$
402
$\overline{K}_i$
422
427
423
426
416
424
C_{out0}
421
425
C_{in0}
420
414
413
C_{out1}
411
415
C_{in1}
410
412
K_i
$\overline{G}_i$
$\overline{P}_i$

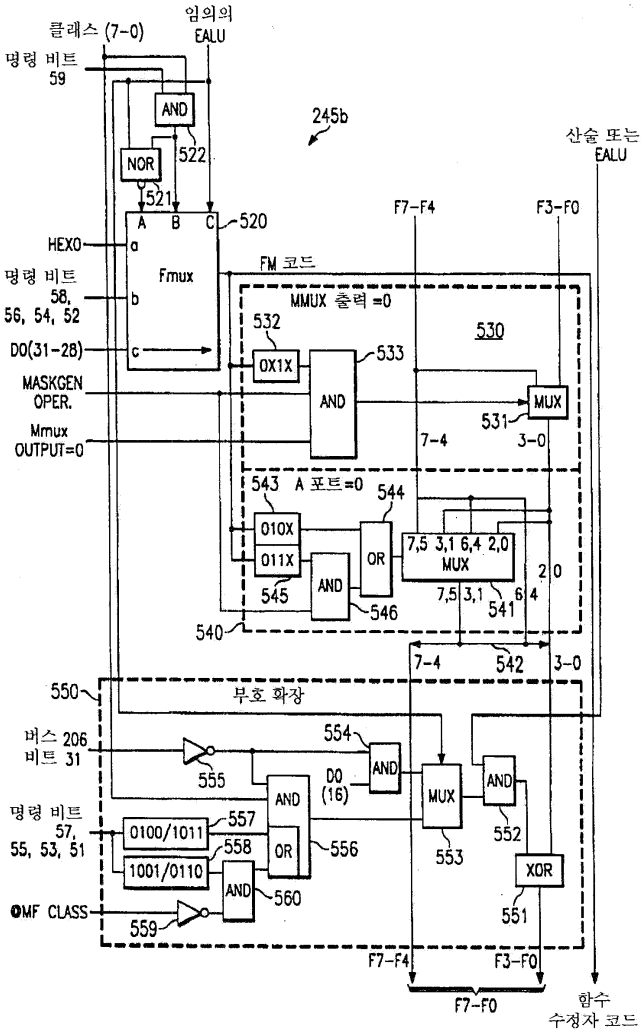
도면 12


$\overline{G}_i$
K_i
$\overline{P}_i$
480
403
484
A_{en}
482
485
483
487
486
481
476
475
A_i
473
471
472
474
B_i
461
463
464
462
465
C_i
451
453
455
457
452
454
456
458
459
441
442
443
444
445
446
447
448
F4
F0
F6
F2
F5
F1
F7
F3

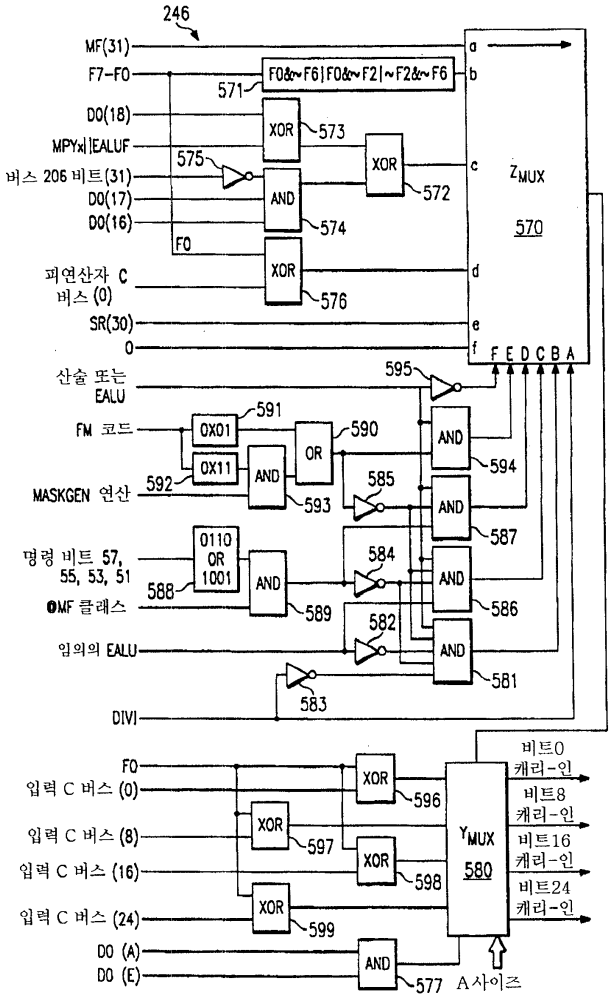
도면 13


명령 비트 59
명령 비트 57, 55, 53 ,51
명령 비트 58-51
HEX99
D0 비트 26-19
HEX60
HEX66
HEXA6
508
245a
NAND
503
501
AND
502
클래스(7-0)
504
506
505
OMUX
500
MPYx|ADD
MPYx|SUB
DIVI
MPYx|EALUT
EALU
MPYx|EALUF
OR
507
7-4
3-0
D0(27) 비트 59
511
임의의 EALU
AEmux
510
CLASS(7-0)
NOR
512
임의의 EALU
산술 인에이블
513
F7-F4
F3-F0
산술 또는 EALU

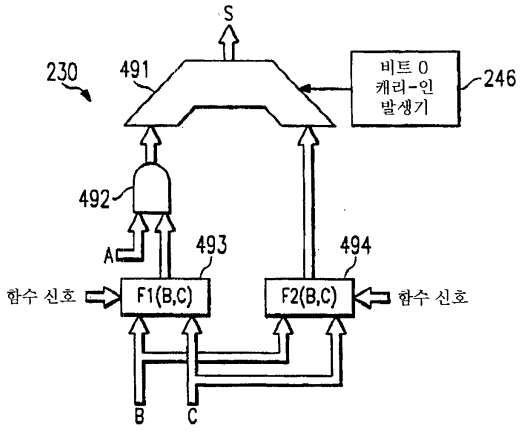
도면 14


클래스 (7-0)
임의의 EALU
명령 비트 59
AND
NOR
522
521
245b
산술 또는 EALU
A B C
520
F7-F4
F3-F0
HEX0
a
Fmux
FM 코드
명령 비트 58, 56, 54, 52
b
MMUX 출력 =0
532
530
DO(31-28)
c
OX1X
533
AND
MASKGEN OPER.
MUX
531
Mmux OUTPUT=0
7-4
3-0
543
A 포트=0
544
010X
011X
OR
7,5 3,1 6,4 2,0 MUX
AND
545
546
7,5 3,1
6,4
541
2,0
540
7-4
542
3-0
550
부호 확장
버스 206 비트 31
554
555
AND
DO (16)
MUX
AND
552
명령 비트 57, 55, 53, 51
0100/1011
557
1001/0110
558
OR
556
553
XOR
551
OMF CLASS
AND
560
559
F7-F4
F3-F0
F7-F0
함수 수정자 코드

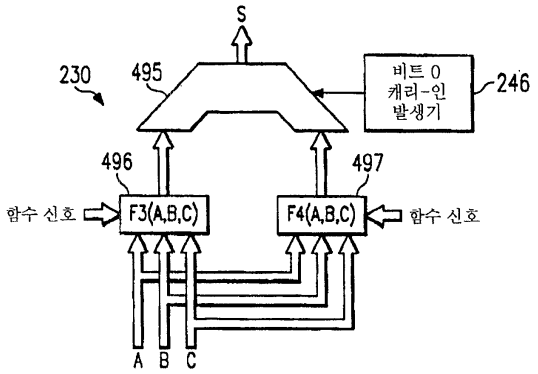
도면 15


246
MF(31)
F7-F0
571
F0&~F6|F0&~F2|~F2&~F6
DO(18)
XOR
573
MPYxl|EALUF
575
XOR
572
버스 206 비트(31)
DO(17)
DO(16)
AND
574
ZMUX
570
F0
XOR
576
피연산자 C 버스 (0)
SR(30)
0
a
b
c
d
e
f
F E D C B A
산술 또는 EALU
595
FM 코드
0X01
591
OR
590
AND
594
0X11
592
AND
593
585
AND
587
MASKGEN 연산
명령 비트 57, 55, 53, 51
0110 OR 1001
584
AND
586
588
AND
589
OMF 클래스
582
AND
581
임의의 EALU
583
DIVI
F0
입력 C 버스 (0)
XOR
596
비트0 캐리-인
입력 C 버스 (8)
XOR
597
YMUX
580
비트8 캐리-인
XOR
598
입력 C 버스 (16)
비트16 캐리-인
XOR
599
입력 C 버스 (24)
비트24 캐리-인
DO (A)
DO (E)
AND
577
A사이즈

도면 16


S
230
491
비트 0
캐리-인
발생기
246
492
A
493
494
함수 신호
F1(B,C)
F2(B,C)
함수 신호
B
C

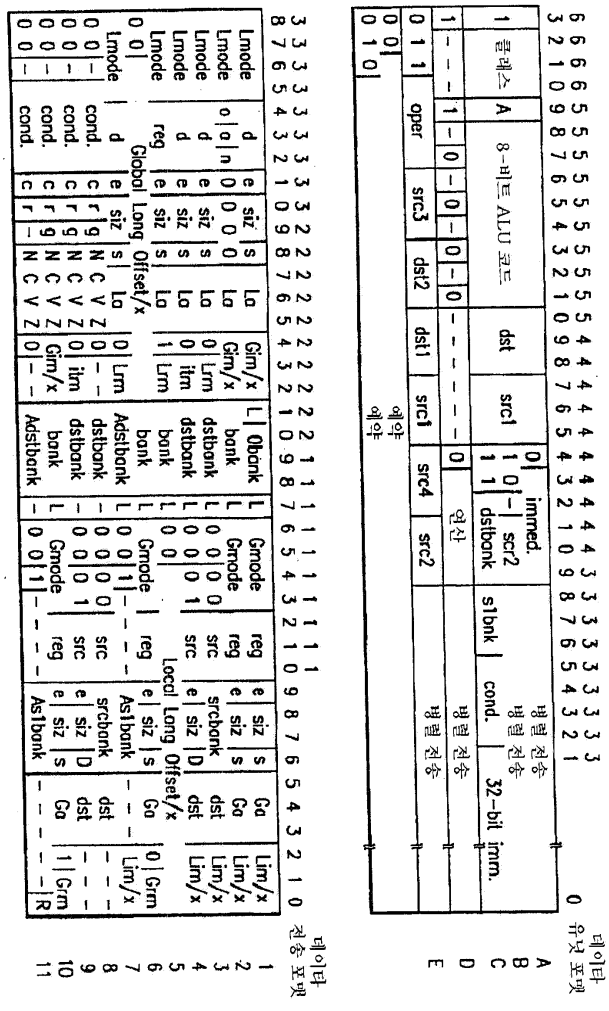
도면 17


S
230
495
비트 0
캐리-인
발생기
246
496
497
함수 신호
F3(A,B,C)
F4(A,B,C)
함수 신호
A
B
C

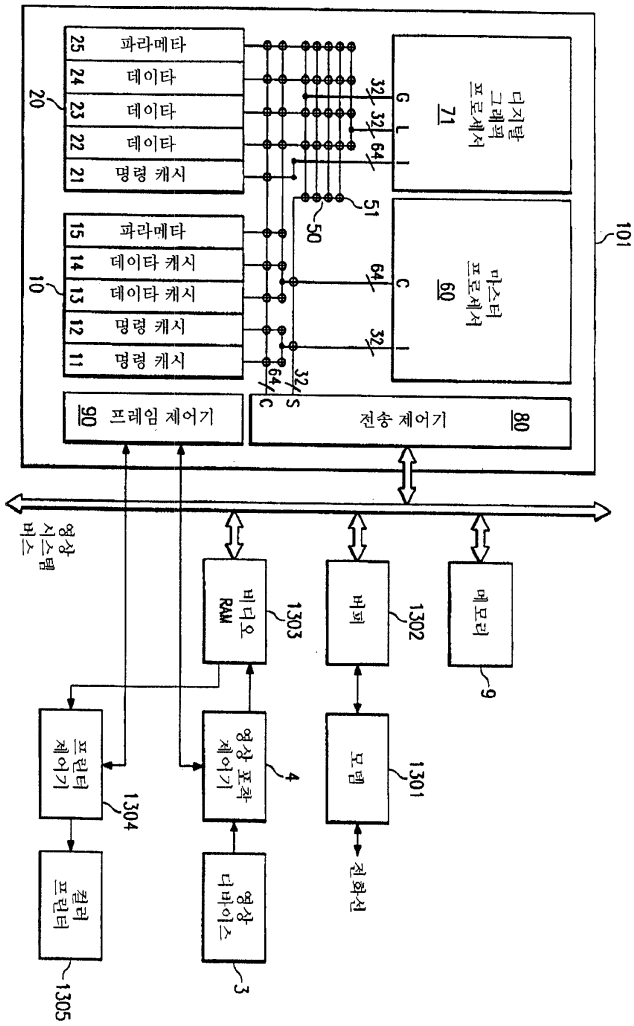
도면 18

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	0	데이타 유닛 포맷
1	클래스			A	8-비트 ALU 코드								dst			src1			0	immed.					병렬 전송									A
																			1	0	-	scr2			병렬 전송									B
																			1	1	dstbank				s1bnk		cond.			32-bit imm.				C
1	-	-	-	1	-	0	-	0	-	0	-	0	-	-	-	-	-	-	0	연산					병렬 전송									D
0	1	1	oper				src3			dst2			dst1			src1			src4			src2			병렬 전송									E
0	0	예약																																
0	1	0	예약																															

38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	데이타 전송 포맷
Lmode				d			e	siz		s	La			Gim/x			L	Obank			L	Gmode				reg			e	siz		s	Ga			Lim/x			1
Lmode				o	a	n	0	0	0	0	La			Gim/x				bank			L	Gmode				reg			e	siz		s	Ga			Lim/x			2
Lmode				d			e	siz		s	La			0	Lrm			dstbank			L	0	0	0	0	src			srcbank				dst			Lim/x			3
Lmode				d			e	siz		s	La			0	itm			dstbank			L	0	0	0	1	src			e	siz		D	dst			Lim/x			4
Lmode				reg			e	siz		s	La			1	Lrm			bank			L	0	0	Local Long Offset/x															5
0	0	Global Long Offset/x																bank			L	Gmode				reg			e	siz		s	Ga			0	Grm		6
Lmode				d			e	siz		s	La			0	Lrm			Adstbank			L	0	0	1	-	-	-	-	As1bank				-	-	-	Lim/x			7
0	0	-	cond.				c	r	g	N	C	V	Z	0	-	-		dstbank			-	0	0	0	0	src			srcbank				dst			-	-	-		8
0	0	-	cond.				c	r	g	N	C	V	Z	0	itm			dstbank			-	0	0	0	1	src			e	siz		D	dst			-	-	-		9
0	0	-	cond.				c	r	g	N	C	V	Z	Gim/x				bank			L	Gmode				reg			e	siz		s	Ga			1	Grm		10
0	0	-	cond.				c	r	-	N	C	V	Z	0	-	-		Adstbank			-	0	0	1	-	-	-	-	As1bank				-	-	-	-	-	R		11



도면 19


101
20
25 파라메타
24 데이타
23 데이타
22 데이타
21 명령 캐시
10
15 파라메타
14 데이타 캐시
13 데이타 캐시
12 명령 캐시
11 명령 캐시
디지탈 그래픽 프로세서 71
마스터 프로세서 60
90 프레임 제어기
전송 제어기 80
영상 시스템 버스
비디오 RAM 1303
버퍼 1302
메모리 9
프린터 제어기 1304
영상 포착 제어기 4
모뎀 1301
컬러 프린터 1305
영상 디바이스 3
전화선