(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2009-0004394
(43) 공개일자 2009년01월12일

(51) Int. Cl.
G06F 15/16 (2006.01) *G06F 9/30* (2006.01)
G06F 9/06 (2006.01)
(21) 출원번호 10-2007-7009924
(22) 출원일자 2007년04월30일
심사청구일자 없음
번역문제출일자 2007년04월30일
(86) 국제출원번호 PCT/US2007/004030
국제출원일자 2007년02월16일
(87) 국제공개번호 WO 2007/098006
국제공개일자 2007년08월30일
(30) 우선권주장
11/355,495 2006년02월16일 미국(US)
(뒷면에 계속)

(71) 출원인
브이엔에스 포트폴리오 엘엘씨
미국 캘리포니아 95014 쿠퍼티노 슈트 500 스티븐스 크리크 불러바드 20400
(72) 발명자
무어 찰스 에이치.
미국 캘리포니아 96125 시에라 시티 그린 로드 110
(74) 대리인
박장원

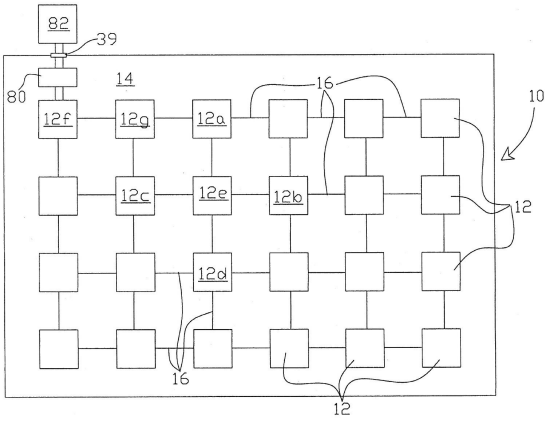
전체 청구항 수 : 총 21 항

(54) 입력 소스로부터의 직접적인 명령어들의 실행

(57) 요 약

컴퓨터 어레이(10)는 다수의 컴퓨터(12) 구비하고 있다. 컴퓨터(12)는 상호간에 비동기적으로 교신하며, 컴퓨터(12) 자체도 대개 내부적으로 각각 비동기적으로 동작한다. 한 컴퓨터(12)가 다른 컴퓨터(12)와 통신할 경우, 다른 컴퓨터가 트랜잭션을 종료할 때까지, 컴퓨터는 수면 상태로 들어가며 이에 따라 전력이 절약되고 열 방출을 감소한다. 수면중인 컴퓨터(12)는 명령어나 데이터를 기다린다. 명령어의 경우, 수면중인 컴퓨터는 깨어나서 명령어를 저장하거나 즉각 실행할 수도 있다. 후자의 경우, 명령어는 수신될 경우 명령어 레지스터에 저장되고 우선적으로 메모리에 저장할 필요 없이 레지스터에서 즉각 실행된다. 명령어는 일련의 동작을 반복적으로 실행하는 마이크로-루프(micro-loop)을 포함한다. 한 응용 실시예에서는, 수면중인 컴퓨터(12)가 입력에 의해 깨어나 동작을 개시하게 된다. 이 동작은 다른 발명의 경우 활동 중인 컴퓨터에 인터럽트를 필요로 하게 되는데 이 발명에서는 필요 없다.

대 표 도



(30) 우선권주장
11/355,513 2006년02월16일 미국(US)
11/441,784 2006년05월26일 미국(US)
11/441,812 2006년05월26일 미국(US)
11/441,818 2006년05월26일 미국(US)
60/788,265 2006년03월31일 미국(US)
60/797,345 2006년05월03일 미국(US)

특허청구의 범위

청구항 1

일련의 명령어를 실행하는 컴퓨터에 있어서,

실행될 명령어 그룹을 일시적으로 저장하는 명령어 레지스터와; 그리고

주소를 저장하는 프로그램 카운터를 포함하여 구성되며,

여기서 상기 주소로부터 명령어 그룹을 상기 명령어 레지스터로 인출되며,

상기 프로그램 카운터 내의 상기 주소는 메모리 주소 또는 레지스터 주소인 것을 특징으로 하는 컴퓨터.

청구항 2

제 1 항에 있어서,

상기 프로그램 카운터 내의 상기 주소는 선택적으로 다수의 레지스터를 지시하는 것을 특징으로 하는 컴퓨터.

청구항 3

제 1 항에 있어서,

상기 명령어 그룹은 하나보다 많은 명령어를 포함하는 것을 특징으로 하는 컴퓨터.

청구항 4

제 1 항에 있어서,

상기 명령어 레지스터에 실행될 명령어가 남지 않은 경우, 상기 컴퓨터는 상기 프로그램 카운터에 저장된 상기 주소에 따라 명령어 그룹을 인출하는 것을 특징으로 하는 컴퓨터.

청구항 5

제 1 항에 있어서,

상기 컴퓨터가 메모리부터 명령어 그룹을 인출하면, 상기 프로그램 카운터가 증분되고; 그리고

상기 컴퓨터가 레지스터로부터 명령어 그룹을 인출하면, 상기 프로그램 카운터가 증분되지 않는 것을 특징으로 하는 컴퓨터.

청구항 6

제 1 항에 있어서,

상기 명령어 그룹 내의 명령어가 JUMP 명령어이면, 상기 프로그램 카운터가 상기 JUMP 명령어에 의해 표시된 주소로 로드되는 것을 특징으로 하는 컴퓨터.

청구항 7

제 6 항에 있어서,

상기 JUMP 명령어에 의해 표시된 상기 주소가 상기 명령어 집합 내의 상기 JUMP 명령어 뒤를 따르는 것을 특징으로 하는 컴퓨터.

청구항 8

컴퓨터에서 명령어를 실행하는 방법에 있어서,

(a) 임의의 주소로부터 명령어 그룹을 인출하는 단계와, 여기서 상기 주소는 메모리 위치의 주소 또는 포트 주소이며;

(b) 실행될 상기 명령어 그룹을 명령어 레지스터에 배치하는 단계와; 그리고

(c) 상기 명령어 레지스터로부터 상기 명령어 그룹 내의 적어도 하나의 명령어를 실행하는 단계를 포함하는 것을 특징으로 하는 컴퓨터에서 명령어를 실행하는 방법.

청구항 9

제 8 항에 있어서,

상기 주소는 프로그램 카운터 레지스터로부터 인출되는 것을 특징으로 하는 컴퓨터에서 명령어를 실행하는 방법.

청구항 10

제 8 항에 있어서,

상기 명령어 그룹 내의 모든 명령어들이 실행되면, 추가의 명령어 그룹이 인출되는 것을 특징으로 하는 컴퓨터에서 명령어를 실행하는 방법.

청구항 11

제 10 항에 있어서,

상기 주소가 상기 메모리 위치의 주소이면, 상기 추가의 명령어 그룹이 인출되기 전에 상기 주소가 증분되고;

상기 주소가 상기 포트 주소이면, 상기 추가의 명령어 그룹이 인출되기 전에 상기 주소가 증분되지 않는 것을 특징으로 하는 컴퓨터에서 명령어를 실행하는 방법.

청구항 12

제 8 항에 있어서,

상기 주소가 하나보다 많은 포트를 표시할 수 있는 것을 특징으로 하는 컴퓨터에서 명령어를 실행하는 방법.

청구항 13

제 8 항에 있어서,

상기 (c) 단계에서, 상기 명령어 그룹 내의 모든 명령어들을 실행하거나, 전환 명령어가 나올 때까지 상기 명령어 그룹 내의 명령어들을 실행하며,

여기서 상기 전환 명령어는 다른 위치로부터 명령어 그룹을 인출하도록 하는 명령어인 것을 특징으로 하는 컴퓨터에서 명령어를 실행하는 방법.

청구항 14

제 13 항에 있어서,

상기 전환 명령어는 JUMP 명령어인 것을 특징으로 하는 컴퓨터에서 명령어를 실행하는 방법.

청구항 15

컴퓨터에서 데이터 블록을 처리하는 방법에 있어서,

(a) 임의의 주소로부터 데이터 블록을 인출하는 단계와, 여기서 상기 주소는 메모리 위치의 주소 또는 포트 주소이며;

(b) 상기 데이터 블록이 명령어 그룹을 포함하면, 상기 데이터 블록을 명령어 레지스터로 제공하는 단계와; 그리고

(c) 상기 데이터 블록이 명령어들이 아닌 데이터를 포함하면, 상기 데이터 블록을 데이터 스택의 상부에 제공하는 단계를 포함하는 것을 특징으로 하는 컴퓨터에서 데이터 블록을 처리하는 방법.

청구항 16

제 15 항에 있어서,

상기 명령어 그룹은 선택적으로 데이터를 또한 포함할 수 있는 것을 특징으로 하는 컴퓨터에서 데이터 블록을 처리하는 방법.

청구항 17

제 15 항에 있어서,

상기 데이타 블록이 명령어 그룹을 포함하는지 데이터를 포함하는지 여부는 이전의 명령어 그룹 내의 최종 명령어가 실행되었는지 여부에 따라 판정되는 것을 특징으로 하는 컴퓨터에서 데이터 블록을 처리하는 방법.

청구항 18

제 17 항에 있어서,

상기 이전의 명령어 그룹 내의 상기 최종 명령어가 실행되었으면, 상기 데이터 블록은 명령어 그룹을 포함하는 것으로 판정되는 것을 특징으로 하는 컴퓨터에서 데이터 블록을 처리하는 방법.

청구항 19

제 15 항에 있어서,

상기 데이터 스택은 주요 데이터 스택인 것을 특징으로 하는 컴퓨터에서 데이터 블록을 처리하는 방법.

청구항 20

제 15 항에 있어서,

상기 데이터 블록이 명령어 그룹을 포함하면, 상기 명령어 레지스터로부터의 상기 명령어 그룹 내의 명령어들을 실행하는 단계를 더 포함하는 것을 특징으로 하는 컴퓨터에서 데이터 블록을 처리하는 방법.

청구항 21

일련의 명령어를 실행하는 컴퓨터에 있어서,

명령어 그룹을 일시적으로 저장하는 명령어 레지스터와;

상기 명령어 레지스터에 저장된 명령어를 실행하는 논리 유닛과; 그리고

임의의 포트로부터 상기 명령어 그룹을 상기 명령어 레지스터로 인출하는 수단을 포함하는 것을 특징으로 하는 컴퓨터.

명 세 서

기 술 분 야

<1> 본 출원은, 동일한 발명자에 의해 2006년 2월 16일자로 출원된 미국출원번호 11/355,513의 일부 계속 출원(continuation-in-part)이며, 동일한 발명자에 의해 2006년 3월31자로 출원된 미국 가 출원번호 60/788,265의 우선권을 주장하며, 동일한 발명자에 의해 2006년 5월 3일자로 출원된 미국 가 출원번호 60/797,345의 우선권을 주장한다. 상기 모든 출원들은 본원의 참조로서 인용된다.

<2> 본 발명은 컴퓨터와 컴퓨터 프로세서에 관한 것이며, 특히 컴퓨터가 명령어들을 외부 소스에서 받았을시 상기 명령어들을 저장하지 않고 상기 명령어들을 실행할 수 있게 하는 방법 및 수단과, 상기 방법 및 수단을 이용하여 컴퓨터 간의 통신을 용이하게 하는 관련 방법에 관한 것이며, 다른 컴퓨터의 가용 자원을 이용하는 컴퓨터의 능력을 촉진시키는 관련 방법에 관한 것이다. 본 발명의 직접 실행(direct execution) 방법과 장치에 대한 가장 보편적인 사용예는, 단일의 마이크로 칩 상에 여러 컴퓨터들을 조합하는 것인바, 여기에서는 동작 효율성이 중요한데, 그 이유는 동작 속도 증가에 대한 요구뿐만 아니라 보다 큰 효율성의 결과인 전력 절감 및 열 감소 때문이다.

배 경 기 술

<3> 컴퓨터 기술에서 프로세싱 스피드는 많이 요구되는 특성이며, 좀더 빠른 컴퓨터 및 프로세서를 달성하고자 하는 추구는 계속 되고 있다. 그러나, 산업계에서는, 적어도 현재 알려진 기술로는 마이크로 프로세서의 속도를 증가시키는 데에 있어, 한계가 다가오고 있음을 인식하고 있다. 따라서, 복수의 프로세서를 사용하여 컴퓨터 태스크(task)를 이들 프로세서들 사이에서 분배하여 속도를 높이고자 하는 관심이 증가하고 있다.

<4> 복수의 프로세서 사용은 프로세서 간의 통신을 요구하는 경우가 많다. 따라서 프로세서 간에 명령어와 데이터 전송에 꽤 많은 시간들이 허비되어 통신량이 매우 많다고 해도 무방하다. 이와 같이 통신량이 많을 때, 이를 달성하기 위해 실행되어야만 하는 각각의 부가적인 명령어들은 프로세스에서 증가된 지연을 야기하는 바, 이는 누적되면 무시 못할 양이 된다. 하나의 컴퓨터에서 다른 컴퓨터로 명령어들 또는 데이터를 통신하기 위한 종래의 방법은, 우선 데이터 또는 명령어를 수신 컴퓨터에 저장한 다음, 실행을 위해 호출하거나(명령어일 경우) 또는 그에 대한 동작을 위해 호출한다(데이터일 경우).

<5> 컴퓨터들 사이에서, 데이터 또는 명령어의 형식(form)으로 정보를 전송하고, 정보를 수신하고, 정보를 이용하는데 필요한 단계(step)들을 감소시키는 것은 매우 용이할 것이다. 하지만, 발명자가 아는바에 따르면, 앞서 설명된 프로세스를 획기적으로 개선한 종래 기술은 존재하지 않는다.

<6> 또한, 종래 기술에 따르면 종종 컴퓨터의 "주의를 요하는 것(get the attention)" 이 필요하다고 알려져 있다. 즉, 컴퓨터가 하나의 태스크로 인해 바쁘다고 하더라도, 좀더 중요하고 급박한 태스크가 발생하여 컴퓨터가 잠시 이전의 태스크를 중단해야(divert) 하는 경우가 종종 일어날 수 있다. 이러한 일례로서, 사용자 입력 장치를 사용하여 컴퓨터에 입력을 제공하는 경우가 있지만, 본 발명은 이에 국한되지 않는다. 이와 같은 경우, 컴퓨터는 입력을 잠시 인지하거나 및/또는 입력에 따른 반응을 할 필요가 있을 것이다. 그 후 컴퓨터는 입력이 들어오기 이전 동작을 개시하기도 하고, 입력에 따라 이전 동작이 변경될 수도 있다. 본 예에서는 외부 입력만이 사용되었지만, 컴퓨터 내에서 ALU의 처리에 대한 잠재적인 충돌이 있는 경우에도 같은 상황이 발생할 수 있다.

<7> I/O 포트에서 데이터 및 상태 변경을 수신할 때 종래 기술에는 두 가지 방법이 있다. 하나는 포트를 폴링(polling) 하는 것인데 이 방법은, 임의의 데이터가 수신되었거나 또는 상태 변경이 발생했는지를 결정하도록, 일정한 간격으로 포트의 상태를 읽는 것을 포함한다. 하지만, 이와 같이 포트를 폴링하는 것은 많은 시간과 자원을 필요로 하며, 이들 시간과 자원은 다른 작업을 하는데 더 유용하게 쓰일 수도 있다. 더 나은 대안으로 제시되는 것이 "인터럽트" 이다. 인터럽트를 사용할 때, 프로세서는 할당된 태스크를 계속 수행하다가 이후, 바이트가 수신되었거나 또는 상태가 바뀜으로 인해 I/O 포트/장치(device)가 프로세서의 주의를 요구할 때에, 인터럽트 요청(Interrupt Request : IRQ)을 프로세서에 보낸다. 일단 프로세서가 인터럽트 요청을 수신하면, 프로세서는 현재의 명령어를 끝내고 스택에 정보를 집어넣고 적절한 인터럽트 서비스 루틴(Interrupt Servie Routine : ISR)을 실행하는바, 인터럽트 서비스 루틴은 포트로부터 바이트를 제거할 수 있으며 이를 버퍼에 위치시킬 수도 있다. ISR이 끝나면, 프로세서는 인터럽트 이전에 진행하던 동작으로 돌아간다. 이와 같은 방법을 통해, 프로세서는 I/O 장치가 주의를 필요로 하는지를 확인하는데에 시간을 낭비하지 않고, 디바이스는 주의를 요할 때에만 인터럽트를 서비스한다. 하지만, 인터럽트의 이용은 많은 경우 바람직하지 않은데, 이는 인터럽트의 이용과 관련하여 많은 양의 오버헤드(overhead)가 있을 수 있기 때문이다. 예를 들면, 인터럽트가 매번 발생할 때마다, 컴퓨터는 이전에 실행하고자 했던 태스크에 관한 정보를 임시적으로 저장해야 하고, 인터럽트에 관한 정보를 로드해야 하며, 이후 인터럽트가 처리되면 이전 태스크에 대해서 필요한 데이터를 다시 로드해야 한다. 자명한 것은, 이러한 모든 시간과 자원을 줄이거나 없애는 것이 바람직하다는 사실이다. 하지만, 인터럽트들로 인한 요구들을 경감시킬 수 있는 종래기술은 아직 개발되지 않았다.

발명의 상세한 설명

<8> 따라서, 2개 이상의 컴퓨터들 사이에서 데이터 및/또는 명령어를 통신하는 데에 있어서, 동작 속도를 빠르게 할 수 있는 장치 및 방법을 제공하는 것이 본 발명의 목적이다.

<9> 본 발명의 다른 목적은, 비싸지 않은 가격으로 방대한 계산 능력을 제공할 수 있는 장치 및 방법을 제공하는 것이다.

<10> 본 발명의 또 다른 목적은, 최소의 시간 안에 계산 집약적인 태스크를 완수할 수 있는 장치 및 방법을 제공하는 것이다.

<11> 본 발명의 또 다른 목적은, 광대한 실행 능력을 갖춘 컴퓨터 디바이스를 제공하는 것이다.

<12> 컴퓨터들 및 컴퓨터에 의해 조종되는 디바이스들 간의 통신의 효율성을 증대시키는 것도 본 발명의 또 다른 목

적이다.

<13> 컴퓨터들 간의 통신 효율성을 증대시키는 것도 본 발명의 목적이다.

<14> 컴퓨터 상호간에 통신하는 방식의 효율성을 증가시키거나 컴퓨터와 가령 사용자 입력 디바이스 및 이와 유사한 장치들 사이에서 통신하는 방식의 효율성을 높이는 것도 본 발명의 목적이다.

<15> 간단히 말하면, 본 발명의 개시된 실시예는 그 자신의 메모리를 구비하여 독립적인 계산 기능을 갖춘 컴퓨터이다. 본 발명의 일 실시예에 따르면 복수의 컴퓨터가 어레이(array)로 구성된다. 협동적으로 태스크를 실행하기 위해서, 컴퓨터들은 데이터와 명령어를 상호간에 전송해야 한다. 동시에 동작하는 모든 컴퓨터들은 흔히 대부분의 태스크에 요구되는 계산 능력보다 더 많은 능력을 제공할 것이기 때문이며, 그리고 복수개의 컴퓨터들 사이에 태스크를 분배하기 위해 그 어떤 알고리즘 또는 방법을 사용하더라도 공평하지 않은 분배가 야기되는 것은 거의 확정적이기 때문에, 최소한 몇몇 아니 어쩌면 대부분의 컴퓨터는 소정 시간에 상기 태스크를 완수하는데 적극적으로 참여하고 있지 않을 수도 있다. 그러므로, 자주 사용되지 않는 컴퓨터가 좀더 바쁜 컴퓨터에 계산 자원과 메모리, 또는 둘 다를 "차용"해주는 방법을 찾는 것이 매우 바람직하다. 이런 관계가 효율적이고 유용하기 위해서는, 이웃한 컴퓨터들 간의 통신과 상호작용이 최대한 신속하고 효과적이어야 한다. 그러므로, 본 발명은, 실행 이전에 데이터 및/또는 명령어를 수신하고 저장해야 하는 대신에, 다른 컴퓨터로부터 제공되는 명령어들을 컴퓨터가 직접적으로 실행하거나 및/또는 데이터에 직접적으로 동작할 수 있는 방법 및 수단을 제공한다. 여기서 유념할 점은, 본 발명은, 한 컴퓨터로 하여금 다른 컴퓨터로부터의 명령어들 또는 데이터를 또 다른 컴퓨터로 "전달(pass on)"하게 하는 중개(intermediary)로서 작용하는 명령어들에 대해 매우 유용하다는 것이다.

<16> 전술된 실시예에 의하면, 어떤 컴퓨터가 하나 또는 그 이상의 이웃과 통신하기를 시도할 때에, 불필요한 전력 낭비와 방출 열을 막기 위하여, 상기 이웃 또는 이웃들중 하나가 통신을 완료하기 위해 반응할 때 까지 그 컴퓨터는 사실상 전력을 거의 소비하지 않는 수면 모드로 들어간다. 그러나, 이는 본 발명의 필수적인 양상이 아니다. 더 나아가, 전력과 열방출을 감소시키기 위해서는, 통신이 완료되기를 기다리는 동안에, 통신을 시작한 컴퓨터는 전력 소비를 중지하거나 상당량 줄이는 것이 바람직하다. 이는 매우 방대한 수의 방법으로 실행 가능하다고 생각되어 질 수 있다. 예를 들어, 컴퓨터가 내장 시계나 외장 시계에 의해 동기화 될 수 있다면, 이러한 기간 동안에는 상기 시계가 늦춰지거나 멈추어 질 수 있다. 사실, 이러한 실시예가 본 발명의 범주를 벗어나서도 구현될 수 있으나, 본 출원에 상술된 실시예들은 본 발명자에 지식에 의하면, 최고의, 최상의 효율성을 갖는 실시예들이라 생각된다.

<17> 여기에 기술된 본 발명의 일 양상은, 명령어나 데이터의 소스가 컴퓨터의 내부 메모리이건 간에, 또는 이러한 명령어나 데이터가 다른 컴퓨터나 외부 통신 포트 등등과 같은 다른 소스로부터 수신되었던지 간에 관계 없이, 상기 데이터나 명령어들은 본질적으로 동일하게 취급된다는 점이다. 이와 같은 특징은 획기적인바, 그 이유는 명령어 또는 데이터를 저장하기 그리고 이후에 내부 메모리부터 이들을 다시 호출하기와 같은 "부가적인" 동작들이 불필요해지기 때문에, 이에 의해 필요한 명령어들의 수를 줄일 수 있고 관련된 컴퓨터들의 동작 스피드를 증대시킬 수 있기 때문이다.

<18> 본 명세서에 기술된 실시예의 또 다른 양상은, 일반적으로 동시에 매우 작은 그룹들의 명령어들이 다른 컴퓨터로 통신될 수 있다는 점인바, 반복적으로 되풀이됨(repetitive iterations)을 요구하는 상대적으로 간단한 동작들은 간단히 실행될 수 있다는 점이다. 이는 컴퓨터간의 통신 과정을 신속하게 만들 수 있다.

<19> 본 발명의 또 다른 양상은, 다양한 태스크를 실행할 수 있는 다수의 컴퓨터가 갖추어져 있으며, 입력을 기다리는 동안 사실상 전력 소비를 하지 않는 수면 상태로 한 개 이상의 컴퓨터가 대기 될 수 있기 때문에, 그러한 컴퓨터들이 입력을 기다리고 있는 태스크에 배정될 수 있고, 이에 따라 다른 태스크들을 실행하고 있을 수도 있는 다른 컴퓨터들을 인터럽트(interrupt)할 필요 없게 된다.

<20> 본 발명의 상기 및 기타의 목적들 및 장점들은 도면에서 도시되어 본원에서 설명되는 본 발명을 실행하기 위한 방법들에 대한 설명 및 그의 산업상 이용가능성에 비추어 당업자에게 명확해질 것이다. 리스트되는 목적들 및 장점들은 본 발명의 가능한 모든 장점들을 속속들이 다 규명한 것은 아니다. 또한, 응용에 있어서 1개 이상의 의도된 목적들 및/또는 장점들이 존재하지 않거나 요구되지 않는 경우에도 본 발명을 실행하는 것이 가능하다.

<21> 또한, 당업자들이라면 본 발명의 다양한 실시예들이 상기 설명한 목적들 그리고/또는 장점드을 1개 이상(반드시 전부일 필요는 없다) 달성할 수 있음을 인식할 것이다. 따라서 본원에서 설명되는 목적들 및/또는 장점들은 본 발명의 본질적인 요소들이 아니며, 제한적인 것으로 해석되어서는 안된다.

실 시 예

<31> 본 발명은 도면을 참조하여 설명되는바, 동일한 참조번호는 동일하거나 유사한 요소들을 나타낸다. 본 발명은 발명의 목적을 달성하기 위한 방법들의 측면에서 설명되지만, 해당 기술의 당업자는 본 발명의 사상이나 범주를 벗어남이 없이 전술한 가르침에 따라 다양한 실시예들이 가능함을 이해할 것이다.

<32> 본 명세서에 기재된 바와 같은 본 발명의 실시예들, 변형예들 및 도면에 도시된 것들은, 단지 예시적인 취지로 제공된 것이며 본 발명의 사상을 제한하는 것이 아니다. 특별히 달리 언급되지 않는 한, 본 발명의 개별적인 양상들 및 구성요소들은 생략되거나 수정될 수도 있으며, 또는 알려진 등가물로 대체되거나 또는 미래에 개발될 수도 있거나 미래에 대용물로 인정받을 수 있는 것과 같은 아직 알려지지 않는 대용물에 의해 대신될 수 있다. 본 출원의 잠재적인 응용 가능성은 광대하며 본 발명이 그와 같은 많은 응용들에 적용될수 있도록 의도되었기에, 발명의 취지와 범주에서 벗어나지 않는 한도에서 본 발명은 변경될 수도 있다.

<33> 본 발명의 실시예는 각각의 컴퓨터들의 어레이(computer array)다. 이 어레이는 도 1에서 도시되어 있으며 범용 참조번호 10에 의해 지시된다. 컴퓨터 어레이(10)는 여러 개의(도시된 도면에서는 24개) 컴퓨터(12)들을 갖는다(컴퓨터들은 때때로 "코어들", "노드들" 이라고 지칭되기도 한다). 도시된 실시예에서, 모든 컴퓨터들(12)은 하나의 다이(14)에 배치되어 있다. 각각의 컴퓨터들(12)은 일반적으로 독립적으로 기능하는 컴퓨터들인바, 이는 좀더 상세히 후술될 것이다. 컴퓨터들(12)은 복수의 상호연결 데이터 버스들(16)에 의해 상호 연결되어 있다(데이터 버스들의 개수는 앞으로 자세히 후술될 것이다). 이러한 실시예에서, 상기 데이터 버스들(16)은 양방향 비동기 고속 병렬 데이터 버스들이지만, 본 발명의 사상의 범위 내에서는 이러한 목적으로 또 다른 상호연결 수단들이 채용될 수도 있다. 어레이(10)에 대한 상기 실시예에서, 컴퓨터들(12) 사이의 데이터 통신은 비동기식일 뿐만 아니라, 개별적인 컴퓨터들(12) 역시 내부적으로는 비동기식 모드에서 동작한다. 이러한 사실은 발명자에 의해 중요한 장점으로 인지된 것이다. 예를 들어, 클럭 신호는 컴퓨터 어레이(10)의 전체에 분포될 필요가 없기 때문에, 전력이 크게 절약될 수 있다. 나아가, 클럭 시그널을 공급할 필요가 없다는 것은, 컴퓨터 어레이(10)의 크기를 제한하거나 또는 다른 알려진 문제점들을 일으킬 수 있는 많은 타이밍 문제들을 제거한다. 또한, 개별적인 프로세서가 비동기적으로 동작한다는 점에서 많은 전력 소비를 줄일수 있고 이는 각각의 프로세서가 내장된 클럭이 없기에 명령어 비실행시 사실상 어떤 전력도 소비하지 않기 때문이다.

<34> 당업자는 도 1의 다이(die)(14) 상에는 명확성을 위해서 생략된 부가적인 구성요소들이 있음을 인식할 것이다. 그러한 부가적인 구성요소들은 파워 버스들, 외부 연결패드들, 및 마이크로프로세서 칩의 기타 다른 공통요소들을 포함하나 이에 국한된 것이 아니다.

<35> 컴퓨터(12e)는 상기 어레이(10)의 주변부에 있지 않은 컴퓨터들(12) 중 하나의 예이다. 즉, 컴퓨터(12e)는 직교적으로 인접한 네 개의 컴퓨터들(12a, 12b, 12c 및 12d)을 갖는다. 하지만, 4개 이상의 컴퓨터를 이용하는 것도 본 발명의 범주에 들어간다. 컴퓨터 12a부터 12e까지의 이 그룹핑은 어레이(10)의 컴퓨터들(12) 간의 통신들에 대한 이후의 보다 상세한 논의와 관련되어 사용될 것이다. 도 1에서 도시된 바와 같이, 컴퓨터(12e)와 같은 내부 컴퓨터들은, 버스들(16)을 통해 직접적으로 서로 통신할 수 있는 네 개의 다른 컴퓨터들(12)을 가질 것이다. 이하의 설명에서, 어레이(10) 주변부의 컴퓨터들(12)은 오직 세 개, 모퉁이의 컴퓨터들(12)의 경우에는 오직 두 개의 다른 컴퓨터들(12)과 직접적으로 통신하게 될 것이라는 것을 제외하고는, 논의되는 원칙은 상기 모든 컴퓨터들(12)에 적용될 것이다.

<36> 도 2는 오직 몇몇 컴퓨터들(12)만을 보여주고 있는 도 1의 부분 및 특히 12a에서부터 12e까지의 컴퓨터들을 포함한 부분에 대한 보다 구체적인 도면이다. 도 2는 역시 데이터 버스들(16)이 각각 읽기 라인(read line)(18), 쓰기 라인(write line) 및 다수의(이 예에서는 18) 데이터 라인(data lines)(22)을 가진다는 것을 보여주고 있다. 상기 데이터 라인(22)은 하나의 18 비트 명령 워드의 모든 비트들을 일반적으로 동시에 병렬적으로 전송할 수 있다. 본 발명의 일 실시예에서, 컴퓨터들(12)의 일부는 인접 컴퓨터들의 미러 이미지들(mirror images)이다. 그러나, 컴퓨터들(12)이 모두 동일한 방향으로 되었는지(oriented identically) 또는 인접 컴퓨터들의 미러 이미지들로서 구성되는지의 여부는 지금 설명되는 본 발명에서 제한된 양상이 아니다. 따라서, 본 발명을 정확히 설명하기 위하여, 잠재적인 문제들은 이 출원에서 다뤄지지 않을 것이다.

<37> 본 발명의 한 예에 따라, 컴퓨터(12e)와 같은 컴퓨터(12)는 한 개, 두 개, 세 개 또는 네개 모두의 읽기 라인들(18)을, 각각 한 개, 두 개, 세 개 또는 모든 네 개의 인접 컴퓨터들(12)로부터 데이터를 수신할 준비가 되어 있다고 하이로 설정할 수 있다. 유사하게, 컴퓨터(12)가 한 개, 두 개, 세 개 또는 모든 네 개의 쓰기 라인들(20)을 하이(high)로 설정하는 것도 역시 가능하다. 본 발명자는 동시에 2개 이상의 쓰기 라인(20)을 설정하는

것이 현재에는 실용적인 가치가 없다고 생각하지만, 이러한 설정 자체는 본 발명의 범주에 들어가며, 이러한 구성이 미래에 실행되는 것은 충분히 가능하다.

<38> 인접 컴퓨터들(12a,12b,12c 또는 12d) 중 하나가 자신과 컴퓨터(12e) 사이의 쓰기 라인(20)을 하이로 설정할 때, 만약 컴퓨터(12e)가 이미 그에 대응하는 읽기 라인(18)을 하이로 설정했다면, 그때 워드는 연결된 데이터 라인들(22) 상에서 컴퓨터(12a, 12b, 12c 또는 12d)로부터 컴퓨터(12e)로 전송된다. 그러면, 상기 전송 컴퓨터(12)는 쓰기 라인(20)을 해제하고 상기 수신 컴퓨터(이 예에서 12e)는 쓰기 라인(20) 및 읽기 라인(18) 둘 다를 로우(low)로 할 것이다. 상기 뒤의 동작은 상기 데이터가 수신되었다는 것을 전송 컴퓨터(12)에 알려주게 될 것이다. 상기 설명이 반드시 일련의 사건들을 순서대로 서술하도록 의도된 것은 아님을 밝혀둔다. 실제상황에서는, 이 예에서의 상기 수신 컴퓨터는 전송 컴퓨터(12)가 그것의 쓰기 라인(20)을 해제하기(하이로 올리는 것을 멈추기)전에 쓰기 라인(20)을 약간 로우로 설정하려고 할 수도 있다. 이러한 경우, 상기 전송 컴퓨터(12)가 그것의 쓰기 라인(20)을 해제하자마자 상기 쓰기 라인(20)이 수신 컴퓨터(12e)에 의해 로우로 될 것이다.

<39> 본 실시예에서는,오직 프로그래밍 에러일때만 버스상에서(16) 양 끝단에 있는 컴퓨터(12) 둘이 이 그 사이의 읽기 라인(18)을 하이로 설정하게끔 할 것이다. 마찬가지로, 동시에 쓰기 라인(18) 사이를 하이로 설정하려고 하는 것도 버스들(16)상에 양 끝단에 있는 컴퓨터들(12) 둘 다에 대한 에러일 것이다. 유사하게, 위에서 논의된 바와 같이, 단일 컴퓨터(12)가 자신의 네 개 쓰기 라인들(20) 중 하나 이상을 하이로 설정하는 것이 현재로서는 바람직하다고 여겨지지는 않는다. 그러나, 다수의 읽기 라인들(18)의 서로 다른 조합을 하이로 설정하는 것이 바람직한 경우가 있을 것이라고 현재 예상되는데, 이 경우는 컴퓨터(12)들 중 하나가, 자신의 상응하는 쓰기 라인(20)을 하이로 설정한 선택된 컴퓨터들 중 첫번째로부터 데이터를 기다리는 대기 상태에 있는 경우이다.

<40> 상기 예에서, 컴퓨터(12e)는 인접 컴퓨터(컴퓨터들 12a,12b,12c 또는 12d의 하나 이상으로부터 선택된)가 그것의 쓰기 라인(20)을 하이로 설정하기 전에, 그것의 읽기 라인들(18)의 하나 이상을 하이로 설정하는 것으로 설명되었다. 그러나, 이 프로세스는 물론 반대 순서로도 일어날 수 있다. 예를 들어, 만약 컴퓨터(12e)가 컴퓨터(12a)에 쓰기를 시도하고 있는 중이었다면, 그때 컴퓨터(12e)는 컴퓨터(12e) 및 컴퓨터(12a) 사이의 쓰기 라인(20)을 하이로 설정할 것이다. 만약, 컴퓨터(12e) 및 컴퓨터(12a) 사이에 읽기 라인(18)이 컴퓨터(12a)에 의해서 이미 하이로 설정되어있지 않았다면, 그 때 컴퓨터(12e)는 컴퓨터(12a)가 그 읽기 라인(20)을 하이로 설정할 때까지 단순히 기다리게 될 것이다. 그러면 상기 설명한 바와 같이, 쓰기 라인(18) 및 읽기 라인(20) 두 개의 대응하는 라인 모두가 하이일 때, 데이터 라인들(22)에서 전송되기를 기다리는 상기 데이터는 전송된다. 그 후에, 수신 컴퓨터(12)(이 예들에서 컴퓨터 12a)는, 전송 컴퓨터(12e)가 쓰기 라인(18)을 해제하자마자, 상기 두 컴퓨터들(이 예에서 12e 및 12a) 사이의 읽기 라인(18) 및 쓰기 라인(20) 둘 다를 로우로 설정한다.

<41> 컴퓨터(12e)와 같은 컴퓨터(12)가 쓰기를 고대하면서 그것의 쓰기 라인들(20) 중 하나를 하이로 설정할 때마다, 만약 상기 데이터가 즉시 전송되는 경우에 있어서, 만약 상기 데이터를 전송받게 되는 컴퓨터(12)가 이미 그것의 읽기 라인(18)을 하이로 설정하지 않았다면, 상기 설명한 바와 같이 상기 데이터가 상기 적절한 인접 컴퓨터(12)로부터 "요청"될 때까지, 본질적으로는 전력을 사용하지 않고, 단지 기다릴 것이다. 유사하게, 컴퓨터(12)가 읽기를 고대하면서 그것의 읽기 라인들(18) 중 하나 이상을 하이로 설정할 때마다, 선택된 컴퓨터(12)로 연결된 쓰기 라인(20)이 상기 두 컴퓨터들(12) 간에 명령을 전송하도록 하이로 올라갈 때까지 본질적인 전력의 사용 없이, 단지 기다릴 것이다.

<42> 컴퓨터들(12)이 상기 설명한 바대로 기능하도록 하는 여러 가능한 수단들 및/또는 방법들이 있을 수 있다. 그러나, 본 발명의 예에서는, 컴퓨터들(12)은 일반적으로 내부적으로 비동기적으로 실행하기 때문에 단순히 그렇게 동작하는 것이다(상기 설명된 비동기식 방법으로 그들 사이에 데이터를 전송하는 것 외에도). 즉, 명령들은 순차적으로 완료된다. 쓰기 또는 읽기 명령이 있을 때, 그 명령이 완료될 때까지는(또는, 아마도 대안적으로는, "리셋(reset)" 등등에 의해 그것이 중단(aborted)될 때까지) 더 이상의 동작이 있을 수 없다. 종래 기술에는 정기적인 클럭 펄스가 없다. 오히려 펄스는, 실행되고 있는 명령이 읽기 또는 쓰기 유형의 명령(주어진 그 읽기 또는 쓰기 유형의 명령은 다른 엔티티에 의해 완료될 것을 요구한다)이 아닐 때만 또는 그 밖에 상기 읽기 또는 쓰기 유형의 동작이 사실상 완료되었을 때에만 다음 명령을 수행하기 위해서 발생 된다.

<43> 도 3은 도 1 및 도 2의 프로세서들(12)의 하나의 예에 대한 일반적인 레이아웃(layout)을 묘사하는 블록 다이아그램이다. 도 3에서 보여지는 바와 같이, 상기 프로세서들(12)의 각각은 일반적으로 자체적인 램(24) 및 롬(26)을 가지는 자체 완비된(self contained) 프로세서이다. 앞서 언급한 바와 같이, 컴퓨터들(12)은 때때로 현 실시예에서 단일 칩 상에서 결합 되는 바, 개별 "노드(nodes)" 로 언급되기도 한다.

<44> 컴퓨터(12)의 다른 기본 구성 요소들로는 리턴 스택(return stack)(28), 명령 영역(instruction area)(30), 산술 논리 연산 장치(arithmetic logic unit:ALU)(32), 데이터 스택(data stack)(34) 및 명령들을 디코딩하기 위한 디코드 논리 영역(decode logic section)(36)이 있다. 당업자는 일반적으로 이 예에서의 컴퓨터들(12)처럼 스택 기반 컴퓨터들의 동작에 익숙할 것이다. 컴퓨터들(12)은 데이터 스택(34) 및 별도의 리턴 스택(28)을 가지는 듀얼(dual) 스택 컴퓨터들이다.

<45> 본 발명의 실시예에서, 컴퓨터(12)는 인접 컴퓨터들(12)과 통신하기 위한 네 개의 통신 포트들(38)을 가진다. 이 통신 포트들(38)은 오프(off) 상태, 수신(receive) 상태(신호들을 컴퓨터(12)로 받아들이기 위한) 및 발신 상태(신호들을 컴퓨터 밖으로 내보내기 위한)를 가지는 3상 상태 드라이버(tri-state drivers)이다. 물론, 만약 특정 컴퓨터(12)가 컴퓨터(12e)의 예와 같이 어레이(도 1)의 내부에 있는 것이 아니라면, 적어도 여기서 기술된 목적을 위해서는, 상기 통신 포트들의 하나 이상은 그 특정 컴퓨터에 사용되지 않을 것이다. 상기 다이(die)의 끝 부분에 있는 그런 통신 포트들(38)은 그러한 통신 포트(38)가 외부 입출력 포트(39)(도 1)로서 역할 수 있도록 하기 위해, 그러한 컴퓨터(12) 내부로 설계되거나 또는 외부에 있으면서 그곳과 연결되도록 설계된 부가적인 회로부를 가질 수 있다. 그러한 외부 입출력 포트들(39)의 예들은 USB 포트, RS232 직렬 버스 포트들, 병렬 통신 포트들, AD/DA 변환 포트들, 그리고 많은 기타 다른 가능한 변형들을 포함하며, 다만 이것들로 한정되는 것은 아니다. 도 1에서, "가장자리(edge)" 컴퓨터(12f)는 외부 입출력 포트(39)을 통해 외부 장치(82)와 통신하기 위해서 인터페이스 회로부(80)에 연결되어 있는 것으로 도시되어 있다.

<46> 명령 영역(30)은 본 실시예에서, 레지스터(40a), B 레지스터(40b) 및 P 레지스터(40c)를 포함하는 다수의 레지스터들(40)을 포함한다. 이 예에서, B 레지스터(40b) 및 P 레지스터(40c)가 9비트 레지스터인 반면, A 레지스터(40a)는 풀(full) 18비트 레지스터이다.

<47> 본 발명은 본예에 한정되지 않으나 현재의 프로세서는 네이티브 Forth 언어 명령어들을 실행하도록 구성되어 있다. 포스 컴퓨터 언어에 친숙한 사람이라면, FORTH "워드들"로서 알려져있는 복잡한 포스 명령들이 프로세서 내에 설계된 네이티브 프로세서 명령들로부터 비롯된다는 것을 인식할 것이다. 포스 워드들의 집합은 "딕셔너리(dictionary)"로서 알려져있다. 다른 언어들에 있어서, 이것은 "라이브러리(library)"로서 알려져 있다. 하기에서 보다 상세히 설명되는 바와 같이, 프로세서(12)는 RAM(26), ROM(26)으로부터, 또는 데이터 버스들(16)(도 2)중 하나로부터 직접적으로, 한번에 18개의 비트들을 읽는다. 하지만, 포스에서, (오퍼랜드가 없는 명령들(operand-less instructions)로서 알려져있는) 대부분의 명령들은 자신들의 오퍼랜드들을 스택들(28 및 34)로부터 직접 얻기 때문에, 이들은 일반적으로 단지 5비트의 길이를 가지며, 이에 따라 그룹 내의 마지막 명령이 단지 3비트 만을 요구하는 제한된 명령들의 세트로부터 선택된다는 조건하에서, 단일의 18 비트 명령 워드 내에 최대 4개의 명령들이 포함될 수 있다. (상술된 실시예를 따르면, 명령 워드내에서 가장 마지막 명령어의 후반 두자리 비트는 "01"이라 가정 된다.) 도 3에서 도시된 블록 다이어그램은 슬롯 시퀀서(12)이다.

<48> 이 발명의 실시 예에서는 데이터 스택은 ALU로 변경되는 값들을 위한 후입 선출법(Last-in-first-out) 스택이며, 리턴 스택은 CALL이나 RETURN 명령에 사용되는 내포된 리턴 주소(nested return address)를 위한 후입 선출법 스택이다. 리턴 스택(28)은 PUSH,POP, NEXT의 명령어에 의해 이용되며, 이는 차후 상세히 다뤄질 것이다. 데이터 스택(34)과 리턴 스택 (28)은, 종래 기술에 따른 많은 컴퓨터와 같은 스택 포인터에 의해 액세스되는 메모리가 아니다. 오히려, 34 와 28 스택은 레지스터들이다. 데이터 스택(34)의 가장 꼭대기의 레지스터는 T 레지스터(44)와 S 레지스터(46)이다. 데이터 스택의 나머지는 부가적인 8개 레지스터를 구비한 순환적 레지스터 어레이(circular register array)이며 그 레지스터는 S2로부터 S9으로 번호되어 있다. 순환적 레지스터 어레이 (34a)중 하나는 아무때나 S 레지스터(46) 밑에 있는 레지스터로서 언제든지 선택될 수 있다. S밑에 있는 스택 레지스터를 선택하는 시프트 레지스터의 값은 소프트 웨어에 의해 읽히거나 쓰일 수 없다. 유사하게, 리턴 스택의 가장 꼭대기 위치는 R 레지스터라 정해져 있으며, 리턴 스택의 나머지는 12개의 부가적인 하드웨어 레지스터로 구성된(도시 되지 않음) 순환적 레지스터 어레이(28a)이며, 이 레지스터는 R1 부터 R11으로 번호화 되어 있다.

<49> 이 발명의 실시예에서는 하드웨어 보조(hardware-assisted) 오버플로우와 언더플로우 감지 시스템이 없다. 일반적으로 종래 기술의 프로세서들은, 스택에 할당된 메모리의 범주를 벗어날 경우 에러 조건이 플래그 되도록, 스택포인터와 메모리 관리 등등을 이용하였다. 스택이 메모리에 있을 경우 오버플로우 되거나 언더 플로우될 경우 스택이라 정해지지 않은 것을 겹쳐 쓰거나 스택 아이템으로 쓸 경우가 발생하기 때문이다. 본 발명은 28과 34의 스택 바닥에 순환적 어레이(28a 와34a)를 갖추고 있기에 스택 영역을 벗어나 언더플로우 오버플로우 할 수 없다. 대신, 순환적 어레이 28a와 34a는 순환적 레지스터 어레이를 돌림(wrap around)한 것이다. 스택 28과 34는 한정된 크기를 갖고 있기에 스택 28과 34의 꼭대기에 푸쉬를 한다는 것은, 스택 맨 밑에 있는 값이 겹쳐 쓰

여진다(overwritten)는 것을 의미한다. 데이터 스택(34)에 10개 이상의 아이템을 넣는다던지 리턴 스택에 13개 이상의 데이터를 넣는다면, 스택 28과 34의 맨 밑에 있는 값은 겹쳐쓰기(overwrite)가 될 것이라는 것을 인지해야 한다. 28과 34의 아이템 수를 관리하고 한계 이상의 아이템을 넣지 못하게 방지하는 것은 소프트웨어의 책임이다. 하드웨어 자체는 스택 맨 밑에서 겹쳐쓰기가 행해지는 것을 감지하거나 에러라고 표시하지 않는다. 여기서 유의할 점은 소프트웨어는 스택 28과 34의 맨 밑의 순환 어레이(28a 와 34a)의 장점을 여러 방법을 통해 이용할 수 있다는 것이다. 한 예로, 스택 28과 34가 항상 비어있다고 가정할 수 있다. 과거의 아이템들은 스택 밑으로 푸시되어 스택으로부터 소거할 필요가 없다. 스택을 비우기 위해 초기화 할 필요가 없다.

<50> 전술된 레지스터와 더불어, 명령어 구역(30)은 현재 사용되고 있는 명령워드 (48)을 저장하는 18비트 레지스터(30a)와 현재 실행되고 있는 명령어를 위한 5 비트 옵코드(opcode) 레지스터(30b)를 갖추고 있다.

<51> 도 4는 명령 워드(48)를 도시적으로 표현한 것이다.(이 명령 워드(48)는 실제로 명령들, 데이터 또는 그들의 몇몇 조합을 포함할 수 있음을 밝혀둔다.) 명령워드(48)는 18개의 비트들(50)로 구성되어 있다. 이진 컴퓨터에서, 상기 비트들 각각(50)은 '1' 또는 '0'일 것이다. 여기서 앞서 논의된 바와 같이, 상기 18 비트 크기의 명령 워드(48)는 슬롯 0(54a), 슬롯 1(54b), 슬롯 2(54c) 그리고 슬롯 3(54d)으로 불리는 4개 슬롯(54)에 4개까지의 명령들(52)을 포함할 수 있다. 본 발명의 실시예에 있어서, 상기 18 비트 명령 워드들(48)은 항상 전체로서 읽힌다. 따라서, 상기 명령 워드(48)에는 항상 잠재적으로 4개까지의 명령들을 가질 수 있기 때문에, 모든 이용가능한 슬롯들(54)을 사용하는 것이 불필요하거나 심지어 바람직하지 않은 경우들에 대비하기 위해 무연산 명령어(no-op or no operation)들이 컴퓨터(12)의 명령 셋에 포함된다. 본 발명의 특정 실시예에 따라, 대체 슬롯들(특히, 슬롯 1(54b) 및 3(54c)) 내의 비트들(50)의 극성(polarity)(active high as compared to active low)은 역으로 바뀔 수 있다. 그러나, 이것은 본 발명의 필수적인 면은 아니며, 따라서, 본 발명의 보다 나은 설명을 위해서 잠재적 문제와 그 복잡성은 아래의 설명에서 피할 것이다.

<52> 도 5는 도 3의 슬롯 시퀀서(slot sequencer)(42)의 개략적인 표현이다. 도 5에서 보여지는 바와 같이, 슬롯 시퀀서(42)는 다수의 인버터들(56) 및 하나의 NAND 게이트(58)을 가짐으로서, 신호가 14개의 인버터(56) 및 NAND 게이트(58)를 통과함으로써 홀수번 인버터 될 수 있게 한다. OR 게이트(60)로의 두 개의 입력 중 어느 하나가 높이 올라갈 때 슬롯 시퀀서(42)에서 신호가 시작된다. 제 1 OR 게이트 입력(62)은 실행되고 있는 명령(52)의 비트 i4(66)(도 4)로부터 들어온다. 만약 비트 i4가 하이이면 그때 그 특정 명령(52)은 ALU 명령이고, 상기 i4 비트(66)은 '1'이다. i4비트가 '1'일 때, 그 때 제 OR 게이트 입력(62)은 하이며, 그리고 슬롯 시퀀서(42)는 다음 명령(52)을 실행하기 위해 펄스를 발생시키도록 트리거된다.

<53> 하이로 올라가는 제 1 OR 게이트 입력(62)에 의하거나 하이로 올라가는 제 2 OR 게이트 입력에 의해(이후에 논의될 내용과 같이), 슬롯 시퀀서(42)가 트리거되면, 신호는 각각의 경우에 슬롯 시퀀서 출력(68)에서 출력을 발생하면서, 슬롯 시퀀서(42)를 두 번 돌게 될 것이다. 상기 신호가 통과하는 첫 번째에는, 슬롯 시퀀서 출력(68)은 로우일 것이고, 두 번째에는 슬롯 시퀀서 출력(68)에서의 출력은 하이로 높아질 것이다. 슬롯 시퀀서 출력(68)으로부터의 상대적으로 넓은 출력은 펄스 제너레이터(pulse generator)(70)(블록 다이아그램의 형식으로 나타난)로 공급되고 이 제네레이터는 출력 값으로서 궁극적으로 좁은 타이밍 펄스(timing pulse)를 공급하게 된다. 당업자는 상기 좁은 타이밍 펄스가 컴퓨터(12)의 동작을 정확하게 개시하기에 바람직하다는 것을 인식할 것이다.

<54> 실행되고 있는 특정 명령(52)이 읽기 또는 쓰기 명령일 때, 또는 실행되고 있는 명령(52)이 연속해서 다음 명령(52)의 즉각적인 실행을 트리거하는 것이 바람직하지 않는 어떤 다른 명령일 때, i4 비트(66)은 '0'(low)이고 제 1 OR 게이트입력(62)은 그 결과 역시 로우이다. 해당 기술 당업자는 컴퓨터(12)와 같은 장치에서 이벤트들의 시간이 일반적으로 매우 중요하다는 것과 이것도 예외가 아니라는 것을 인식할 것이다. 슬롯 시퀀서(42)를 시험해보면, 당업자는 상기 링(ring)의 두번째 바퀴 신호(second lap)를 개시하기 위해서, 상기 신호가 NAND 게이트(58)를 지나서 계산된 후가 될 때까지, OR 게이트(60)로부터의 출력이 하이로 유지되어야 한다는 것을 인식하게 될 것이다. 그 결과, OR 게이트(60)로부터의 출력은, 상기 회로의 원하지 않은 계속된 발진(oscillation)을 막기 위해서 상기 두번째 바퀴 신호시(second lap) 낮아지게 될 것이다.

<55> 상기 설명으로 비추어 볼 때 이해될 수 있는 바와 같이, i4 비트(66)가 '0'이면, 슬롯 시퀀서(42)는-이후에 논의될 제 2 OR 게이트 입력(66)이 하이가 아니라고 가정해 보면- 트리거되지 않을 것이다.

<56> 상기 논의된 바대로, 각 명령(52)의 i4 비트(66)는, 명령어가 입력과 출력을 요구하는지에 따라 설정되기 보다는 그 명령이 읽기 또는 쓰기 유형의 명령인지 여부에 따라 설정된다. 명령(52)에서 남아있는 비트들(50)은 그 명령을 위한 상기 특정 op코드(opcode)의 나머지(remainder)를 제공한다. 읽기 또는 쓰기 유형의 명령의

경우에, 상기 비트들의 하나 이상은 그 특정 컴퓨터(12)에서, 데이터가 어디서 읽혀지는지 또는 어디로 쓰여지는지를 지시하는데 사용될 수 있다. 본 발명의 예에서, 쓰여지는 데이터는 항상 T 레지스터(44)(데이터 스택(34)의 맨 위)로부터 오지만, T 레지스터(44) 또는 실행될 수 있는 명령 영역(30)으로 선택적으로 읽혀질 수 있다. 본 발명에 대한 이 특정 실시예에서, 그것은 데이터나 명령들이 여기 설명된 방식으로 통신할 수 있고 그리고 비록 이것이 본 발명의 필수적인 것은 아니지만, 그 결과 명령들이 데이터 버스(16)로부터 직접 실행될 수 있기 때문이다.

<57> 비트들(50)의 포트가 존재한다면, 하나 이상은 포트들(38) 중 어느 것이 읽기 또는 쓰기를 위해 설정될 것인지를 지시하는데 사용될 것이다. 이 이후의 동작은 A 레지스터(40a), B 레지스터 등등과 같은 레지스터(40)를 지정하기 위해서 하나 이상의 비트들을 사용함으로서 선택적으로 수행된다. 그러한 예에서, 상기 지정된 레지스터(40)는, 상기 포트들(38)(그리고 역시, 상기 컴퓨터(12)가 통신하려고 시도하는, 메모리, 외부 통신 포트, 등등과 같은 기타 다른 가능한 엔티티) 각각에 대응되는 비트를 가지는 데이터로 프리로딩(preloading) 될 것이다. 예를 들면, 상기 특정 레지스터(40)에서 네 개 비트들 각각은, 상위 포트(38a), 오른쪽 포트(38b), 왼쪽 포트(38c) 또는 하위 포트(38d)의 각각에 대응할 수 있다. 그러한 경우에, 그러한 비트 위치들 중 '1'이 있는 곳에서, 통신은 대응되는 포트(38)을 통해 진행되도록 설정될 것이다. 여기서 앞서 논의한 바대로, 본 발명의 실시예에서, 읽기 op코드(opcode)가 단일 명령에서 통신을 위해 하나 이상의 포트(38)를 설정하는 것이 기대되는 반면, 비록 가능하긴 하지만, 쓰기 op코드는 단일 명령 통신을 위해 하나 이상의 포트(38)를 설정할 것으로 예상되지 않는다.

<58> 바로 아래의 예는 어떤 인접한 컴퓨터(12) 간의 통신에도 적용 가능하지만, 컴퓨터(12e)가 컴퓨터(12c)로 쓰기를 시도하게 되는 통신을 가정할 것이다. 쓰기 명령이 쓰고 있는 컴퓨터(12e)에서 실행되면, 선택된 쓰기 라인(20)(이 예에서, 컴퓨터(12e)와 컴퓨터(12c) 간의 쓰기 라인(20))은 하이로 설정되며, 만약 그에 대응하는 읽기 라인(18)이 이미 하이라면, 그때는 데이터가 상기 선택된 통신 포트(38)를 통해서 상기 선택된 위치로부터 즉시 전송된다. 또한, 만약 상기 대응하는 읽기 라인(18)이 이미 하이가 아니라면, 그때 컴퓨터(12e)는 대응하는 읽기 라인(18)이 하이로 올라갈 때까지 단순히 동작을 멈출 것이다. 읽기 또는 쓰기 유형의 명령이 있을 때, 컴퓨터(12a)를 멈추기 위한 메카니즘(또는, 더 정확하게는 더 이상의 동작을 하지 않도록 하는 것)이 여기서 앞서 논의되어 왔다. 간단히 말하면, 명령(52)의 op코드는 비트 포지션 i4(66)에서 '0'을 가질 것이고, 그래서 OR 게이트(60)의 제 1 OR 게이트 입력(62)은 로우이고, 슬롯 시퀀서(42)는 인에이블링 펄스(enabling pulse)를 발생시키기 위해 트리거 되지 않는다.

<59> 읽기 또는 쓰기 유형 명령이 완료되었을 때, 컴퓨터(12e)의 동작이 어떻게 재개되는지에 대해, 그에 대한 메카니즘은 다음과 같다: 컴퓨터들(12e 및 12c) 간의 읽기 라인(18) 및 그에 대응하는 쓰기 라인(20)이 모두 하이일 때, 두 개의 라인들(18 및 20)은 그것을 하이로 유지하고 있는 각각의 컴퓨터들(12)에 의해 해제될 것이다. (이 예에서, 수신 컴퓨터(12c)가 읽기 라인(20)을 하이로 유지하는 반면, 전송 컴퓨터(12e)는 쓰기 라인(18)을 하이로 유지하게 될 것이다.) 그러면 수신 컴퓨터(12c)는 두 라인들(18 및 20) 모두를 로우로 할 것이다. 실제 경우에, 수신 컴퓨터(12c)는 전송 컴퓨터(12c)가 읽기 라인(18)을 해제하기 전에 라인들(18 및 20)을 로우로 낮추려고 할 수도 있다. 그러나, 라인들(18 및 20)은 이미 하이로 높여져 있으며 약하게 낮춰지기 때문에, 그 라인(18 및 20)은 이를(18 및 20) 하이로 높게 유지하고 있는 컴퓨터(12)에 의해 해제될 때까지는 라인(18 또는 20)을 로우로 낮추려는 어떠한 시도도 실제로 성공하지 못할 것이다.

<60> 데이터 버스(16)의 두 라인들(18 및 20)이 로우로 낮춰지면, 이것은 "응답(acknowledge)" 상황이다. "응답(acknowledge)" 상황에 따라 각각의 컴퓨터들(12e 및 12c)은 자신들만의 내부 응답(acknowledge) 라인(72)를 하이로 높게 설정할 것이다. 도 5에서 보여지는 바와 같이, 응답(acknowledge) 라인(72)은 제 2 OR게이트 입력(64)을 공급한다. OR 게이트(60) 입력들(62 또는 64) 둘 중에 하나로의 입력이 OR 게이트(60)의 출력을 하이로 높게 올릴 것이므로, 이것은 여기서 앞서 설명된 방식으로 슬롯 시퀀서(42)의 동작을 개시하여, 명령 워드(48)의 다음 슬롯(54)에서 명령(52)이 실행될 것이다. 응답(acknowledge) 라인(72)은 가짜(spurious) 주소들이 주소 버스로 들어가는 것을 막기 위해서 다음 명령(52)이 디코딩될 때까지 하이로 유지된다.

<61> 어떤 경우에도, 실행되고 있는 명령(52)이 상기 명령 워드(48)의 슬롯 3의 위치에 있을 때면, 컴퓨터(12)는, 다음 대기하고 있는 18 비트 명령 워드(48)를 페치(fetch) 할 것이다. 이는 물론 비트i4(66)이 '0'이 아니거나, 슬롯 3에 있는 명령어가 Next 명령어가 아닐 때 가능하다.

<62> 실제의 경우에, 본 발명의 메카니즘은 "프리페칭(prefetching)" 명령에 대한 방법 및 장치를 포함하여 상기 페치(fetch)가 명령 워드(48)에 있는 모든 명령들(52)의 실행 종료 전에 시작될 수 있도록 한다. 그러나, 이는 또

한 비동기 데이터 통신들을 위한 본 발명 방법 및 장치에 대한 필요한 양상이 아니다.

<63> 컴퓨터(12e)가 컴퓨터(12c)에 쓰기를 하는(write) 상기 예는 상세하게 설명되었다. 상기 설명의 견지에서 알 수 있는 바와 같이, 컴퓨터(12e)가 컴퓨터(12c)에 먼저 쓰기를 시도하는지 또는 컴퓨터(12c)가 먼저 컴퓨터(12e)로부터 읽기(read)를 시도하는지에 관계없이, 동작들은 사실상 동일하다. 동작은 컴퓨터들(12 및 12c) 모두가 준비될 때까지 완료될 수 없으며, 그리고 어느 컴퓨터(12e 또는 12c)라도 먼저 준비되면, 다른 컴퓨터(12e 또는 12c)가 전송을 완료할 때까지, 제 1 컴퓨터(12)는 단순히 "수면 상태"(sleep)로 진입한다. 상술한 과정을 조사하는 또 하나의 방법은, 송신 컴퓨터(12e)와 수신 컴퓨터(12c)가 쓰기 및 읽기 명령들을 각각 실행하는 때에, 실제로 이들 모두가 수면으로 진입하지만, 마지막으로 트랜잭션에 진입하는 것은 읽기 라인(18) 및 쓰기 라인(20) 모두가 하이(high)인 때에, 거의 순간적으로 기상(awaken)하는 반면에, 트랜잭션을 개시하는 제 1 컴퓨터는 제 2 컴퓨터(12)가 과정을 완료할 준비가 될 때까지, 거의 무한정으로 수면을 유지할 수 있다.

<64> 발명자는 디바이스들 간의 효율적인 비동기 통신들을 가능하게 방법은 일종의 승인 신호 또는 승인 조건(acknowledge signal or condition)이라 믿는다.종래 기술에선 대부분의 통신은 클록화되어 있고, 수신 장치가 데이터를 받았는지를 송신 장치에서 알 수 있는 직접적인 방법이 없다. 검사합계(checksum)와 같은 방법으로 데이터가 제대로 받아졌는지를 확인하기 위해 실행 될수 있으나, 송신 장치는 동작이 끝났는지 알 길이 없다. 본원에서 설명되는 바와 같은 이러한 방법은 디바이스들간의 비동기 통신들을 허용하거나 적어도 실행되게 하는 필요한 승인 조건을 제공한다. 더욱이, 승인 신호 또는 승인 조건이 발생할 때까지, 하나 이상의 디바이스들로 하여금 "수면상태"로 들어갈 수 있도록 한다. 당연하게, 승인 조건은 (상호연결 데이터 버스(16)를 통해 또는 개별 신호 라인을 통해) 컴퓨터들(12) 간에서 송신되는 개별 신호에 의해 컴퓨터들(12) 간에서 통신될 수 있으며, 이러한 승인 신호는 본 발명의 이러한 양상의 범주 내에 들 것이다. 그러나, 본원에서 설명되는 본 발명의 실시예에 따르면, 승인 방법이 통신에 실질적으로 영향을 미치기 위해 임의의 신호, 클록 사이클, 타이밍 펄스, 또는 설명되는 것 이외의 모든 이러한 자원을 요구하지 않는다는 점에서, 관련된 훨씬 많은 경제성이 있음을 이해할 수 있다.

<65> 4개의 명령들(52)이 명령 워드(48)에 포함될 수 있기 때문에, 그리고 본 발명에 따르면, 명령 워드(48) 전체가 컴퓨터들(12) 간의 일 시점에서 통신할 수 있기 때문에, 이 발명 안에서 한 동작 명령어로 매우 작은 프로그램을 전송할 수 있는 기회가 주어진다. 예를 들어, 작은 "For/Next" 루프의 대부분은 단일 명령 워드(48)에서 구현될 수 있다. 도 6은 마이크로-루프(100)를 도시한다. 다른 종래기술 루프들과 다르지 않는 마이크로-루프(100)는 FOR 명령(102)과 NEXT 명령(104)을 갖는다. 명령 워드(48)(도 4)가 4개 만큼의 명령들(52)을 포함하기 때문에, 명령 워드(48)는 단일 명령 워드(48) 내에 3개의 연산 명령들(106)을 포함할 수 있다. 연산 명령들(106)은 본질적으로 프로그래머가 마이크로-루프(100)에 포함하기를 원할 수 있는 이용가능한 명령들 중 임의의 하나가 될 수 있다. 일 컴퓨터(12)로부터 또 하나의 컴퓨터로 전송될 수 있는 마이크로-루프(100)의 전형적인 예는 제 2 컴퓨터(12)의 RAM(24)으로부터 읽기(read)를 실행하거나 혹은 이에 쓰기(write)를 할수 있으며, 이에 따라 제 1 컴퓨터(12)가 이용가능한 RAM(24) 용량을 "차용(borrow)"할 수 있는 명령들 집합이 될 수 있다.

<66> FOR 명령(102)은 원하는 반복 횟수들을 나타내는 리턴 스택(28)상에 값을 푸시한다. 즉, 데이터 스택(34)의 상부에 있는 T 레지스터(44)상의 값은 리턴 스택(28)의 R 레지스터(29)에 푸시(PUSH)된다. 명령 워드(48)의 제 3 슬롯(54d)에 종종 위치되는 FOR 명령(102)은 사실상 임의의 슬롯(54)에 위치될 수 있다. FOR 명령(102)이 제 3 슬롯(54d)에 위치되지 않는 경우에, 이 명령 워드(48)의 나머지 명령들(52)은 마이크로-루프(100)로 진행하기 이전에 실행될 것이며, 일반적으로 마이크로-루프(100)가 다음 명령 워드로 로드될 것이다.

<67> 본 발명에 대해 현재에 설명되는 실시예에 따르면, 도 6에서 도시되는 NEXT 명령(104)은 특정 타입의 NEXT 명령(104)이다. 이는 이 명령이 제 3 슬롯(54d)(도 4)에 위치되기 때문이다. 본 발명에서의 이러한 실시예에 따르면, "통상적인(ordinary)" NEXT 명령(미도시)을 후에 따르는 특정 명령 워드(40)에서의 모든 데이터들은 어드레스(여기서, 어드레스는 for/next 루프가 시작하는 어드레스이다)로 가정된다. NEXT 명령(104)에 대한 오피코드(opcode)는 (본원에서 전술한 바와 같이, 이것이 제 3 슬롯(54d)인 경우에, 처음 2개의 비트값(bit)들이 명시적으로 써넣어지기 보다는 가정되는 명백한 예외를 제외하고) 4개의 슬롯들(54) 중 어디에 있는지와 관계없이 동일하다. 그러나, 이것이 제 3 슬롯(54d)에 있는 때에, NEXT 명령(104)을 따르는 어떤 어드레스 데이터가 있을 수 없기 때문에, 제 3 슬롯(54d)에 있는 NEXT 명령(104)은 MICRO-NEXT 명령(104a)으로 또한 가정될 수 있다. MICRO-NEXT 명령(104a)은 복귀할 어드레스로서 그것이 위치되어 있는 동일한 명령 워드(48)의 제 0 슬롯(54a)에 위치된 제 1 명령(52)의 어드레스를 사용한다. MICRO-NEXT INSTRUCTION(104a)은 또한 R 레지스터(29)로부터의 값을 취하며(이는 본래 FOR 명령(102)에 의해 거기에 PUSH 되었음), 이를 1만큼 감소시키며, 그리고 이후에 이를 R 레지스터(29)에 복귀시킨다. R 레지스터(29) 상의 값이 (0과 같은) 소정의 값에 도달한 때에, MICRO-NEXT

명령은 후속 명령 워드(48)를 적재함과 아울러 전술한 바와 같이 진행할 것이다. 그러나, MICRO-NEXT 명령(104a)이 소정의 값보다 큰 R 레지스터(29)로부터의 값을 읽는 때에, 이는 자신의 명령 워드(48)의 제 0 슬롯(54a)에서의 연산을 재개함과 아울러 제 0 슬롯에 후에 위치된 3개의 명령들(52) 전체를 실행할 것이다. 즉, MICRO-NEXT 명령(104a)은 본 발명에서의 이러한 실시예에서, 항시 3개의 연산 명령들(106)을 실시할 것이다. 일부 경우들에서, 모든 3개의 잠재적으로 이용가능한 명령들(52)을 사용하는 것이 원해지지 않을 수 있기 때문에, "무연산 명령어(no-op)" 명령이 요구되는 바와 같이, 슬롯들(54) 중 하나 이상을 채우는데 이용가능하다.

<68> 마이크로-루프들(100)은 전체적으로 단일 컴퓨터(12) 내에서 사용될 수 있음이 주목되어야 한다. 사실상, 이용가능한 기계 언어 명령들의 집합 모두가 연산 명령들(106)로서 이용가능하며, 그리고 마이크로-루프들의 응용 및 사용은 오직 프로그래머의 창조력에 의해서만 제한된다. 그러나, 본질적으로 데이터 버스(16)로부터 직접적으로 비롯된 내부의 명령들(52)을 실행하기 위해, 단일 명령 워드(48) 내의 전체 마이크로-루프(100)를 실행하는 성능과 컴퓨터(12)로 하여금 이웃 컴퓨터(12)에 명령 워드(48)를 송신하게 하는 성능이 결합하게 되면, 이는 컴퓨터(12)로 하여금 그 이웃들의 자원들을 이용하게 하는 강력한 툴을 제공한다.

<69> 소형 마이크로-루프(100)(이들 모두는 단일 데이터 워드(48) 내에 포함되어 있음)는 본원에서 설명되는 바와 같이 컴퓨터들(12) 간에서 통신될 수 있으며, 이는 본원에서 설명되는 바와 같이 명령 워드(48)에 포함된 임의의 다른 명령들 세트와 마찬가지로 수신 컴퓨터(12)의 통신 포트(38)로부터 직접적으로 실행될 수 있다. 이러한 종류의 "마이크로-루프"(100)에 대한 많은 사용들이 있지만, 전형적인 사용은 일개의 컴퓨터(12)가 이웃 컴퓨터(12)의 메모리 상에 일부 데이터를 저장하기를 원하는 경우가 될 것이다. 예를 들어, 이는 먼저 그 이웃 컴퓨터에 명령을 송신할 수 있는데, 이때에 그 컴퓨터는 이웃 컴퓨터에게 인입(incoming) 데이터 워드를 특정 메모리 어드레스에 저장하고, 이후에 그 어드레스를 증가시키며, 그리고 소정의 반복 횟수(여기서, 횟수는 데이터 워드들이 전송되는 횟수이다)로 반복하도록 명령한다. 데이터를 다시 읽기(read) 위해, 제 1 컴퓨터는 제 2 컴퓨터(여기에서 저장을 위해 사용된 것)로 하여금 유사한 마이크로-루프를 사용하여, 저장된 데이터를 다시 제 1 컴퓨터에 쓰기(write)를 실행하도록 명령할 것이다.

<70> 본원에서 설명되는 다이렉트 실행 양상과 더불어 마이크로-루프(100) 구조를 사용함으로써, 컴퓨터(12)는 필요한 데이터 저장소가 각 개별 컴퓨터(12) 내에 구축된 비교적 작은 용량을 초과하는 때에, 초과 데이터 저장을 위해 다른 휴식중인 이웃 컴퓨터(12)를 사용할 수 있다. 상기 예가 데이터 저장의 관점에서 설명되었지만, 동일한 기법은 컴퓨터(12)로 하여금 그 이웃 컴퓨터가 자신의 연산 자원들(computational resources)을 공유하게 하는데 동일하게 사용될 수 있는데, 이는 다른 컴퓨터(12)로 하여금 일부 동작들을 수행하고, 그 결과를 저장하며, 그리고 소정의 횟수로 반복하게 하는 마이크로-루프(100)를 생성함으로써 된다. 이해될 수 있는 바와 같이, 본 발명의 마이크로-루프(100) 구조가 사용될 수 있는 방식들의 개수는 거의 무한정이다.

<71> 본원에서 전술한 바와 같이, 본 발명에서의 현재에 설명되는 실시예에서, 데이터 또는 명령들은 본원에서 설명되는 방식으로 통신될 수 있으며, 따라서, 명령들은 본질적으로 데이터 버스(16)로부터 직접적으로 실행될 수 있다. 즉, 명령들을 RAM(24)에 저장하고 실행 전에 이들로부터 호출할 필요가 없다. 대신에, 본 발명의 이러한 양상에 따르면, 통신 포트(38)상에서 수신되는 명령 워드(48)는 본질적으로 이것이 RAM(24)이나 ROM(26)로부터 호출되는 경우에 취급되는 것과 다르게 취급되지 않는다. 이러한 차이점이 없음은 종래기술 논의에서 나타났지만, 설명되는 컴퓨터들(12)의 연산에 관하여, 명령 워드들(48)이 어떻게 페치(Fetch)되고 사용되는지에 대한, 하기의 보다 구체적인 설명은 본 발명의 이해를 돕게 될 것이다.

<72> 이용가능한 기계 언어 명령들 중 하나는 FETCH 명령이다. FETCH 명령은 어디서부터 18 비트 워드를 페치할지를 결정하기 위해 A 레지스터(40a) 상의 어드레스를 사용한다. 당연하게, 프로그램은 A 레지스터(40a) 상에 정확한 어드레스를 위치시키도록 미리 규정하였을 필요가 있을 것이다. 본원에서 전술한 바와 같이, A 레지스터(40a)는 18비트 레지스터이며, 이에 따라 페치가 발생할 수 있는 모든 잠재적인 자원들이 차별화될 수 있도록 이용가능한 충분한 범위의 어드레스 데이터가 있다. 즉, ROM에 할당된 어드레스들 범위 및 RAM에 할당된 다른 어드레스들 범위가 있으며, 그리고 포트들(38) 각각에 대한 특정 어드레스들 및 외부 I/O 포트들(39)에 대한 특정 어드레스들이 있다. FETCH 명령은 자신이 T 레지스터(44) 상에서 페치하도록 항상 18 비트를 위치시킨다.

<73> 대조적으로, 본원에서 전술한 바와 같이, (데이터와 대조적으로) 실행가능 명령들은 명령 레지스터(30a)에 일시적으로 저장된다. 18 비트 명령 워드(48)를 명령 레지스터(30a)로 "페치" 하기 위한 어떤 특정 커맨드가 없다. 대신에, 명령 레지스터(30a)에 더 이상의 실행가능 명령들이 남아 있지 않은 때에, 컴퓨터는 "후속" 명령 워드(48)를 자동으로 페치할 것이다. "후속" 명령 워드가 어디에 위치되어 있는지는 "프로그램 카운터"(P 레지스터(40c))에 의해 결정된다. P 레지스터(40c)는 명령 워드들(48)의 시퀀스가 RAM(24)이나 ROM(26)으로부터 페치되

는 경우에서와 같이, 종종 자동으로 증가된다. 예를 들어, JUMP 또는 CALL 명령은, JUMP 또는 CALL 명령 이후에 증가되기 보다는, 현재에 적재된 명령 워드(48)의 나머지에 있는 데이터에 의해 지정되는 어드레스로 P 레지스터(40c)가 적재되게 할 것이다. P 레지스터(40c)가 이후에 하나 이상의 포트들(38)에 대응하는 어드레스로 적재되는 때에, 후속 명령 워드(48)는 포트들(38)로부터 명령 레지스터(30a)로 적재될 것이다. P 레지스터(40c)는 명령 워드(48)가 포트(38)로부터 명령 레지스터(30a)로 검색된 때에, 또한 증가하지 않는다. 오히려, 이는 P 레지스터(40c)를 변경하기 위한 특정 JUMP 또는 CALL 명령이 실행될 때까지, 이 동일한 포트 어드레스를 계속해서 보유할 것이다. 즉, 일단 컴퓨터(12)가 포트(38)로부터 후속 명령을 탐색하도록 명령받게 되면, 이는 후속 명령 워드(48)에 대해 메모리(RAM(24) 또는 ROM(26)으로의 다시 검색과 같은 다른 곳을 탐색하도록 명령받을 때까지, 이 동일한 포트(38)(또는 포트들(38))로부터의 명령들을 계속하여 탐색할 것이다.

<74> 전술한 바와 같이, 컴퓨터(12)는 현재의 명령 워드(48)에 더 이상의 실행가능 명령들이 남아 있지 않은 때에, 페치될 후속의 18개의 비트들이 명령 레지스터(30a)에 위치될 것임을 알고 있다. 기본적으로, JUMP 또는 CALL 명령 이후에(또는 또한 본원에서 구체적으로 논의되지 않을 일정한 다른 명령들 이후에) 더 이상의 실행가능 명령들이 현재의 명령 워드(48)에 남아 있지 않는데, 이는 당연히, JUMP 또는 CALL 명령 이후에 남아 있는 18개 비트의 명령 워드가 JUMP 또는 CALL 명령에 의해 참조되는 어드레스에 전용이기 때문이다. 이를 진술하는 또 하나의 방식은 전술한 프로세스들이 많은 방식들에서 고유하다는 것이며, 이는 JUMP 또는 CALL 명령이 선택적으로 단지 메모리 어드레스 등에 대한 것이라기보다는 포트(38)에 대한 것이 될 수 있다는 사실을 포함하지만, 이에 국한되지는 않는다.

<75> 본원에서 전술한 바와 같이, 컴퓨터(12)는 일 포트(38)로부터 또는 임의의 포트들의 그룹(38)으로부터 후속 명령을 탐색할 수 있다. 따라서, 어드레스들은 다양한 포트들(38)의 조합들에 대응하도록 제공된다. 예를 들어, 컴퓨터가 포트들(38)의 그룹으로부터 명령을 페치하도록 명령받은 때에는, 선택된 포트들(38) 중 임의의 하나로부터 먼저 이용가능한 명령 워드(48)를 수락할 것이다. 만일, 어떤 이웃 컴퓨터(12)가 이러한 포트들(38) 중 임의의 하나에 아직 써넣기 시도를 하지 않은 경우에, 당해의 컴퓨터(12)는 상세하게 전술한 바와 같이, 이웃 컴퓨터가 선택된 포트(38)에 쓰기(write)할 때까지 "수면상태로 진입" 할 것이다.

<76> 도 7은 전술한 다이렉트 실행 방법(120)의 일 예를 도시하는 흐름도이다. 연산들의 "정상적인" 흐름은 본원에서 전술한 바와 같이, 더 이상의 실행가능 명령들이 명령 레지스터(30a)에 남아 있지 않은 때에 시작할 것이다. 이 때에, 컴퓨터(12)는 "워드 페치(fetch word)" 연산(122)에 의해 표시된 바와 같이(여기서, 용어 "페치"는 실제 FETCH 명령이 사용되지 않는다는 점에서, 본원에서 일반적인 의미로 사용됨을 주목하자) 또 하나의 명령 워드를 "페치"할 것이다. 이 연산은 (도 7의 흐름도에서의 "어드레스" 결정 연산에 의해 표시된 바와 같이) P 레지스터(40c)의 어드레스에 따라 수행될 것이다. 만일 P 레지스터(40c)의 어드레스가 RAM(24) 또는 ROM(26) 어드레스인 경우에, 후속 명령 워드(48)는 "메모리로부터의 페치" 연산(126)에서, 지정된 메모리 위치로부터 검색될 것이다. 반면에, P 레지스터(40c)의 어드레스가 포트(38) 또는 포트들(38)의 어드레스(메모리 어드레스가 아님)인 경우에, 후속 명령 워드(48)는 "포트로부터의 페치" 연산(128)에서, 지정된 포트 위치로부터 검색될 것이다. 어느 경우에서든지, 검색되는 명령 워드(48)는 "명령 워드 검색" 연산(130)에서 명령 레지스터(30c)에 위치된다. "명령 워드 실행" 동작(132)에서, 명령 워드(48)의 슬롯들(54)에 있는 명령들은 본원에서 전술한 바와 같이 순차적으로 수행된다.

<77> "점프" 결정 동작(134)에서, 명령 워드(48)의 연산들 중 하나가 JUMP 명령인지, 또는 본원에서 전술한 바와 같이, 계속되는 "정상적인" 진행으로부터 연산을 전환시킬 다른 명령인지가 결정된다. 만일 "JUMP" 명령이면, JUMP 명령 (또는 이러한 다른 것) 이후 명령 워드(48) 내에 제공되는 어드레스가 "P 레지스터 로드" 동작(136)에서 P 레지스터(40c)에 제공되고, 시퀀스는 도 7의 흐름도에 나타낸 바와 같이 "워드 페치" 동작(122)에서 다시 시작된다. 만일 "JUMP" 명령이 아니면, "포트 어드레스" 결정 동작(138)으로 나타낸 바와 같이, 다음 동작은 마지막 명령 페치가 포트(38)로부터였는지, 아니면 메모리 어드레스로부터 였는지에 의존한다. 만일 마지막 명령 페치가 포트(38)로부터 였다면, P 레지스터(30a)에 대한 어떠한 변경도 이루어지지 않고, 시퀀스는 "워드 페치" 동작(122)으로부터 시작하여 반복된다. 한편, 마지막 명령 페치가 메모리 어드레스(RAM(24) 또는 ROM(26))로부터였다면, 도 7의 "P 레지스터 증분" 동작(140)에 의해 나타낸 바와 같이, "워드 페치" 동작(122)이 이루어지기 전에, P 레지스터(30a) 내의 어드레스가 증분된다.

<78> 상기 설명은 실제 동작 단계들을 나타내는 것으로 의도되지 않는다. 대신에, 본 발명의 설명되는 실시예에 따라 수행되는 것으로부터 비롯되는 다양한 결정들 및 동작들의 흐름도이다. 실제로, 이러한 흐름도는, 제시되어 설명되는 각 동작이 개별적인 별개의 순차 동작을 필요로 하는 것을 의미하는 것으로서 이해되서는 안된다. 실제

로, 도 7의 흐름도에서 설명되는 많은 동작들은 일반적으로 동시에 이루어진다.

<79> 도 8은 본 발명에 따라 컴퓨터를 경계시키는(alerting) 개선된 방법에 대한 일례를 도시한 흐름도이다. 이전에 설명된 바와 같이, 설명되고 있는 실시예의 컴퓨터들(12)은 입력을 기다리는 동안 "수면(sleep) 상태가 된다". 도 1 내지 5와 관련하여 설명되는 실시예에서처럼, 이러한 입력은 이웃하는 컴퓨터(12)로부터 올 수 있다. 대안적으로, 또한 본원에서 이전에 설명된 바와 같이, 다이(14)의 가장자리에 인접하는 통신 포트들(38)을 갖는 컴퓨터들(12)은, 이러한 컴퓨터(12) 내에 설계되거나, 또는 그렇지 않으면 컴퓨터(12)의 외부에 있지만 그와 결합되는 부가 회로를 구비함으로써, 이러한 통신 포트(38)가 외부 I/O 포트(39)로서 기능할 수 있게 한다. 어느 경우이든, 본 발명에 따른 결합 내에서는, 입력이 수신될 때 "수면하고 있는" 컴퓨터(12)가 깨어날 준비를 하고 있다가 정해진 동작을 재빨리 취할 수 있다는 부가적인 장점을 제공할 수 있다. 따라서, 또한 본 발명은, 입력들이 외부 입력장치로부터 오든, 어레이(10) 내의 다른 컴퓨터(12)로부터 오든지 간에, 입력들을 처리하기 위해 인터럽트들을 이용하는 방법에 대한 그 대안을 제공한다.

<80> 인터럽트를 처리하기 위해 컴퓨터(12)로 하여금 그것이 행하고 있는 것을 중단(stop)[또는 중지(paues)]하게 하는 대신, 본원에서 설명되는 개선된 조합은 전술한 바와 같이, 컴퓨터(12)가 "수면하고 있지만 경계하는(asleep but alert)" 상태에 있을 수 있게 한다. 따라서, 1개 이상의 컴퓨터들(12)이 소정의 입력들을 수신하고 그에 입각하여 동작하도록 지정될 수 있다. 이러한 특징이 이용될 수 있는 많은 방법들이 있지만, 단지 1개의 이러한 "컴퓨터 경계 방법"을 예시하는 일례가 도 8에 도시되어 있으며, 여기에서는 참조 문자 150으로 열거되어 있다. 도 8로부터 알 수 있는 바와 같이, "경계 상태 진입(enter alert state)" 동작(152)에 있어서, 컴퓨터(12)는 "수면 모드가 되고", 이에 따라 이웃 컴퓨터(12)로부터, 또는 1개 이상(모두 4개)의 이웃 컴퓨터들로부터 입력을 기다리거나, 또는 다이 상의 주변부(edge) 컴퓨터(12)의 경우에는, 외부 입력, 또는 이웃 컴퓨터(12)로부터의 입력들 및/또는 외부 입력들의 어떠한 결합을 기다리게 된다. 본원에서 설명되는 바와 같이, 컴퓨터(12)는 "수면 모드가 되어", 읽기 또는 쓰기 동작중 어느 하나의 완료를 기다릴 수 있다. 본 예에서 설명되는 바와 같이, 어떠한 가능한 "입력"을 기다리기 위해 컴퓨터(12)가 이용되는 경우, 기다리고 있는 컴퓨터는 그의 읽기 라인(18)(read line)을 하이로 설정하고, 이웃하는 또는 외부의 소스로부터 "쓰기"(write)을 기다린다고 가정하는 것은 당연할 것이다. 실제로, 이것이 현재 통상의 조건으로 예측되고 있다. 하지만, 기다리고 있는 컴퓨터(12)가 그의 쓰기 라인(20)(write line)을 하이로 설정하고, 그에 따라 이웃하는 또는 외부의 소스가 그것을 "읽기"(read)할 때에 깨어나게 되는 것도 본 발명의 범위 내에 있다.

<81> "기상(awaken)" 동작(154)에 있어서, 이웃하는 컴퓨터(12) 또는 외부 장치(39)가 기다리고 있는 트랜잭션을 완료했기 때문에, 수면하고 있는 컴퓨터(12)는 동작을 재개한다. 기다리고 있는 트랜잭션이 실행될 명령 워드(48)의 수신이었다면, 컴퓨터(12)는 그 내에서 명령들을 실행하도록 진행된다. 기다리고 있는 트랜잭션이 데이터의 수신이었다면, 컴퓨터(12)는 큐(queue) 내의 다음 명령(이것은 본 명령 워드(48) 내의 다음 슬롯(54)의 명령이다)을 실행하도록 진행하거나, 또는 그렇지 않으면 다음 명령 워드(48)가 로드되고, 다음 명령은 그 다음 명령 워드(48)의 슬롯 0 내에 있게 될 것이다. 어느 경우이든, 상기 설명된 방식으로 이용되고 있는 동안, 다음 명령은 막 수신된 입력을 처리하기 위해 1개 이상의 명령들의 시퀀스를 시작할 것이다. 이러한 입력을 처리하기 위한 옵션들은 미리 정의된 어떠한 기능을 내부적으로 수행하기 위해 반응하고, 어레이(10) 내의 1개 이상의 다른 컴퓨터들과 통신하거나, 또는 (정해진 조건하에서 인터럽트들이 무시될 수 있는 통상의 종래 기술과 마찬가지로) 심지어 입력을 무시하는 것을 포함할 수 있다. 이러한 옵션들은 도 8에서 "입력에 대해 작용하는(act on input)" 동작(156)으로서 도시된다. 유념해야 할 점은, 특정한 경우에는 입력의 콘텐츠는 중요하지 않을 수도 있다는 것이다. 예를 들어 어떠한 경우에는, 외부 장치가 통신을 시도했다는 것만으로도 중요한 사실이 될 수 있다.

<82> 도 8에 도시된 방법에 있어서, 컴퓨터(12)에 "경계" 컴퓨터로서 작용하는 태스그(task)가 할당되면, 도 8에 나타낸 바와 같이 그것은 "수면하고 있지만 경계하는" 상태로 돌아가는 것이 일반적이다. 하지만, 모니터되고 있는 특정의 입력 또는출력들을 더 이상 모니터할 필요가 없을 때, 또는 태스크를 어레이 내의 컴퓨터들(12)중 어떠한 다른 것에게 넘기는 것이 보다 편리할 때, 이러한 옵션은 컴퓨터(12)에게 어떠한 다른 태스크를 항상 자유롭게 할당할 수 있다.

<83> 당업자라면 상기 설명된 이러한 동작 모드가 통상적인 인터럽트 이용에 대한 보다 효율적인 대안으로서 유용하게 될 것임을 인식할 것이다. 컴퓨터(12)가 1개 이상의 그 읽기 라인들(18)(또는 쓰기 라인(20))을 하이로 설정하면, 이것은 "경계" 상태라 불릴 수 있다. 이러한 경계 상태에서, 컴퓨터(12)는 하이로 설정된 읽기 라인 또는 라인들(18)에 대응하는 데이터 버스(16) 상에서 자신에게 전송된 임의의 명령를 즉시 실행할 준비를 하거나, 또는 데이터 버스(16) 상으로 전송되는 데이터에 대해 동작할 준비가 된다. 이용가능한 컴퓨터들(12)의 어레이가

있는 경우, 임의의 소정의 시간에서, 상기 설명된 경계 상태가 되도록 1개 또는 그 이상의 컴퓨터가 이용될 수 있으며, 이에 따라 지정된 입력들의 세트중 임의의 것에 의해 동작이 시작될 것이다. 이것은 컴퓨터의 "주의를 얻기 위해(get the attention)" 통상의 인터럽트 기술을 이용하는 것에 대해 바람직한바, 인터럽트는 컴퓨터로 하여금 인터럽트 요청에 응답하여 어떠한 데이터를 저장해야만 하고, 어떠한 데이터를 로드해야만 하는 등등을 필요로 하기 때문이다. 대조적으로, 본 발명에 따르면, 컴퓨터를 경계 상태에 두고 필요한 입력을 기다리는 것에 전용할 수 있는바, 따라서 입력에 의해 제공되는 명령들의 실행을 시작함에 있어서 어떠한 단일의 명령 주기도 낭비하지 않게 된다. 다시 한번 주목할 사항으로서, 현재 설명되고 있는 실시예에서, 경계 상태에 있는 컴퓨터들은 실제로 "수면하고 있지만 경계하는" 상태에 있게 되는 바, 이것은 이들이 본질적으로 어떠한 전력도 이용하지 않고 있다는 의미에서 "수면하고"있지만, 이들이 입력에 의해 즉시 동작을 개시할 수 있다는 점에서 "경계" 상태에 있음을 의미한다. 하지만, "수면하고" 있지 않다고 하더라도 컴퓨터에서 "경계" 상태를 구현하는 것도 본 발명의 범위 내에 있다. 설명되는 경계 상태는 본질적으로, 그렇지 않으면 통상의 종래 기술의 인터럽트(하드웨어 인터럽트 또는 소프트웨어 인터럽트)가 이용되게 되는 임의의 상황에서도 이용될 수 있다.

<84> 도 9 컴퓨터 경계 방법(computer alert method)의 또 다른 예이다. 이는 모니터링하는 프로세서(12f 도1)와 일이 할당된 다른 프로세서(12g)간의 상호 교신에 관한 한 예이다. 도 8에서 볼 수 있듯이 각각의 프로세서(12f와 12g)를 나타내는 2개의 독립적인 흐름도가 있다. 이는 본 발명의 협동적 코프로세서(cooperative coprocessor) 접근의 성질을 나타내는바, 여기서 각각의 컴퓨터들(12)은 본 출원에 설명되는 바와같이 상호 작용이 달성되는 때를 제외하고는 독립적으로 실행할 수 있는 그 자신의 주어진 일들이 있다.

<85> 컴퓨터(12f)와 관련하여, "경계 상태 진입"(enter alert status" 동작(152), "기상"(154) 동작, 및 "입력에 대한 작용"(act on input) 동작 (156) 각각은, 컴퓨터 경계 방법(150)의 제 1 예와 관련되어 본원에서 이전에 설명된 바와같이 이루어진다. 하지만, 이러한 실시예는 컴퓨터들(12f 및 12g) 사이의 상호작용에 대한 잠재적인 필요성을 예측하고 있기 때문에, 이후 "입력에 대한 작용" 동작(156) 후에 컴퓨터(12f)는 "정보 전송?" 결정단계(158)로 들어가며, 그 프로그래밍에 따라, 방금 받은 입력이 다른 컴퓨터(12g)의 주목을 필요로 하는 지가 결정된다. 전송이 필요 없다고(아니오) 결정되면 컴퓨터(12f)는 "경계 상태"로 들어가거나 또는 이전에 설명된 것과 같은 어떠한 다른 대안적인 상태(alternative)로 들어간다. 만약 "예"라고 결정되면, 컴퓨터(12f)는 "다른 컴퓨터로 전송" 동작(160)에서 컴퓨터(12g) 와의 통신을 개시한다. 여기서 유념할 점은, 프로그래머의 선택에 따라, 컴퓨터(12f)는 외부 장치(82)로부터의 입력에 대응하여 자체 생성한 명령어와 같은 명령어를 전송할 수 있으며, 또는 외부 장치(82)로부터 수신한 것 같은 명령어들을 전송할 수 있다. 대안적으로, 컴퓨터(12f)는 컴퓨터(12g)에게 데이터를 전달할 수 있는바, 이러한 데이터는 컴퓨터(12f)에서 내부적으로 생성될 수 있으며 또는 외부 장치(82)로부터 전달되는(pass through) 데이터일 수 있다. 또 다른 대안에 있어서, 컴퓨터(12f)는 특정한 경우, 컴퓨터(12g)가 외부 장치(82)로부터 입력을 수신하는 때에 컴퓨터(12g)로부터 읽기를 시도할 수 있다. 이와 같은 이용 가능성은 모두 프로그래머에 달려 있다.

<86> 일반적으로 컴퓨터(12g)는 "주요 기능 실행" 동작(162)에서 나타낸 바와 같이, 무엇이든 자신의 할당된 주요 태스크를 달성하기 위해 코드를 실행한다. 하지만, 프로그래머가 컴퓨터들(12f 및 12g) 사이에서 가끔씩의 상호작용이 바람직하다고 결정하면, "입력 검색"(look for input) 동작(166)으로 나타낸 바와같이, 프로그래머는 컴퓨터(12g)가 하나 이상의 이웃 컴퓨터들이 통신을 시도했는지를 확인하기 위해 종종 포우즈(Pause)하도록 구성할 수도 있다. "입력?" 결정 동작(168)은 통신 시도가 대기하고 있을시(예를 들어 컴퓨터(12f)가 컴퓨터(12g)에 이미 쓰기를 개시한 경우)에 실행된다. 이미 통신 시도가 있었으면("예"라는 결정), 본원에서 이미 설명된 바와같이, 컴퓨터(12g)는 "다른 컴퓨터로부터 수신" 동작(170)에서 통신을 완료한다. 만약 "아니요"라는 결정이 되면, 컴퓨터(12g)는 도8에 도시된 것과 같이 할당된 기능 실행을 개시한다. "다른 컴퓨터의 수신" 동작(170)이 끝나면, 컴퓨터(12g)는 "입력에 대한 작용" 동작(172)를 실행한다. 마찬가지로, 상기 언급된 봐와 같이 프로그래머가 컴퓨터 12g가 입력으로서 명령어를 기대한다면 이미 명세된 바와 같이 컴퓨터 12g 는 명령어를 실행할 것이다. 또 다른 경우에는, 컴퓨터 12g는 처리할 데이터를 기대하도록 프로그램될 수 있다.

<87> 도 9의 예시에서는 "입력에 대한 작용" 동작(172)이 끝난 후 컴퓨터 12g는 주요 기능 실행(162)을 개시한다. 그러나, 더욱 복잡한 경우가 발생할 수 있다. 예를 들어, 컴퓨터로 12f부터 받은 입력에 따라 기존에 진행하던 주요 기능을 취소하여 새로운 태스크를 실행하거나, 잠시 중단하여 더 많은 외부로부터 입력을 기다리게 프로그램될 수 있다. 해당 기술의 당업자가 인식하듯, 다수의 응용이 가능하며 이는 프로그래머의 상상력에 제한될 따름이다.

<88> 본원에서 설명되는 본 발명의 실시 예에 따르면, 소정의 컴퓨터(12)는 태스크를 수행하는 동안에 인터럽트될 필요가 없는데, 이는 입력들을 모니터링하고 처리하는 잠재적으로 인터럽트가 요구될 수 있는 태스크가 개별의 컴

퓨터(12)에게 할당되기 때문이다. 그러나, 또 하나의 태스크를 핸들링하느라 바쁜 컴퓨터(12)는, 컴퓨터가 입력에 대해 자신의 포트들(38)을 주시하도록 그 프로그래밍이 규정하지 않는 경우에 방해될 수 없음을 주목해야 한다. 따라서, 컴퓨터(12)로 하여금 다른 입력들을 탐색하도록 중지하거나 퍼즈(pause)하도록 하는 것이 때때로 필요하다. 여기서 인지해야 할 중요한 점은 본원에서 명세된 바가 단일 프로세서안에서 실행되는 태스크가 이제 여러 프로세서에 의해 나누어진 "협력적인 멀티 태스킹(Cooperative Multitasking)"이라 표현되는 컴퓨팅 파라다임의 한 예라는 것이다.

<89> 본 발명에서 발명의 가치나 범주를 바꾸지 않고도 다양한 변경이 가능하다. 예를 들어, 특정 컴퓨터에 예를 들어 본원에서 발명이 묘사되었으나, 다수의 발명 특성은 다른 컴퓨터 디자인이나 다른 형의 컴퓨터 어레이 등등에 적용가능하다.

<90> 유사하게, 본 발명은 주로 단일 다이(die)상에 존재하는 어레이(10)의 컴퓨터(12)들의 상호 통신에 연관되어 설명되었으나, 컴퓨터(12)와 지정된 메모리간의 통신 또는 컴퓨터(12)와 외부장치간의 통신과 같은 상호장치 통신을 실행하기 위해서도, 동일한 법칙과 방법이 사용되거나 변경, 이용될 수 있다.

<91> 비록, 개선된 컴퓨터 어레이(10), 컴퓨터(12), 마이크로 루프(100), 직접 실행법(120), 그리고 이와 관련된 장치, 및 컴퓨터 경계 방법(150)에 대한 특정한 실시예들이 본 명세서에 개시되었으나, 아직 생각해내지 못한 대단히 많은 수의 응용예가 있을 것이라고 예측된다. 사실, 본 발명에 따른 방법 및 장치는 매우 다양한 사용용도에 적용할 수도 있다는 것이 본 발명의 장점중의 하나이다.

<92> 상기 전술된 실시예들은 모든 가능성 있는 본 발명의 실시예들의 일부일 뿐이며, 본 발명의 해당분야의 당업자들은 본 발명의 사상 및 범위를 벗어나지 않는 한도에서 수 많은 변경과 수정이 가능하다는 것을 즉시 알 수 있을 것이다. 따라서, 발명의 도시된 바는 본 발명의 제한성을 나타내는 것으로 의도된 것이 아니며, 그리고 첨부된 특허청구항이 본 발명의 전체 범위를 포함하는 것으로 해석되어져야 한다.

산업상 이용 가능성

<93> 본 발명에서 컴퓨터 어레이(10), 컴퓨터(12), 마이크로 루프(100), 직접 실행법(120),그리고 이와 관련된 장치, 및 컴퓨터 경계 방법(150) 등은 다양한 컴퓨터 응용에 이용되도록 의도되었다. 본 발명은 방대한 컴퓨터 계산력이 필요하며 파워 소비와 열 방출의 고려가 중요한 분야에서 유용할 것으로 예상된다.

<94> 본원에서 기술된 바와 같이, 본 발명의 응용 가능성은 어레이에 있는 컴퓨터 간의 정보와 자원 공유에 있어 속도와 그 다재 다능함이 급격히 향상될 것이라는 점이다. 또한, 컴퓨터 어레이와 다른 기기의 통신은 상술된 방법과 장치에 의해 향상될 것이다.

<95> 본 발명에서 컴퓨터 어레이(10),컴퓨터(12), 마이크로 루프(100), 직접 실행법(120), 그리고 이와 관련된 장치, 및 컴퓨터 경계 방법(150)은 즉시 생산될 수 있으며 기존하는 태스크, 입/출력 장치 등등과 복합되어 사용될 수 있으며, 본원에서 제시된 바와 같은 장점이 제공되기에 때문에 본 발명은 업계에서 쉽게 받아 들여질 것으로 예상된다. 이런 이유와 또 다른 이유들로 인해 본 발명의 효율성과 산업적 응용 가능성은 그 범위에 있어서 중요하고 그 지속 기간이 오래 유지될 것으로 기대된다.

도면의 간단한 설명

<22> 도 1은 본 발명에 따른 컴퓨터 어레이를 도시한 도면이다.

<23> 도 2는 도1에 도시된 컴퓨터들의 서브세트와 데이터 버스들의 연결을 상세히 도시한 도면이다.

<24> 도 3은 도1 및 도2에 도시된 컴퓨터들중 하나의 일반적인 레이아웃을 도시한 도면이다.

<25> 도 4는 발명의 실시예에 따른 명령 워드(48)를 도시한 도면이다.

<26> 도 5는 도 3의 슬롯 시퀀서(42)를 도시한 도면이다.

<27> 도 6은 본 발명의 일실시예에 따른 마이크로-루프에 대한 일례를 도시한 흐름도이다.

<28> 도 7은 포트로로부터의 명령들을 실행하는 방법에 대한 일례를 도시한 도면이다.

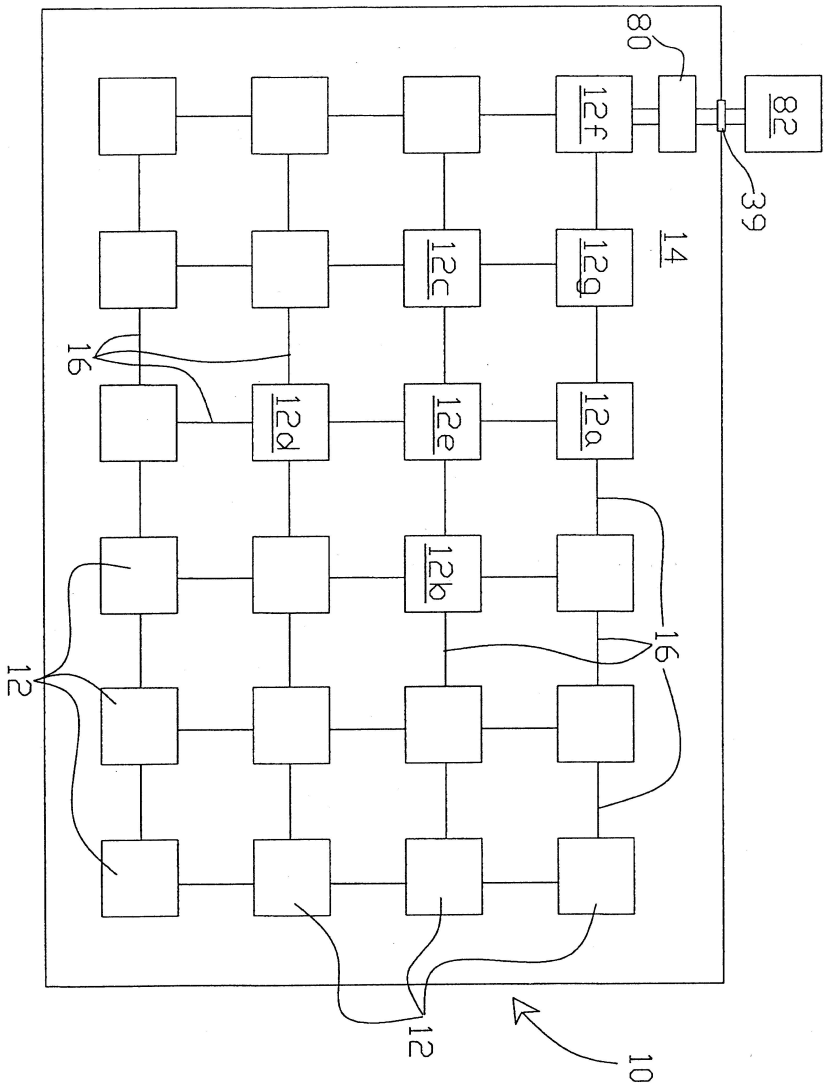
<29> 도 8은 컴퓨터를 깨우는 향상된 방법에 대한 일례를 도시한 흐름도이다.

<30> 도 9는 비활성화된 프로세서를 깨우고 입력 데이터를 깨워진 프로세서로부터 상기 입력 데이터를 처리 가능한

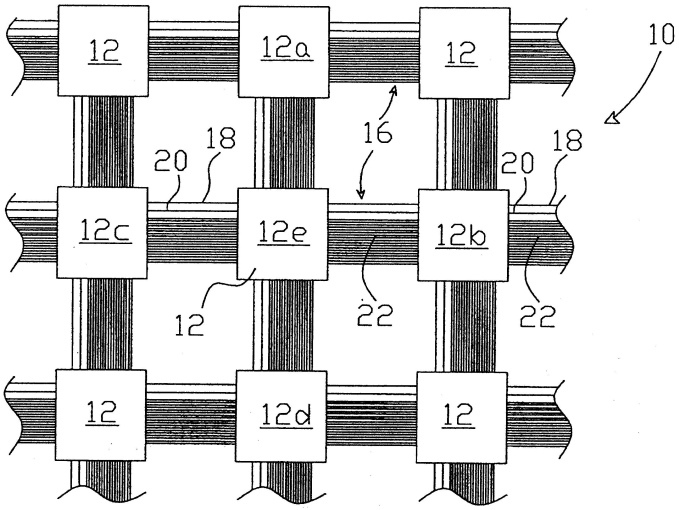
실행 프로세서로 전송하는 방법을 도시한 흐름도이다.

도면

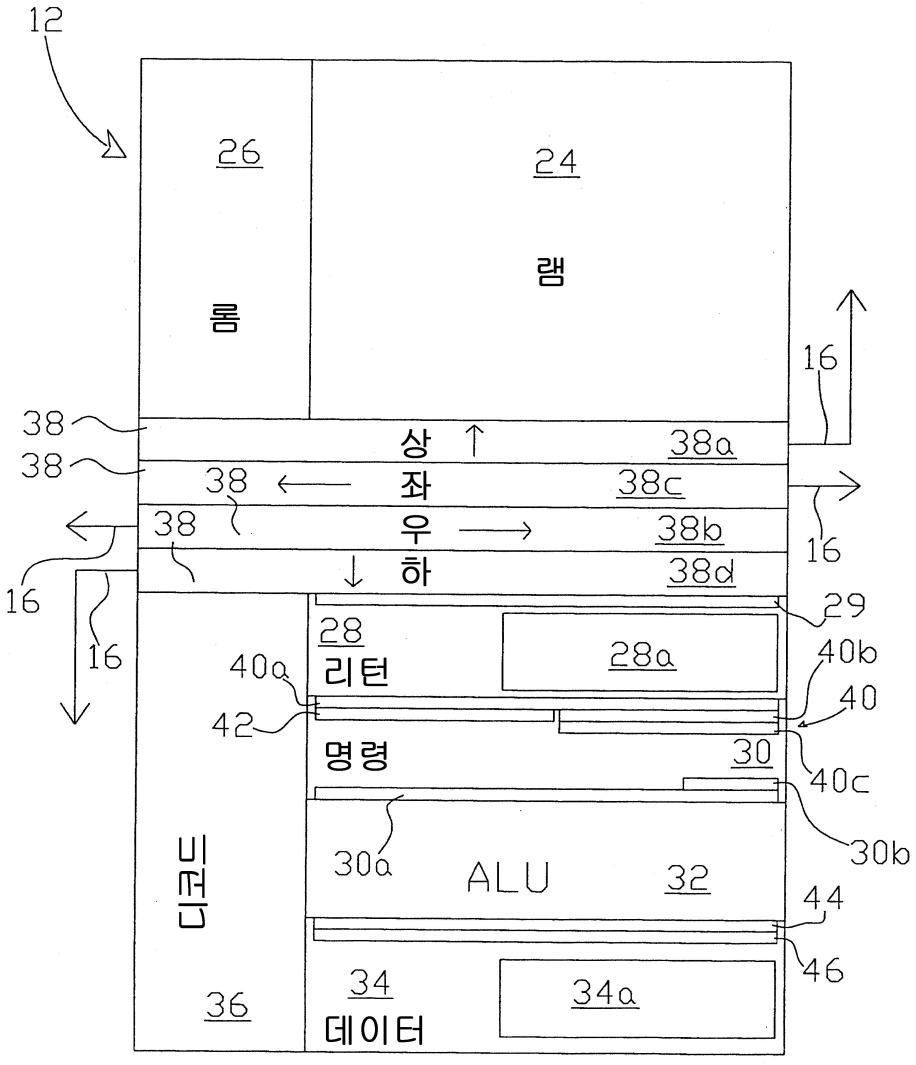
도면1



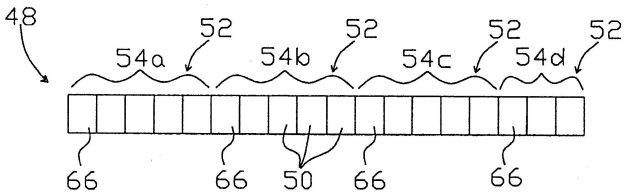
도면2


12
12a
12
10
18
20
16
18
20
12c
12e
12b
12
22
22
12
12d
12

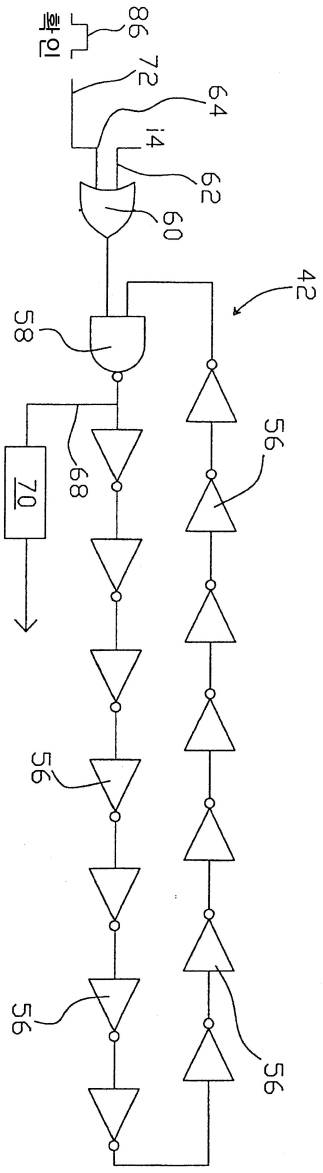
도면3


12
26
롬
24
램
16
38
상
38a
38
38
좌
38c
38
우
38b
16
16
하
38d
29
28
리턴
28a
40b
40a
40
42
명령
30
40c
30b
30a
ALU
32
CPU
44
46
34
34a
36
데이터

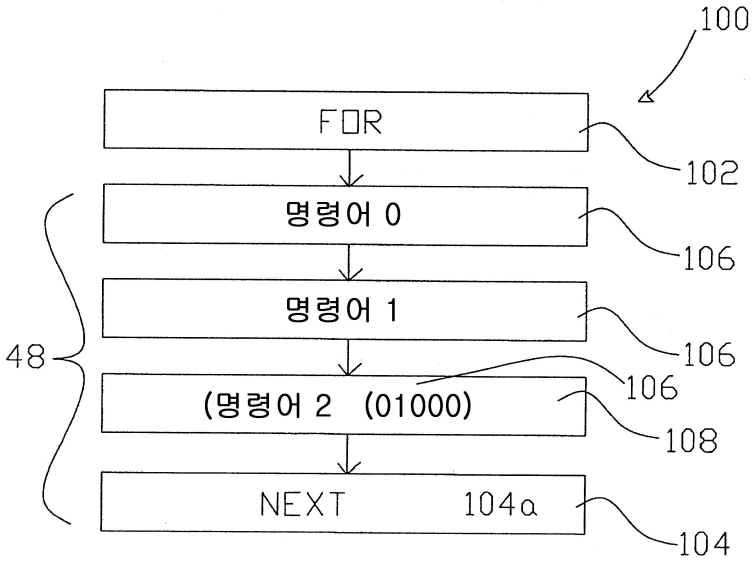
도면4


48
52
52
52
52
54a
54b
54c
54d
66
66
50
66
66

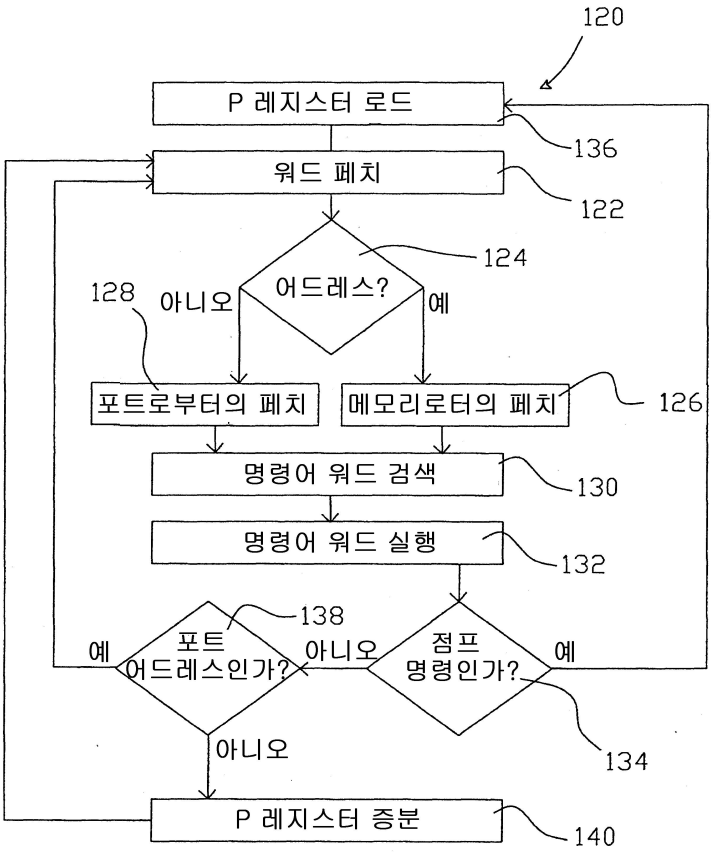
도면5


86
확인
72
64
i4
62
60
42
58
56
68
70
56
56
56

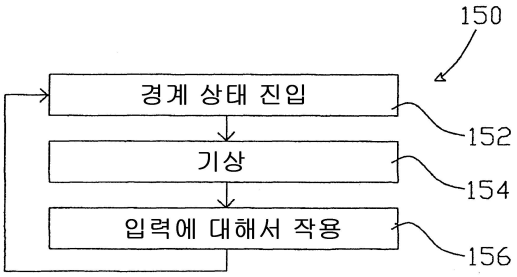
도면6


100
FOR
102
명령어 0
106
명령어 1
106
48
(명령어 2 (01000)
106
108
NEXT 104a
104

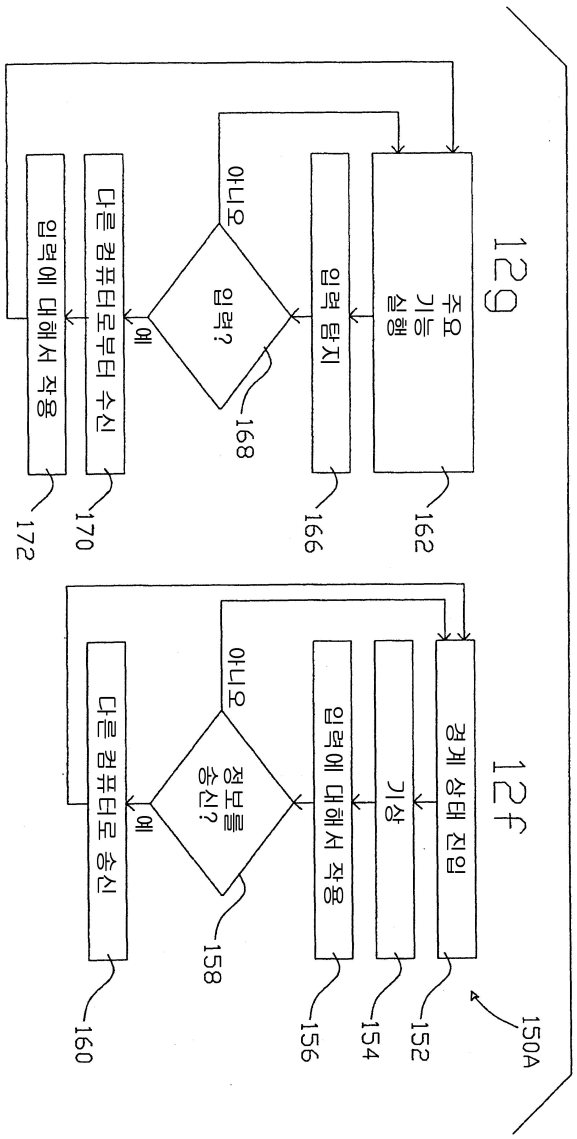
도면7


120
P 레지스터 로드
136
워드 페치
122
124
128
아니오
어드레스?
예
포트로부터의 페치
메모리로터의 페치
126
명령어 워드 검색
130
명령어 워드 실행
132
138
예
포트
어드레스인가?
아니오
점프
명령인가?
예
134
아니오
P 레지스터 증분
140

도면8


150
경계 상태 진입
152
기상
154
입력에 대해서 작용
156

도면9


129
주요 기능 실행
162
입력 탐지
166
입력?
168
아니오
예
다른 컴퓨터로부터 수신
170
입력에 대해서 작용
172
12f
경계 상태 진입
152
기상
154
입력에 대해서 작용
156
정보를 송신?
158
아니오
예
다른 컴퓨터로 송신
160
150A