(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) Int. Cl.[6]
G06F 9/38

(45) 공고일자 2001년02월01일
(11) 등록번호 10-0280161
(24) 등록일자 2000년11월08일

(21) 출원번호	10-1993-0702016	(65) 공개번호	특1993-0703642
(22) 출원일자	1993년07월05일	(43) 공개일자	1993년11월30일
번역문제출일자	1993년07월05일		
(86) 국제출원번호	PCT/US 92/09418	(87) 국제공개번호	WO 93/09492
(86) 국제출원일자	1992년11월03일	(87) 국제공개일자	1993년05월13일
(81) 지정국	EP 유럽특허 : 오스트리아 벨기에 스위스 리히텐슈타인 독일 스웨덴 프랑스 영국 이탈리아 룩셈부르크 네덜란드 국내특허 : 일본 대한민국		
(30) 우선권주장	787,676 1991년11월04일 미국(US)		

(73) 특허권자 유니시스 코포레이션 보베테 죤스
미합중국 펜실바니아 19424 블루 벨 우편사서함 500 타운쉽 라인 앤드 유니온 미팅 로드
(72) 발명자 크랙 알. 처어치
미합중국 펜실바니아 19083 해버타운 매리랜드 애비뉴 307
조세프 에스. 쉬빙어
미합중국 펜실바니아 19460 포네닉스빌리 실버 스프링스 로드 156
앤드루 티. 제닝스
미합중국 펜실바니아 19382 웨스트 체스터 엔. 디어우드 드라이브 200
(74) 대리인 강명구

심사관 : 전현진

(54) 순서를 벗어난 작업처리 방법 및 장치

요약

컴퓨터 시스템(제1도)이 의도된 순서와는 다른 순서로 결과를 제공하도록 단계들을 수행한다. 지시들이 다수 작업들로 접속된다. 컴퓨터 지시스트림내 한 변수의 다른 인보게이션이 각기 다른 저장위치에 할당되며 각 저장위치가 각기 다른 작업에 해당된다. 한 특정 작업과 관련된 모든 저장위치가 이용가능 리소스를 나타내는때(가령, 유효한 가변입력), 작업이 실행될 수 있다. 한 메카니즘이 필요하다면 인터럽터 또는 에라 때문에 작업 재-실행을 허용한다.

대표도

도1

명세서

[발명의 명칭]

순서를 벗어난 작업처리 방법 및 장치

[도면의 간단한 설명]

제1도는 중앙처리모듈(CPM)의 아키텍쳐를 도시한 블럭도표.

제2도는 주소지정결합 연합 메모리내 레지스터의 구성을 도시한 도표.

제3도는 CPM 발생 작업의 수행을 설명하는데 유용한 흐름도.

제4도는 재시작 메카니즘의 동작을 설명하는 흐름도.

제5도는 재시작 메카니즘의 동작을 설명하는데 유용한 예견된 브랜지 지시를 설명하는 흐름도.

제6도는 재시작 메카니즘의 한 특징에 대한 동작을 설명하는 흐름도.

제7도는 재시작 메카니즘의 추가의 한 특징에 대한 동작을 설명하는 흐름도.

제8도는 푸시-다운(push-down) 스택으로 배열된 자료와 코드를 포함하는 메모리의 한 부분에 대한 도면.

[발명의 상세한 설명]

본 발명은 본래 기록되고 컴파일된 것이 아닌 명령으로 컴퓨터 프로그램을 위한 목적코드를 수행하기 위

한 방법 및 장치에 관한 것이며 이같은 수행이 아치코드가 컴파일된 것과 같이 순차적으로 수행되는 것과 같은 결과를 달성하는 것에 관한 것이다.

통상, 컴퓨터는 순차적 명령으로 컴퓨터 프로그램의 목적코드를 수행하는 것으로 알려져 있다. 이같은 명령은 프로그램에 의해 기록된 지시순서에 의해 결정된다. 이와 같이 수행하도록 디자인된 컴퓨터는 본 뉴만(von Neumann) 아키텍쳐를 갖는 것으로 알려져 있다. 이같은 디자인의 단점은 처리기가 아직 이용되지 않고 있는 컴퓨터 시스템내의 몇가지 다른 리소스(resource)를 필요로 하는 지시를 수신하기 위해 아이들 상태로 있는다는 것이다.

이같은 아이들 시간을 이용하는 공지의 방법은 멀티플 프로그램은 동시에 수행하는 것이다. 이와 같이 하여, 한가지 프로그램이 리소스를 기다리는동안, 또다른 프로그램이 수행될 수 있다. 그러나, 개별 지시코드는 아직 순차적 명령으로 수행된다. 이는 그것이 리소스를 기다려야 하는때마다 프로그램의 개별적인 수행의 완성을 느리게한다. 추가적으로, 프로세서는 그것이 수행하는 프로그램이 사용되지 않는 리소스를 요구하는 때마다 처리 스위치를 수행하는데 몇가지 오버헤드(overhead)를 일으켜야 한다.

가능한한 신속히 개별 프로그램을 완성해야 하는 것은 최근에 트랜잭션 처리 시스템에 대한 요구가 증가함에 따라 중요하게 강조되고 있다. 이같은 시스템은 많은 수의 트랜잭션 완성을 중앙 집중시키기 위해 컴퓨터를 사용한다. 이들 트랜잭션은 시스템 데이타 베이스와 같은 한 공동 리소스로의 접근을 포함한다. 이같은 시스템에서 가능한한 신속하게 개별 트랜잭션을 완성하는 것이 중요하다. 이는 트랜잭션을 수행하는 프로그램이 가능한한 신속하게 수행되어야함을 의미한다.

프로그램의 지시는 서로 무관하게 순차적으로 발생되는 경향이 있는 것으로 알려져 있다. 만약 이같은 코드의 순서가 독립적으로 수행될 수 있다면, 개별 프로그램은 보다 신속하게 완성될 것이다. 이는 상기에서 언급한 바와 같은 트랜잭션 처리 환경에서 특히 유익하다.

이와 같은 환경에서, 처리기는 모든 리소스가 이용될 수 있을때 수행을 위한 작업(즉 한 그룹의 지시)를 선택할 수 있다. 이와같은 접근을 하드웨어가 보다 많은 시간동안 프로그램 수행에서 사용될 수 있도록 하며 리소스가 이용될 수 없는때 발생된 처리 스위칭 오버헤드를 피한다. 이는 전체 시스템의 효율을 개선시킨다.

몇몇 시스템은 병렬 수행을 달성하기 위해 프로그램에 대한 수정을 필요로 한다. 이같은 수정은 본질적으로 정적인 코드 세그먼트를 발생시킨다. 세그먼트화는 프로그램이 디자인된때 또는 컴파일 시간에 수행된다. 따라서, 코드의 세그먼트가 특정시간에 머신의 상태와 관련된 정보를 이용하지 않고 미리 결정된다. 이같은 정보는 같은 코드 세그먼트의 실행에 의해 택하여진 리소스 및 이전의 브랜지에 대한 이용가능성을 포함한다. 이같은 정보는 간섭없이 더욱더 완성되기 쉬운 코드 세그먼트를 발생시키도록 사용된다. 이와 같은 세그먼트의 발생은 머신의 특정상태에 대한 훨씬 덜 효율적인 상황을 발생시킨다.

한 컴퓨터 시스템이 예정된 순차로 결과를 제공하도록 된 멀티플단계를 갖는 컴퓨터 프로그램을 실행하기 위해 제공된다. 이 컴퓨터 시스템은 연쇄된 지시를 형성시키기 위한 컴퓨터 프로그램 단계들을 그룹으로 하는 한 컴포넌트를 포함한다. 또다른 시스템 컴포넌트는 연쇄된 지시들을 작업이라고 하여지는 유닛으로 결합시킨다. 또다른 컴퓨터 시스템이 이 작업들을 수행한다. 이같은 추가의 컴포넌트는 결과가 예정된 순차와는 다른 순차로 획득하여진다.

본 발명의 한 예시적 실시예가 제1도에서 도시된다. 본 발명은 4개의 유닛, 코드유닛 (CU)(110), 실행유닛 (EU)(120), 기준유닛 (RU)(130) 그리고 메모리유닛 (MU)(140)을 포함한다. 각 유닛은 각 유닛에 의해 처리하기 위한 자료가 저장될 수 있는 다수의 저항기를 포함한다. 본 발명의 예시적 실시예에서, 유닛에서의 이들 다수 레지스터의 내용은 동일하다. 레지스터는 한 유닛내의 한 특정 레지스터로 컴퓨터 지시스트림내의 가변 기준을 맵핑(mapping)하므로써 접근될 수 있다(테이블을 통하여)이때의 테이블은 주소지정 결합 연합메모리 (ACAM)(111)로 간주된다. 만약 가변기준과 일치하는 어떠한 레지스터도 없다면, CU(110), RU(130) 및 MU(140)이 레지스터 공간을 할당할 수 있으며, 메모리(또는 캐쉬)로부터 이같은 할당된 공간으로 자료를 전달하며 그리고 ACAM(111)을 갱신하여 이같은 가변 기준으로의 장래의 호출이 결국 새로이 할당된 레지스터로의 맵핑이 되게끔한다.

컴퓨터 지시는 의도된 지시와는 다른 지시로 결과를 제공하도록 그룹되며 실행된다. 이는 컬티플 레지스트를 단일 변수로 할당하므로써 성취될 수 있다. 따라서, 코드의 단일변수 섹션이 이전 섹션 이전에 수행될 수 있으며 코드의 한 이전 섹션에 중대한 한 가변크기가 코드의 나중 섹션에 중대한 한 가변크기가 조작되고 있는동안 일정하게 유지된다. 한 자료 유효비트는 작업 실행을 위한 입력으로서 사용되는 모든 레지스터가 유효한 자료를 포함하기전에 작업이 실행되지 않도록 각 레지스터와 일치한다.

작업이 실행되기전에, CU(110)은 작업실행(가령, 초기 처리기상태)과 관련된 파라미터를 저장한다. 이는 만약 필요하다면 작업이 다시 시작될 수 있도록 한다. 작업은 인터럽트, 잘못 예언된 브랜치 또는 하드웨어 오기능으로 인한 재시작을 필요로 할 수 있다.

코드유닛 (CU)(110)은 컴퓨터 지시를 한 연결 지시속으로 결합시키므로써 처리율을 최대로한다. 다음에 다수의 처리기 리소스가 이같은 지시로 할당되며, 따라서 한 작업을 형성시킨다. 다음에 작업이 실행유닛(120)으로 발행된다. 특정 작업이 할당된 레지스터에 상응하는 자료유효비트가 유효가되는때, 이 작업은 EU(120)에 의해 실행된다. 선택적으로, 상기 언급한 ACAM 갱신이 필요하다면 기준유닛 (RU)(130)에 대한 한 작업이 발생될 수 있다.

만약 한 작업이 재시작을 필요로한다면, CU(110)은 재실행을 위해 적절한 유닛으로 그 작업을 다시 발행시킬 수 있다.

본 발명의 예시적 실시예에 따른 중앙처리모듈이 제1도에 도시된다. CPM은 코드유닛 (CU)(110), 실행유닛 (EU)(120), 기준유닛 (RU)(130) 그리고 메모리유닛 (MU)(140)을 포함한다. 각 유닛의 기능은 따로 논의된다.

각 유닛(즉, CU(110), EU(120), RU(130), MU(140)은 각각 다수의 자료 레지스터를 포함한다. 본 발명의

예시적 실시예에서, 이들 다수의 레지스터 각각은 분산된 자료버퍼(DDB)(150a-d)로서 참조된 한 레지스터 파일로 조직된다. 모든 DDB의 내용은 동일하다. 또한, DDB가 갱신된때 모든 DDB는 유사하게 갱신된다. 처리기내 거의 모든 자료 아이템은 각 DDB 내 엔트리를 지시하는 레지스터 번호로 관리된다. 본 발명의 예시적 실시예에서, 64개의 레지스터 번호는 각 DDB내에서 동적으로 할당된다.

한 유효비트는 각 DDB 위치와 관계가 있다. 한 레지스터의 유효비트가 CU(110)에 의해 레지스터의 할당을 받자마자 리세트되며 한 크기가 EU(120), RU(130) 또는 MU(140) 어느 하나에 의해 DDB 위치로 기록되는때 세트된다.

[코드 유닛]

CU(110)은 MU(140)으로부터의 컴퓨터 지시의 흐름을 수신한다. 지시가 수신되는때, CU(110)은 이들을 연결시키도록 하여 세개의 주소지정 지시를 형성시키도록 한다. 다음에 CU(110)은 이같이 연결된 코드를 작업의 형태로 CPM 내에 나머지 유닛중 하나로 발생시킨다.

프로그램 코드에 대한 여러자료 크기와 기준이 블럭 또는 스택내로 지시되는 방식이 제8도에서 설명된다. 여기서 도시된 바와 같이, 제8도의 자료스택은 4개의 렉시컬(lexical) 크기 0-3에 의해 규정된 4개의 활성화 기록으로 구성된다. 활성화 기록의 각 베이스 레벨 메모리내 실질적인 주소는 렉시컬 크기 또는 베이스 주소 그리고 이들로부터의 한 오프셋으로 구성된 주소커플에 의해 규정된다.

메모리의 스택부분내 활성화 기록 각각은 한 표현스택으로 구성되며 언제 특정처리가 시작되는가를 처리하기 위해 이 표현스택으로부터 자료가 채취된다. 한 처리가 많은 서브처리를 포함하며 각각은 그 자신의 활성화기록을 갖게된다. 그러나, 메인 메모리의 일부는 스택지역의 바깥에 존재하며 하기에서 설명되는 바와 같이 자료배열을 위해 사용된다.

어떻게 블럭 구성코드가 본 발명에서 연결되는가를 설명하기 위해, 이같은 블럭 구성코드의 순서가 테이블 1에서 설명되며 이같은 코드는 포트란 루프의 실행을 위한 것이다.

테이블 1

		작업	트리플릿
L1:	VALC I		0
	NAMC A	JOB 0	
	INDX		
	DUPL		1
	LOAD	JOB 1	2
	VALC I		
	NAMC B	JOB 2	3
	NXLV		
	VALC Z		
	MULT)		4
	) PAIR	JOB 3	
	)		
	ADD		
	STOD		5
	VALC I		
	ONE		6
	ADD)		
	) PAIR	JOB 4	
	NAMC I)		
	STON		7
	VALC N		
	LSEQ	JOB 5	8
	BRTR L1		
LO:			

이 코드는 프로그램 세그먼트를 위한 것이다 :

```
      DO 100 I = 1, N
100   A(I)=A(I) + B(I) * Z
```

테이블 1의 좌측 컬럼에서 나타낸 바와 같이 코드 스트링의 첫번째 지시는 자료 아이템 I의 크기호출 또는 메모리 채취, 지시가 스택의 상부에서 자료 설명기 기준을 제공하는 크기 A의 이름호출이며, 인덱스 지스가 메모리내에 있는 배열요소 A(I)에 대한 포인터(pointer)의 발생을 일으킨다. 지시 LOAD는 메모리 채취 명령으로서, 이는 인덱스된 설명기에 의해 주소지정된 자료크기를 획득한다. 지시쌍 VALC I, NAMC B는 변수 I 의 크기와 메모리로부터의 자료배열에 설명기를 채취한다. 지시 NXLV는 인덱스 동작을 발생시키며 또한 배열셀 B(I)내에서 자료 채취를 발생시킨다. 지시 VALC Z, MULT 및 ADD는 계산 A(I)+B(I)*Z 를 수행하며 그결과를 스택의 상부에 남긴다. STOD는 A(I)에 의해 주소지정된 메모리로의 자료 아이템을 저장하며 STON 이 다음사용을 위해 스택에 저장하고 이를 떠나는동안 그 스택 입력을 삭제한다.

테이블 I에서 설명된 바와 같이, 코드는 서브시퀀스(subsequences)로 잘려지는데, 이때 서브시퀀스 각각은 본 발명의 처리기에 의해 병렬 수행을 위해 연결될 수 있는 세개의 오퍼레이터(트리플릿)까지의 시퀀스로 형성된다. 따라서 코드의 전체순서가 제1도 처리기의 각 파이프라인으로 입력된 소위 6개의 작업으로 확인된 9개의 트리플릿으로 나뉘어진다. 작업3과 작업4 각각은 순서적으로 단일의 작업으로 발행될 수 있는 한쌍의 트리플릿으로 형성된다. 트리플릿은 본 발명에 32MHz 인 프로세서의 정규 클럭 주파수 두배

의 다른 기능 유닛으로 발행된다. 따라서, 트리플릿은 64MHz의 속도로 발생된다.

제1도 코드유닛(110)의 특정특징은 기능유닛 각각이 제공된 다양한 자료버퍼(150a-d)에서 버퍼 레지스터를 할당할 수 있는 방식이다. 따라서, 테이블 1로의 리터닝(복귀)는 테이블 1의 특정 포트란 루프에서 총 12개의 메모리 기준일 수 있으며, 이들 기준들중 많은 것이 메인 메모리 저장 위치 대신에 각각 할당, 분산된 자료버퍼 레지스터로 변환될 수 있다. 테이블 1의 루우프를 통과하는 각 통로에서, 메인 메모리저장 위치에 대하여 최소 4개의 기준이 수행된다. 이들 기준은 메모리 채취 지시 LOAD 및 NXLV이며 두개의 메모리 저장동작이 STOD와 STON이다. 이들 4개의 지시를 제외하고는 제1도의 여러 기능 유닛이 이들의 분산된 자료버퍼(150a-d)로부터 동작하며, 이는 상기에서 지적한 바와 같이 각 자료버퍼가 항상 다른 자료버퍼(150a-d) 각각과 같은 내용을 갖도록 구성된다.

테이블 1에서 다양한 자료 아이템 I, A, B, Z 및 N 는 메인 메모리내에 저장된 크기를 주소지정하기 위해 스택내의 활성화 기록을 나타내는 실제 주소커플이다. 코드유닛(110)에서는 테이블 1의 포트란 루우프 실행중에 발생되는 각 주소커플로 각각의 자료 버퍼내 다양한 레지스터를 할당시키는 준비가 된다.

테이블 1에서 대부분의 경우 각 작업은 오퍼레이터의 트리플릿으로 형성된다. 코드연결은 멀티스테이지 처리이다. 첫번재 단계, 스캔레벨은 오퍼레이터 트리플릿을 형성한다. 두번째 단계, 연결레벨은 트리플릿이 한 엔티티로 발생될 수 있는가 아니면, 더블릿(doublet) 또는 싱글릿(singlet)으로 이루어진 한 시퀀스로 잘려져야 하는 가를 결정한다. 마지막 단계, 작업레벨은 연속적인 트리플릿이 “슈퍼” 작업으로 쌍으로 이루어져서 단일 작업번호가 그같은 쌍으로 할당될 수 있도록 하는가를 결정한다. 한 작업내의 다양한 지시는 처리기의 각기 다른 컴포넌트에 의해 병렬로 실행될 수 있다.

스캐너는 코드 스트림을 파싱(parsing)하므로써 오퍼레이터 트리플릿을 형성시킨다. 스캔닝 목적을 위하여 오퍼레이터는 다음의 카테고리내로 분류되어진다:

a) ROP - 기준 오퍼레이터는 VALC, NAMC, ZERO, ONE, LT8, LT16이다.

b) TOP - 단자 오퍼레이터는 STOD, STON, BRFL, BRTR, BRUN 이다.

c) MOP - 메이저 오퍼레이터는 모든 다른 오퍼레이터이다.

ROP 오퍼레이터는 하나와 세개의 음절 사이를 점유한다. TOP 오퍼레이터는 각 하나의 음절 오퍼레이터이며 MOP 오퍼레이터는 하나의 음절 오퍼레이터이다.

스캐너는 시스템 클럭 속도(패스트 클럭)에서 두번 트리플릿을 형성시킨다. 다음 규칙은 트리플릿 형성을 결정한다:

a) 패스트 클럭에서 트리플릿 마다 6개 음절 최대.

b) 시스템 클럭에서 두개의 트리플릿마다 8개 음절최대.

c) 가능한 트리플릿은

1) ROP ROP MOP (또는 ROP ROP 또는 ROP MOP 또는 MOP)

2) ROP ROP TOP (또는 ROP TOP 또는 TOP)

3) ROP MOP TOP (또는 MOP TOP).

LT8과 LT16은 8비트 및 16비트 리터럴 크기 지시이며, BRFL은 만약 조건이 거짓이면 스택과 브랜치의 상부에서 조건을 검사한다. BRTR은 만약 조건이 사실이면 브랜치를 발생시키고, 그리고 BRUN은 조건없는 브랜치이다.

상기에서 설명된 바와 같이 본 발명의 스캐닝과 연결은 두개의 레벨로 행하여지며 코드 스캔닝은 6개 음절 각각의 두 프로그램 단어들을 스캔할 수 있고 각각으로부터 상기 지적된 규칙에 따라 연결시키기 위한 8개 음절을 선택할 수 있다. 테이블 1의 대부분 오퍼레이터는 VALC I 과 같은 다수의 보다 복잡한 오퍼레이터가 음절 오퍼레이터이라하더라도 단지 하나의 8-비트 음절이다.

인셉션(inception)으로부터 완성까지 작업이 진행되는 것을 추적하기 위해, 각 작업은 작업번호를 할당받는다. 작업번호는 순차적으로 작업이 CU(110)에 의해 발생되는때 작업으로 할당된다. 한 작업번호는 CU(110)에 의해 할당을 받지 않게될때까지 다시 사용되지 않는다.

CU(110)은 처리기상태 목록(PSL)을 유지시킨다. PSL 은 부울대수 플레그를 기록하도록 사용된 다수의 레지스터 번호, 그리고 실행되고 있거나 실행준비중인 작업과 관련된 주소크기를 포함 한다.

CU(110)은 또한 한 작업이 부적절한 실행때문에 조급하게 종료된 경우에 사용된 재시작 RAM 을 유지시킨다. PSL 은 재시작 RAM 내에서 유지된다. 재시작 RAM은 작업번호를 사용하여 주소가 지정된다. 재시작 RAM 의 동작은 제4도와 관련하여 하기에서 설명된다.

CPM 이 클럭마다 한 작업을 완성하도록 하기위해, 코드의 자료 참조 요구사항을 줄이는 것이 유용하다. 이는 주소지정 연결 연합메모리(ACAM)(111)의 사용으로 달성된다. 프로그램 지시는 가변 이름에 해당하는 코드흐름내의 기준을 포함한다. 본 발명의 예시적 실시예에서, 이들 기준은 주소지정-결합의 형태를 한다. ACAM(111)은 완전히 작은 연합적 저장-통과 캐쉬를 형성시키는 CU(110)내의 한 메모리이다. 본 발명의 예시적 실시예에서, ACAM 엔트리는 제2도에서 도시된 정보를 갖는다. 도시된 바와 같이, ACAM(111)은 주소지정 연결을 위한 한 필드(field)(이는 프로그램 지시로부터 획득된다) 그리고 레지스터 번호를 위한 필드(이는 레지스터의 콜렉션과 일치하며 이 컬렉션에서 변수로 할당된 자료크기가 저장된다) 선택적으로, 제1도의 가장 우측 ACAM 필드는 DDB 내 레지스터 번호와 일치한다. ACAM(111)의 동작은 DDB 와 결합하여 설명되는데, 이는 레지스터의 어떠한 콜렉션도 사용될 수 있기도한 것이다.

ACAM(111)은 주소지정 연결을 DDB 레지스터 번호와 연합시키므로써 동작한다. 주소지정 결합이 실행작업의 코드 스트림에 나타나는때, 이는 ACAM(111)내 주소지정 연결과 비교되며 자료가 이미 DDB 레지스터내

에 있는가를 결정하도록 한다. 만약 주소지정 결합이 발견되면, 관련된 DDB 레지스터번호가 입력크기로 제공되며, 따라서 주조지정 평가 동작과 메모리로부터의 채취를 없앤다. 만약 주소지정 결합이 ACAM(111)내에서 발견되지 않으면, 대체가능 ACAM 위치가 선택되며 DDB 내 레지스터가 할당된다. 주소지정 연결이 주소지정 연결의 표적에 해당하는 저항기 번호와 함께 ACAM 엔트리내로 입력된다. 다음에 프로그램 지시에 의해 참조된 자료가 메모리로부터 채취되고 할당된 DDB 레지스터내로 삽입된다.

DDB 내 이용가능 레지스터를 추적하기 위해, 사용된 레지스터 번호 테이블이 유지된다. 이 테이블은 작업에 의해 사용되고 있는 레지스터를 추적하며(이들과 관련된 레지스터 번호를 통하여) 새로운 작업을 위한 자유 레지스터를 제공한다. 만약 DDB내 모든 레지스터가 할당된다면, CU(110)은 DDB 레지스터의 예정된 수가 할당을 받지 않게될때까지 새로운 작업을 실행할 것을 정지한다. DDB 레지스터는 그와 같은 DDB 레지스터에 대한 모든 참조(가령 ACAM(111)로부터)가 삭제되어진때에 할당에서 벗어나게된다.

ACAM 엔트리를 통해 앞서 할당된 DDB 레지스터내로 한 크기를 저장하는때는 새로운 DDB 레지스터가 이같은 크기를 수신하도록 할당된다. 따라서 프로그램 변수의 멀티플 카피가 유지된다. 이는 뉴네이밍(newnaming)으로 참조된다. 프로그램 변수의 멀티플 카피를 유지시키므로써, 프로그램 코드의 실행순서가 결과의 정확성에 영향을 주지않고 변경될 수 있다. 이는 하기에서 상세히 설명된다.

[참조 유닛]

RU(130)의 일차적 목적은 CU(110)의 요규에 따라 메모리 참조를 발생시키는 것이다. RU(130)은 메모리 참조의 발생을 요구하는 CU(110)으로부터 작업을 수신하므로써 이를 달성시킨다. 이들 작업은 RU(130)내 주소지정 연결(AC) 작업 대기행렬내에 저장될 수 있다. RU(130)은 입력 레지스터 번호가 오래된 기록작업에 의해 부과된 오더링(ordering) 제한조건에 따라 유효하게되는때 작업을 소유한다. 이같은 작업의 결과가 MU(140)로 보내지며 이때 캐쉬/메모리 접근이 수행될 수 있고 각 DDB(150a-d)내로 적재된다. RU(130)에 의한 더이상의 처리가 요구된다면, RU(130)으로의 끝나지 않은 Op (UFO) 대기행렬 입력이 활성화된다.

[실행유닛]

EU(120)은 CU(110)으로부터 수신된 자료 기능작업을 소유한다.

자료기능 작업의 예는 산술적 연산, 스트링 연산, 필드연산 그리고 논리연산을 포함한다. 이들 작업은 두 작업 대기행렬-S 작업 대기행렬(자료가 EU(120)에 대한 지역인 개연성이 높은 작업) 그리고 V 작업대기행렬(그밖의 모든작업)중 한 작업에 저장된다. 한번에 단 하나의 작업이 발행될 수 있다. 발행된 작업은 두 실행 유닛 - 간단한 연산을 위한 패스트 산술 엘리먼트(FAE) 또는 보다 복잡한 연산을 위한 범용 엘리먼트(APE)중 한 유닛에서 실행된다. 한가지 작업은 또한 자료배열 연산을 수행하기 위해 자료배열 처리기로 발행될 수 있다.

[메모리유닛]

MU(140)은 캐쉬/메모리 자료가 CPM 내 남아있는 유닛에 이용될 수 있도록 한다. 또한 MU(140)은 메모리 시스템과 CPM 내 남아잇는 유닛 사이의 인터페이스이다. CU(110)과의 MU의 인터페이스는 MU 가 CU 코드 요구와 메모리 코드 캐쉬 소거에 응답하도록 허용한다. MU(140)은 또한 RU(130)을 통한 파이프 라인의 연장과 DDB로의 자료캐쉬(112)로 작용하기도 한다.

[작업실행]

본 발명의 예시적 실시예의 동작이 제3도의 흐름도에 의해 설명된다.

단계(320)에서, CU(110)은 코드 스트림을 수신하며 다수의 작업을 구성시킨다. 이는 코드연결부분과 작업처리부분에 의해 달성되며, 바람직하게는 각각이 CU(110)내에 놓인다. 코드연결 부분은 코드 스트림을 수신하며 지시들을 다수의 연결된 지시속으로 그룹시킨다. 이같은 동작이 상기에서 설명된다. 작업처리부분은 연결된 지시에 의해 요구된 어떠한 리소스도 할당하며, 따라서 다수의 작업을 형성시킨다. 한가지 작업이 CU(110)에 의해 형성된 각 작업과 관련된다.

단계(325)에서, CU(110)은 각 작업내 프로그램 변수를 평가하며, 만약 단계(330)에서 평가된 프로그램 변수의 주소지정 연결이 ACAM내에 있는가를 결정하도록 한다. 만약 주소지정 연결이 ACAM내에 있으면, 단계(335)에서, CU 는 ACAM 으로부터 상응하는 DDB 레지스터 번호를 획득하며 레지스터 번호를 갖는 EU(130)으로 작업을 발생시킨다. 그렇지 않으면, 만약 주소지정 연결이 ACAM 내에 있지 않으면 단계(340)에서 CU(110)은 레지스터 번호를 위해 ACAM 내에 한 슬롯을 할당시키며 관련된 레지스터 번호를 갖는 한 주소지정 연결을 ACAM 내에 위치시킨다.

단계(345)에서 CU(110)은 작업으로부터 RU(120)으로 주소지정 결합을 보내며 메모리로부터 자료크기를 채취하도록 한다.

단계(350)에서, RU(120)은 MU(140)을 위한 메모리주소를 발생시킨다. 이같은 주소는 요구된 자료를 채취하도록 사용된다.

단계(355)에서 MU(140)은 요구된 자료를 채취하며 (필요하다면 캐쉬 혹은 메인 메모리로부터) 이와 같이 채취된 자료를 적절한 DDB 레지스터내에 위치시키며 적절한 자료 유효비트를 세트시킨다.

특정 작업과 관련이 있는 모든 자료 유효비트가 세트된때 단계(360)에서 EU(110)내의 작업이 실행될 수 있다.

[순서를 벗어난 작업실행]

분산된 자료 버퍼에서 리네임잉 기능을 사용하여, 순서를 벗어난 작업실행이 달성될 수 있다. 이는 테이블 2a 와 2b의 설명적 예시를 사용하여 설명된다.

테이블 2a

```
10  X : = a + b
20  Y : = f(x)
30  X : = c + d
40  Y : = f(x)
```

테이블 2b

```
10  X o : = a + b
20  Y   : = f(x o)
30  X n : = c + d
40  Y   : = f(x n)
```

테이블 2a, 단계(10)에서 프로그램 변수 a 및 b 는 함께 가산되며, 그 합계가 프로그램 변수 x 에 해당하는 레지스터내에 저장된다. 테이블 2a, 단계(30)에서 프로그램 변수 c 와 d 가 역시 함께 가산되며, 그 합계가 x 레지스터내에 저장된다. 따라서 단계(30)은 단계(10)에서 저장된 x 의 크기를 중복기재할 수 있다. 테이블 2a, 단계(40)에서 몇가지 기능이 단계(30)에서 결정되었던 x 의 크기에 적용되며 이같은 기능을 적용시킨 결과가 y 에서 저장된다.

단계(40)이 단계(30)에서 결정되었던 x 의 크기를 사용하여 수행되기 때문에 테이블 2a 에서 도시된 단계의 시퀀스는 순차적으로 수행되는 것이 바람직하다. 만약 테이블 2a 에서 도시된 단계가 비순차적 방식(가령, 먼저 단계(30), 다음에 단계(10) 그리고 마지막으로 단계(40))으로 수행된다면 단계(40)의 기능은 단계(30)으로부터 계산된 x 의 크기 대신에 입력으로 단계(10)으로부터 계산된 x 를 수신할 것이다. 만약 이같은 상황이 발생된다면, 단계(40)에서 계산된 y 의 크기는 바르지 않을 것이다.

그러나 한 그룹의 컴퓨터 지시를 순차적으로 수행하는 것이 항상 바람직하지는 않다. 계산성 효율은 비-순차적으로 한 그룹의 컴퓨터 지시를 수행하므로써 개선될 수 있다. 비-순차적으로 지시를 수행하므로써 리소스 사용의 최적화가 실현될 수 있다. 그러나 상기에서 도시된 바와 같이, 비-일련적으로 실행된 프로그램은 착오의 결과를 발생시키지 않도록 주의하여야 한다.

본 발명은 리네임잉 기능을 사용하는 각 접근된 변수를 위해 필요한대로 멀티플 저장위치를 제공하므로써 이들 단점들을 극복한다. 이는 테이블 2b 에서 설명된다. 테이블 2b 에서 도시된 프로그램은 테이블 2b 에서 도시된 컴퓨터 프로그램과 거의 동일하다. 그러나, 테이블 2b 에서 라인(10)내 변수 x 의 접근과 라인(30)내 변수 x 의 접근을 위해 분리된 저장위치가 제공된다. 이 리네이밍 기능을 사용하므로써 달성된다. 명료함을 위해 테이블 2a 내 x 는 테이블 2b 에서 x_o 와 x_n 로 대체되었다. 이와 같이 하여, 테이블 2b 에서는 지시(30)(40)이 지시(10)(20)가 실행되는 시간과는 무관하게 실행될 수 있다. 따라서 컴퓨터 프로그램내 각기 다른 포인트에서 x 의 크기가 유지되는 사실로 인해 지시(10)(20)가 실행되기전에 지시(30)(40)이 실행될 수 있다.

각 변수로의 신속한 접근을 허용하기 위해 (그리고 각 변수의 각 복사를 허용하기 위해) DDB 내에서 각 변수(및 변수복사)를 위한 공간이 할당되었다. 공간이 더이상 가변크기를 보유할 필요가 없을때 공간이 DDB 로부터 할당을 받지않게될 수도 있다. 유효비트는 DDB 내 각 할당된 공간과 관계가 있다. 이같은 유효비트는 한 가변크기가 적절한 저장위치내에 저장된때를 나타내며, 이 가변크기가 뒤이은 지시에 의해 접근되는때를 나타낸다.

다시 테이블 2b 에서 x_o 는 DDB 의 레지스터 1 (R1)에서 공간을 할당받을 수 있으며 x_n 는 DDB 의 레지스터 2(R2)에서 공간을 할당 받을 수 있다. a 와 b 의 합계가 R1 내에 저장된때, R1 으로 할당된 유효비트가 세트된다. 유사하게, c 와 d 가 R2 내에 저장된때, R2 로 할당된 유효비트가 세트된다. 특정변수로 할당된 유효비트가 세트된때, 그같은 변수는 더이상의 계산을 위해 사용될 수 있다. 이와 같이 하여 컴퓨터 코드의 특정순서 실행순서가 변경될 수 있다.

본 발명의 설명적 실시예에서, 기준유닛(RU)(130)과 실행유닛(EU)(120) 각각은 각 유닛에서의 실행을 위해 계획된 작업의 대기행렬을 담고있다. 앞서 언급한 바와 같이, 작업은 코드유닛(CU)(110)에 의해 이들 대기행렬 각각으로 입력된다.

따라서 실행을 위해 가장긴 시간을 대기하였던 작업은 이들 대기행렬의 전방에 위치하게 된다. 실행을 위해 가장 짧은 시간 대기하였던 작업은 이들 대기행렬의 뒤에 있게된다.

기준유닛(130)과 실행유닛(120) 모두는 이들 각각의 대기행렬에서 작업을 시험할 수 있으며, DDB 와 관련된 유효비트로부터 어느 작업이 실행될 수 있는가를 결정하도록 한다. 따라서 몇가지 작업은 다른 작업이 실행을 위해 준비되있지 않은동안 실행을 위해 준비되게 된다. 일련의 머신은 그것이 이들 대기행렬들중 어느 한 대기행렬에서 작업을 수행할 수 있는 순서로 제한된다. 본 발명에서는 모든 입력 변수가 결정되자마자 작업이 실행될 수 있기 때문에, 대기행렬내 어떠한 순서로라도 작업을 실행하는 것이 가능하다. 이와 같이하여 기준(참조)유닛(130)과 기준유닛(120)을 사용하는 것이 일련의 작업 실행과 비교하여 증가

된다.

비록 유효자료가 레지스터내 자료의 유효를 할당받는다해도, 한 유효자료비트가 역시 시스템 리소스(가령, 시스템테스트, 하드웨어장치등)의 이용가능성을 나타낼 수도 있다. 따라서, 자료 유효비트는 작업이 실행되기전에 리소스 이용가능성을 결정하기 위해 시험될 수 있다.

작업으로의 입력이 유효함을 신호함과 더불어, 레지스터(이들의 관련된 레지스터번호(RNs)를 통하여)가 역시 작업의 완성을 나타내도록 사용될 수 있다. CU(110)에 의해 발생된 각 작업은 그 끝이 RN 으로 지정된 한 RN 을 갖는다. 이 끝 RN 은 신호출력을 갖는 작업, 멀티플 출력 RNs 을 갖는 작업을 위한 출력 RN 이며, 이 출력 RNs 중 하나가 끝 RN 으로 지정된다. 끝 RN 과 관련된 유효비트가 세트된때, 끝 RN 과 관련된 작업이 완성된다. CU(110)이 언제 작업이 완성되었는가를 알아서 다양한 리소스(즉, 레지스터)가 할당을 받지 않을 수 있도록 함이 바람직하다.

[재시작]

순서를 벗어난 작업실행에 추가하여, 본 발명은 오차 및 인터럽트와 같은 여러가지 이유로 시스템이 재시작되도록 하는 한 메카니즘을 포함한다. 재시작을 개시하기 위한 예시적 이유는 잘못된 조건부 브랜치 예견, 프로세서 내부에라, 및 환경적 고장 (즉, 디스크헤드 파괴, 메모리 패리티 에라등)등이 있다.

그 에라(error) 처리능력 때문에 재시작 메카니즘이 회복 메카니즘으로 특징될 수도 있다. 재시작 메카니즘이 어떤한 종류의 컴퓨터 처리환경에서도 사용될 수 있으나 이같은 메카니즘이 이같은 환경에 특히 적합하다. 특히, 이같은 메카니즘은 컴퓨터 프로그램 결과가 이들의 의도된 순서와는 다른 순서로 획득되는 환경에 적합하다.

하기에서 설명된 바와 같이, 재시작 메카니즘은 CU(110)에서 실현된다. 이와 같은 메카니즘으로부터 이용가능한 회복 능력을 제공하기 위해, CU(110)은 회복 RAM(RC)(113)을 유지한다. 제1도에서 RC(113)은 다수의 기록을 유지시키며 각 기록이 특정작업(따라서 특정 작업번호)로 할당된다. 각 기록은 다음과 같은 특정작업과 관계가 있는 다수의 정보를 유지시킨다.:

1) 현재의 코드 포인터 - 이는 작업을 위한 코드 스트림내 첫번째 오퍼레이터를 지적한다.

2) 양자택일의 코드 포인터 - 이는 브랜치 지정을 지적한다. 선택에 따라, 이는 예정된 브랜치 지정을 나타내기도 한다.

3) 엔드 레지스터 번호 - 이는 작업이 상기에서 설명한 바와 같이 언제완성에 도달하는가를 결정하기 위해 사용된다.

4) 처리기 상태 리스트 - 상기에서 설명한 바와 같이, 이는 적절한 재시작을 위해 필요할 수 있는 특정작업을 위한 특정 상태 정보를 포함한다. 이 정보는 DDB 레지스터내에 유지될 수 있는 상태 아이템에 대한 하나의 레지스터 번호를 포함한다. 따라서 이들 레지스터들은 이들이 잠정적 재시작에 더이상 필요하지 않을때까지 다른 작업에 사용될 수 없다.

5) 인터럽트 오프셋 - 이들은 인터럽트 벡터를 발생시키기 위해 사용될 수 있다.

각 기록이 작업 재시작을 용이하게 위한 추가의 정보를 포함하도록 하는 것이 필요하다. 이같은 정보는 컴퓨터 시스템 실현의 개별사항에 의존하는 것이다. 이같은 세부사항들은 어떠한 스택이나 대기행렬의 복사(또는 부분적인 복사), 작업을 실행하기 위한 종래 시도와 관련된 이력정보 그리고 머신동작의 특정모드등을 포함한다.

작업정보로 참조되어진 상기 정보는 CU(110)에 의해 RC(113)내 측정작업을 위해 저장된다. 이와 같은 저장은 특정작업 정보와 관련된 작업의 발생이 있게되면 발생된다.

재시작 메카니즘의 동작이 제4도에서 설명된다. 재시작을 위한 필요가 EU(120)내 작업 제어논리(도시되지 않음)에 의해 결정될 수 있다. 특히 EU(120)내 작업 제어논리를 활동적인 작업진행을 추적한다. 작업제어 논리는 인터럽트 조건들이 탐지되는때(단계510) 컴퓨터 시스템내의 다른 처리 엘리먼트로부터 인터럽트 리포트(존재하는 때)를 수신한다. 다음에 EU(120)이 각 인터럽트 리포트를 포맷하며 각 인터럽트를 CU(110)(스텝 515)로 보낸다.

인터럽트 리포트를 수신하자마자, CU(110)은 수신된 인터럽트중에서 가장 오래되고 가장 우선인 인터럽트를 선택하며 이같은 인터럽트를 현재의 재시작 작업번호(RJN) 레지스터(단계 520)내로 일어나게 하였던 작업의 작업번호를 적재한다. 만약 하나이상의 인터럽트가 같은 작업번호와 동시에 도달하였다면, CU(110)은 임의로 작업을 일어나게하는 하나의 인터럽트를 선택한다. 진행되어질 다음작업에 해당하는 작업번호는 스킵(skip)작업번호(SJN) 레지스터내로 적재된다(단계 525).

RJN 을 사용하여 CU(110)은 모든 남아있는 유닛으로 재시작 지시를 발생시킨다. 이 재시작 지시는 RJN 을 즉 재시작을 필요로하는 작업에 상응하는 작업번호를 포함한다. 나머지 유닛은 RJN 보다 오래된 모든 작업을 수행하면서 계속해서 동작한다. 한편, CU(110)은 재시작 메카니즘을 필요로 했던 문제를 분석하게된다. 머신내 가장 오래된 작업번호는 재시작 작업번호(RJN)과 같거나 더욱크며, 그러면 인터럽트 이전의 모든 작업은 완성된다. 이같은 조건이 갖추어진때, 모든 유닛은 재시작하려는 작업보다 크거나 같은 번호를 갖는 모든 작업(즉 RJN 후의 모든작업)을 버리며, RJN 후 모든 작업에 의해 획득된 결과를 버리고, 개별작업을 처리하기 위해 사용된다(단계 535).

CU(110)은 또한 RJN 에 상응하는 작업을 재시작하기전에 처리기를 그 상태로 세트시킨다. 또한 스킵 작업번호(SJN)는 OPJ 레지스터내로 가장 오래된 펜딩작업(OPJ)로 적재된다.

인터럽트 조건을 처리하기 위해 어떤 교정 측정을 택할 것인가를 결정함에 따라, CU(110)은 인터럽트를 처리하기 위해 적절한 작업을 발생시키기 시작한다.

이들은 여러 예를 통하여 설명된다.

첫번째 예에서, 제5도에서 설명된 바와 같이 인터럽트는 바르지 않은 브랜치 선택에 의해 발생된다. 경로 A 를 따른 바르지 않은 브랜치 선택은 다음 바르지 않은 예견된 브랜치로부터 발생될 수 있다. 따라서 이 바르지 않은 브랜치 선택과 관련된 재시작 범위내 작업(작업 D 및 작업 E)는 바르지 않은 브랜치 선택이 다시 실행되지 않아야 하기 때문에 버려질 것이다. 대신에 CU(110)은 정해진 처리의(작업 C)의 선택적 브랜치를 실행하기 위한 발행코드를 즉시 시작한다. 바르지 않은 브랜치 인터럽트를 수신한 처리에 관련되지 않은 재시작 범위내 작업은 다시 시작된다.

두번째 예에서, 인터럽트가 무효의 목표를 갖는 프로그램 변수와 같은 나쁜 엔트리 에라에 의해 발생된다. 이들 프로그램 변수중 하나가 코드순서에서, 이같은 코드순서를 한가지 작업으로 연결시키는 것과 이같은 작업을 실행하는 것은 오차를 일으키게될 것이다. 뒤이은 인터럽트를 따라, CU(110)은 유닛 각각으로 재시작 명령을 발생시킨다. 모든 유닛이 RJN 보다 오래된 모든 작업의 실행을 완성한때 CU(110)은 분리된 작업으로서(단일의 작업이 아닌) 세개의 언급된 코드 각각을 발행시킨다. 에라가 다시 발생했을때(프로그램 변수가 아직 무효인 목표를 갖고 있기 때문에), CU(110)은 에라를 발생시킨 프로그램 변수를 분리시킬 수 있다. CU(110)은 특정 타입의 에라에 대한 적절한 인터럽트 처리 루틴을 불러낼 수 있다.

또한 에라를 포함하였던 작업과 관계없는 RJN 이후의 모든 작업이 버려진다. 비록 이들 작업이 에라 인터럽트이전 이들의 실행을 완성하였다하여도, RJN 이후의 작업으로부터 얻어진 어떠한 결과도 삭제된다. 이는 이같은 작업으로부터 얻어진 결과의 정확성을 결정하는 것이 비효율적이기 때문에 바람직한 것이다. 세번째 예에서, 재시작을 야기시키었던 동작은 인터럽트의 포인트(동작의 인셉션 포인트에 반대된)로부터 재시작된다. 이같은 타입의 상황에 대한 특정예는 수천개의 배열요소로 구성된 한 배열을 갱신하는 처리중에 발생된다. 이 갱신처리내 몇개의 포인트에서, 배열의 한 부분은 메모리내에 존재하지 않는다. 이와 같은 경우에, 동작시스템이 디스크 드라이브와 같은 대용량 저장장치로부터 메모리로 자료를 가져올 수 있도록 인터럽트가 발생된다.

따라서, 이같은 세번째 예는 재시작된 작업에 의해 완성된 일이 버려진다는 점에서 첫번째 및 두번째 예와는 다르다. 이 세번째 예에서, 인터럽트 처리과정중에, 완성된 배열갱신의 부분이 MU(140)에 의해 확인된다. 나머지부분의 배열이 메모리내로 가져가진 뒤에 인터럽트 처리과정이 CU(110)으로 신호를 보내어 앞서 완성된 동작이 남아 있었던 곳으로부터 배열 갱신을 계속하는 특정된 작업(배열 갱신작업)을 발생시키도록 한다. 만약 이와 같은 방법이 사용되지 않으면, 전체 배열이 그 처리기에 속하는 메모리내에 동시에 속하지 않으면 완성시키기가 곤란하다.

재시작 메카니즘은 RJN 으로의 재시작이 활동적인 동안 인터럽트의 발생을 처리하기 위한 예정된 한 프로토콜을 포함한다. 이는 제6도에서 설명된다. 여기에는 다음과 같은 세경우가 있다:

1) 만약 RJN 보다 오래된 한 작업이 한 인터럽트를 발생시키면(단계 715), 현재의 재시작은 포기되며 새로운 재시작이 개시된다(단계 720).

2) 만약 RJN 과 SJN 사이의 작업이 인터럽트를 발생시키면(단계 725), 그 인터럽트는 무시되는데, 이는 인터럽트를 발생시킨 작업이 버려진 처리내에 있기 때문이다(단계 730).

3) 만약 SJN 과 같거나 혹은 오래되지 않은 작업이 인터럽트를 발생시키면(단계 735), 이에 상응하는 작업번호는 저장 재시작 작업번호(SRJN) 레지스터내로 적재된다(단계 740). SRJN 은 RJN 으로의 재시작이 완성될 때까지 CU(110)에 의해 보유된다.

이때에 SRJN이 처리된다.

RJN으로의 재시작이 활동적이고 펜딩(pending)인 재시작이 SRJN 내에 저장된 상기에서 설명된 경우와 관련하여, 4개의 경우가 더있다. 이들이 제8도에 설명된다:

1) 만약 RJN 보다 오래된 작업이 인터럽트를 발생시키었다면 (단계 815), 현재의 재시작과 펜딩인 재시작(SRJN)이 버려지며 새로운 재시작이 시작된다(단계 820).

2) 만약 RJN 과 SJN 사이의 한 작업이 인터럽트를 발생시키었다면(단계 825), 작업이 버려지는 과정중에 있기 때문에 무시된다(단계 825).

3) 만약 SJN 과 SRJN 사이의 작업이 인터럽트를 발생시키면(단계 835), SRJN내 저장된 새로운 인터럽트를 중복 기재된다(단계 840).

4) 만약 SRJN 보다 오래되지 않은 작업이 인터럽트를 발생시키면(단계 845), 인터럽트는 이 오래되지 않은 작업이 펜딩 재시작이 서비스된때 버려질 것이기 때문에 무시된다(단계 850).

재시작 메카니즘의 추가의 특징은 ACAM 의 관리와 관계가 있다. ACAM 으로의 트랜잭션은 관련된 작업이 유효하다라는 가정으로 수행된다. 실재에서는, 관련된 작업이 앞선 작업의 재시작으로 인해 결국 삭제된다. 또한 만약 ACAM 엔트리(즉, 단일변수에 대하여 발생된 멀티플 저장장소)를 만들도록 하였던 작업이 삭제되면, 그같은 ACAM 엔트리는 리네임된 DDB 레지스터의 어떠한 더이상의 사용도 막기위해서 버려지는 것이 바람직하다. 이같은 이유로, 각 ACAM 엔트리는 ACAM 엔트리를 발생시켰던 작업의 작업번호를 갖는 것이 바람직하다.

한 재시작에서 ACAM 엔트리의 삿게를 최소로하기 위해, RJN 은 각 ACAM 엔트리에 대하여 비교된다. RJN 보다 오래되지 않은 각 ACAM 엔트리는 무효로된다. 또한 각 작업이 할당되지 않기 때문에(작업의 완성에 뒤이어), 각 ACAM 엔트리와 관련된 유효비트는 리세트된다. 이같은 비트를 리세트하므로써, 처리기에서 가장 오래된 작업보다 더욱 오래된 엔트리와의 비교가 불능이 된다.

(57) 청구의 범위

청구항 1

컴퓨터 프로그램이 다수의 단계를 포함하고, 이같은 단계의 실행이 상기 다수의 단계에 상응하는 다수의 결과를 발생시키며, 상기 다수의 단계가 예정된 순서로 상기 다수의 결과를 제공하도록되는 이같은 컴퓨터 프로그램을 실행하기 위한 장치가: 다수의 접합된 지시를 형성시키기 위해 상기 다수의 단계 하나하나를 결합시키기 위한 접합수단; 다수의 작업을 형성시키기 위한 작업구성 수단으로서, 각각의 작업이 상기 다수의 접합된 지시중 한 지시를 포함하게되는 작업 구성수단; 그리고 다수의 작업을 실행하기 위한 작업 실행수단으로서, 상기 다수의 결과가 상기 예정된 순서와는 다른 순서로 획득되는 작업 실행수단을 포함함을 특징으로 하는 컴퓨터 작업 처리장치.

청구항 2

제1항에 있어서; 상기 작업중 한 작업을 실행하는 동안에 에라의 탐지가 있게되면 상기 작업구성 수단을 인터럽트하기 위한 수단; 그리고 상기 한 작업을 재시작시키기 위해 상기 인터럽트 수단에 응답하는 재시작 수단을 더욱더 포함함을 특징으로 하는 작업 처리 장치.

청구항 3

제2항에 있어서, 상기 재시작 수단이 상기 작업 실행수단에 의한 실행을 위해 또다른 다수의 작업을 형성시키기 위한 수단을 포함하며, 상기 또다른 다수의 작업 각 작업이 다수의 단계중 각기 다른 단계를 포함하고, 상기 다수의 접합된 지시중 한 지시가 이로부터 형성됨을 특징으로 하는 작업 처리장치.

청구항 4

제2항에 있어서, 다수의 작업 실행 파라미터가 상기 다수의 작업 각각과 관련이 있으며, 상기 다수의 작업 실행 파라미터중 하나가 상기 관련된 작업을 재시작시키기 위해 상기 재시작 수단에 의해 사용되고, 상기 접합수단과 상기 구성수단이 상기 다수의 작업실행 파라미터를 저장하는 코드유닛내에 포함됨을 특징으로 하는 작업 처리장치.

청구항 5

제1항에 있어서, 상기 장치가 다수의 리소스(resources)를 포함하며, 그리고 상기 작업구성 수단이 상기 다수의 접합된 지시 가운데 다수의 리소스를 할당하기 위한 할당수단을 포함함을 특징으로 하는 작업 처리장치.

청구항 6

제1항에 있어서, 상기 다수의 단계 하나하나가 각각 다수의 변수에 접근하며, 상기 장치가 다수의 각 메모리 위치내에 상기 다수의 변수에 의해 규정된 다수의 각 크기를 저장하기 위한 레지스터수단, 그리고 상기 다수의 변수를 상기 다수의 위치 적어도 하나로 맵핑하기 위한 관련 메모리 수단을 더욱더 포함함을 특징으로 하는 작업 처리장치.

청구항 7

제6항에 있어서, 상기 레지스터 수단내에 상기 다수의 각 메모리 위치를 할당하기 위한 작업 실행수단에 연결된 기준 유닛 수단을 더욱더 포함함을 특징으로 하는 작업 처리장치.

청구항 8

제7항에 있어서, 상기 장치가 한 메모리를 포함하고, 이 메모리와 상기 기준유닛 수단 사이의 자료를 전달하기 위한 메모리 유닛수단을 포함함을 특징으로 하는 작업 처리장치.

청구항 9

제1항에 있어서, 상기 작업 실행수단이 상기 다수의 결과중 적어도 하나를 결정하기 위해 적어도 하나의 산술 처리기를 포함함을 특징으로 하는 작업 처리장치.

청구항 10

컴퓨터 프로그램을 실행하기 위한 장치로서, 상기 컴퓨터 프로그램이 적어도 하나의 메인변수를 참조하는 다수의 단계를 포함하며, 이같은 장치가; 상기 메인변수를 다수의 일시적 변수로 대체시키기 위한 수단으로서, 다수의 일시적 변수 각각이 다수의 단계중 각각 상이한 단계에서 상기 메인변수를 대체하는 메인변수 대체수단; 자료크기를 저장하기 위한 다수의 위치를 포함하는 레지스터 수단; 상기 레지스터 수단내 상기 다수의 위치중 적어도 한 위치를 상기 다수의 일시적 변수로 할당하기 위한 수단; 유효자료를 가지는 상기 레지스터 수단내 다수의 위치중 선택된 각 위치에 응답하여 다수의 단계중 한 단계를 실행하기 위한 실행수단을 포함함을 특징으로 하는 작업 처리장치.

청구항 11

제10항에 있어서, 다수의 위치를 포함하는 유효자료 배열을 더욱더 포함하며, 상기 유효자로 배열내 각 위치가 레지스터 파일내 일시적 변수 각각의 유효를 나타내게됨을 특징으로 하는 작업 처리장치.

청구항 12

제11항에 있어서, 작업을 구성시키기 위한 수단을 더욱더 포함하며, 상기 작업이 상기 다수의 일시적 변수들중 선택된 변수에 대한 오퍼레이터와 참조(기준)을 포함함을 특징으로 하는 작업 처리장치.

청구항 13

제12항에 있어서, 상기 실행수단이 상기 레지스터 파일내 상응하는 위치들이 유효한 자료를 포함함을 나타내는 상기 유효 자료 배열내 다수의 위치중 하나에 응답하여 상기 작업을 실행하기 위한 수단을 포함함

을 특징으로 하는 작업 처리장치.

청구항 14

컴퓨터 프로그램이 다수의 단계를 포함하고, 이같은 단계의 실행이 상기 다수의 단계에 상응하는 다수의 결과를 발생시키며, 상기 다수의 단계가 예정된 순서로 상기 다수의 결과를 제공하도록되는 이같은 컴퓨터 프로그램을 실행하기 위한 방법이: a) 다수의 접합된 지시를 형성시키기 위해 상기 다수의 단계 하나 하나를 결합시키며; b) 다수의 작업을 형성시키고, 각각의 작업이 상기 다수의 접합된 지시중 한 지시를 포함하게되며; 그리고 c) 다수의 작업을 실행하기 위해 상기 다수의 결과가 상기 예정된 순서와는 다른 순서로 획득되는 단계들을 포함함을 특징으로 하는 컴퓨터 작업 처리방법.

청구항 15

한 컴퓨터 시스템내에서 작업을 실행하는 동안 발생되는 인터럽트로부터 기인되는 인터럽트 조건을 해결하기 위한 방법이며, 이때의 인터럽트가 작업이 인터럽트되어지도록 하고, 상기 컴퓨터 시스템이 시간 순서로 작업을 발생시키기 위한 수단을 포함하며, 상기 방법이: a) 상기 인터럽트 조건을 탐지하며; b) 다수의 작업을 버리고 상기 다수의 작업이 인터럽트된 작업과 같은 시간에 발생된 작업과 인터럽트된 작업보다 나중에 발생된 작업중 적어도 한 작업을 포함하고; 그리고 c) 다수의 재시작 지시를 실행하며, 상기 다수의 재시작 지시가 인터럽트된 작업과 인터럽트된 작업 보다 나중에 발생된 작업을 포함하는 단계들을 포함함을 특징으로 하는 컴퓨터 작업 처리방법.

청구항 16

제15항에 있어서, 상기 컴퓨터 시스템이 다수의 메모리 위치를 포함하며, 상기 작업이 다수의 단계를 포함하고, 상기 다수의 단계들중 한단계가 다수의 변수에 접근하고, 다수의 엔트리가 상기 다수의 변수들중 하나씩들을 상기 다수의 메모리 위치에 맵핑시키기 위해 발생되며, 상기 다수의 엔트리 각각이 상기 다수의 작업의 적어도 하나와 관련되며 상기 다수의 작업들 각각이 상기 인터럽트 조건에 응답하여 포기되며, 상기 방법이 상기 다수의 작업들중 포기된 작업과 관련된 다수의 엔트리 적어도 하나를 삭제하는 단계들을 더욱더 포함함을 특징으로 하는 컴퓨터 작업 처리방법.

청구항 17

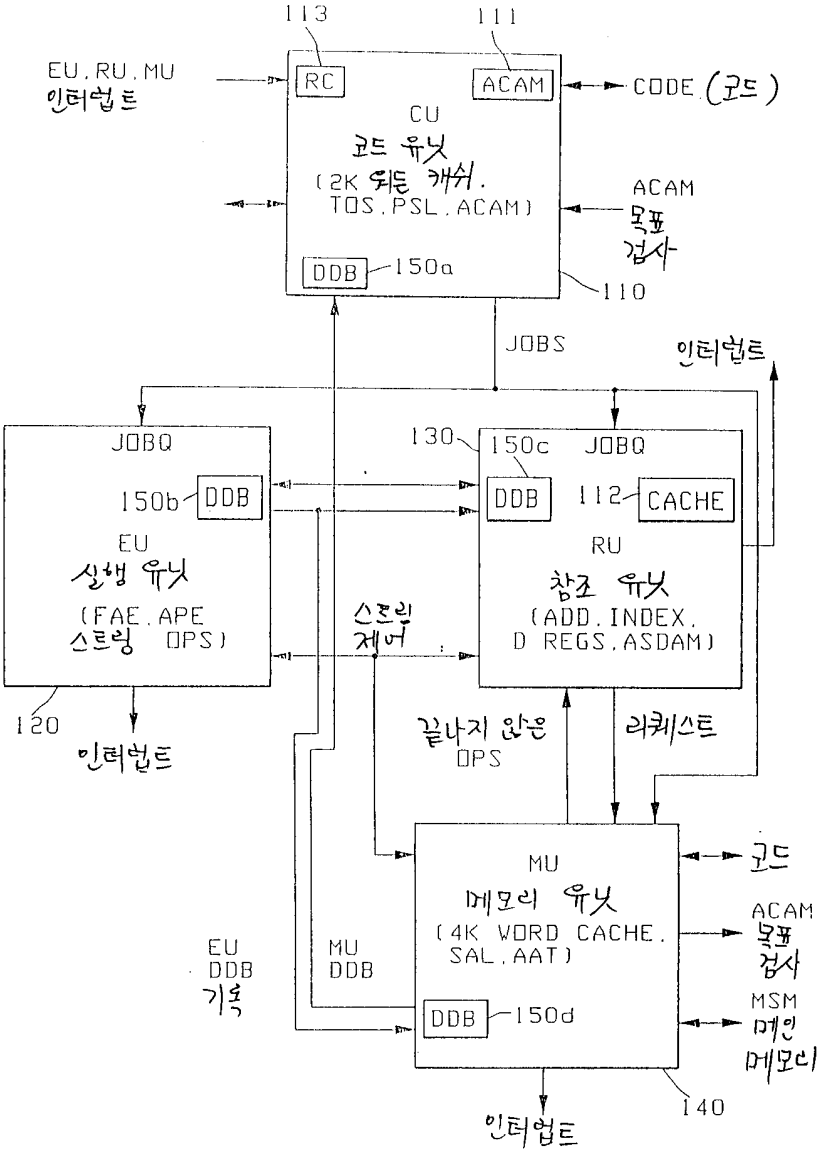
제15항에 있어서, 상기 컴퓨터 시스템이 또다른 인터럽트된 작업을 실행하는 동안 발생된 또다른 인터럽트로부터 발생되는 또다른 인터럽트 조건을 해결하기 위한 수단을 포함하며, 상기 또다른 인터럽트를 수신하게되면, 상기 방법이: d) 상기 다수의 재시작 지시를 포기하고 새로운 다수의 재시작 지시를 실행하며 상기 또다른 인터럽트 작업이 상기 인터럽트 작업보다 먼저 발생되면 상기 새로운 다수의 재시작 지시가 또다른 인터럽트 작업을 포함하며; e) 상기 인터럽트 작업이 발생된 후 그리고 인터럽트 작업보다 나중에 발생된 작업이 발생되었던 시간보다 늦지않게 또다른 인터럽트 작업이 발생되면 상기 또다른 인터럽트를 무시하고; f) 인터럽트 작업보다 나중에 발생된 작업이 발생된 뒤에 상기 또다른 인터럽트 작업이 발생되면 상기 다수의 재시작 작업을 실행한 뒤에 상기 또다른 인터럽트 작업을 실행하는 단계들을 포함함을 특징으로 하는 컴퓨터 작업 처리방법.

청구항 18

컴퓨터 프로그램을 실행하기 위한 장치이며, 이 컴퓨터 프로그램이 다수의 단계들을 포함하고 이에 의해 상기 다수의 단계들중 한 단계가 다수의 변수에 접근하고 그리고 이에 의해 상기 단계의 실행이 상기 다수의 단계에 상응하는 다수의 결과를 발생시키며, 상기 다수의 단계가 예정된 순서로 상기 다수의 결과를 제공하도록 하고, 상기 장치가: 다수의 접합된 지시들을 형성시키기 위해 상기 다수의 단계들중 하나 하나들을 결합시키고 그리고 상기 다수의 접합된 지시들로부터 다수의 작업을 형성시키기 위한 코드 유닛수단; 상기 단수의 단계들중 하나에 의해 조작하기 위해 자료를 저장하기 위한 메모리 유닛수단; 그리고 상기 다수의 단계들중 하나에 의해 조작하기 위해 상기 실행유닛으로 자료를 제공하기 위한 참조유닛 수단으로, 상기 실행유닛으로 제공된 상기 자료의 일부가 상기 메모리 유닛수단으로부터 리트리브되는 참조유닛 수단을 포함함을 특징으로 하는 컴퓨터 작업 처리장치.

도면

도면1

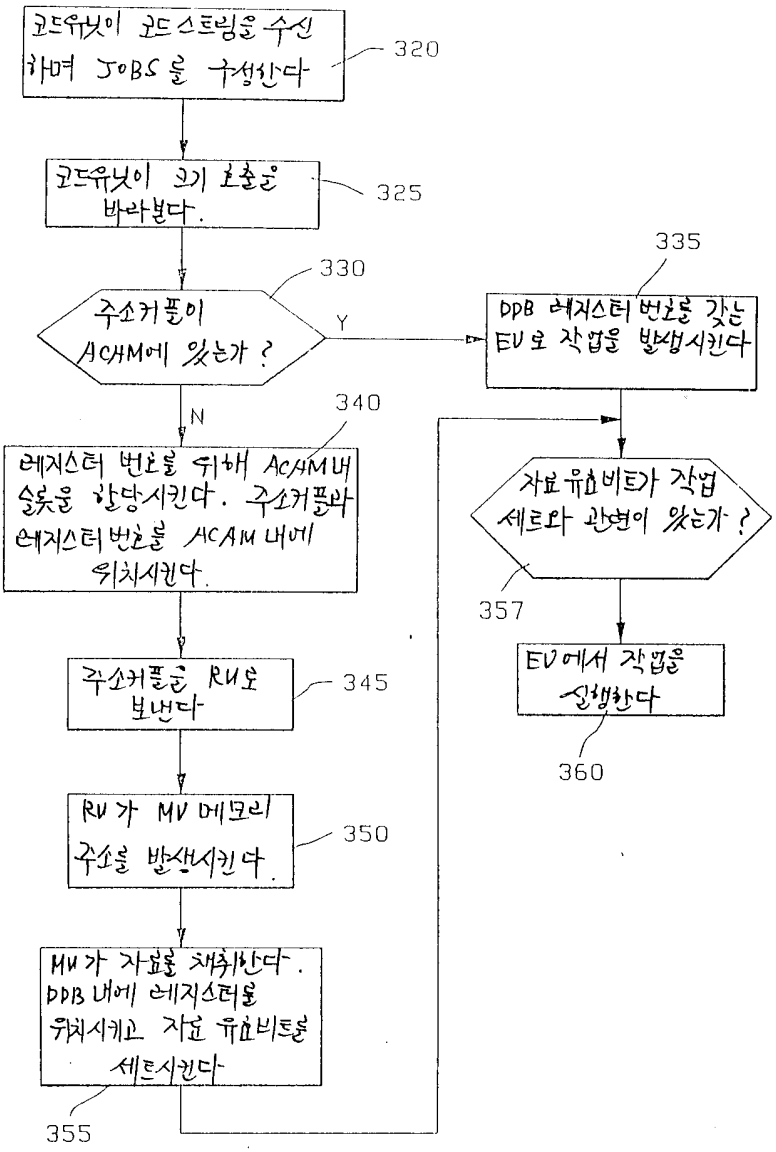


도면2

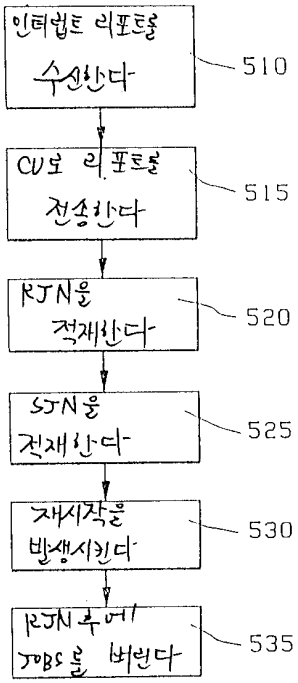
ACAM

주소지정 연결	DDB에 대한 레지스터 번호
.	.
.	.
.	.
.	.
.	.

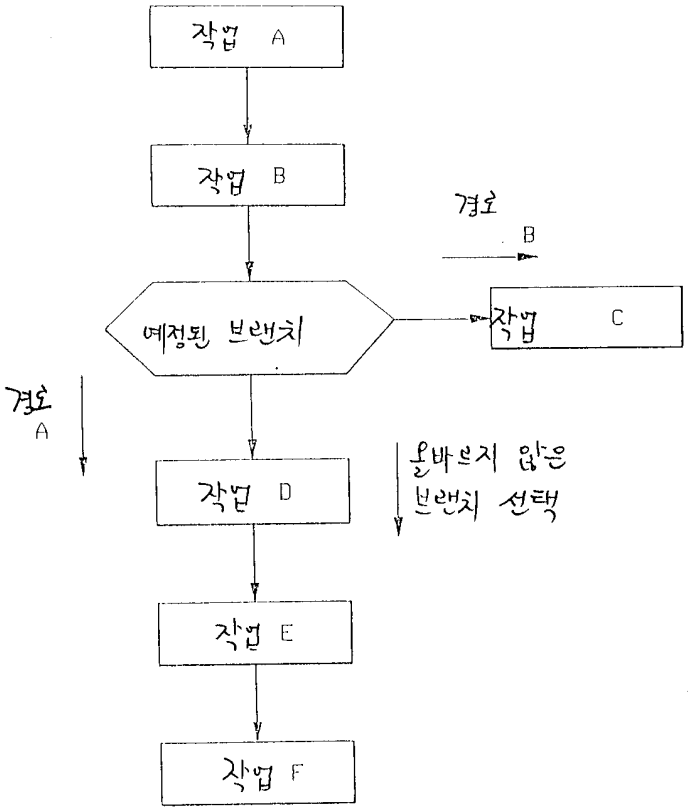
도면3


코드유닛이 코드스트림을 수신하며 JOBS를 구성한다
320
코드유닛이 오기 호출을 바라본다.
325
주소커플이 ACAM에 있는가?
330
Y
N
DPB 레지스터 번호를 갖는 EU로 작업을 발생시킨다
335
레지스터 번호를 위해 ACAM내 슬롯을 할당시킨다. 주소커플과 레지스터 번호를 ACAM 내에 위치시킨다.
340
자료유효비트가 작업 세트와 관련이 있는가?
357
EU에서 작업을 실행한다
360
주소커플을 RU로 보낸다
345
RU가 MV 메모리 주소를 발생시킨다.
350
MU가 자료를 채취한다. DPB내에 레지스터를 위치시키고 자료 유효비트를 세트시킨다
355

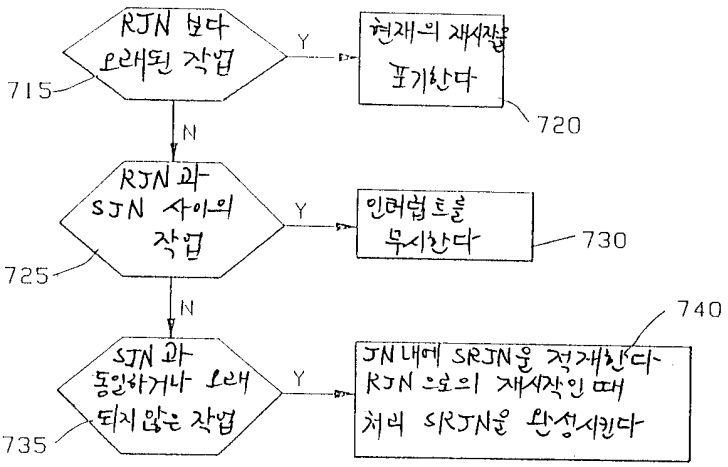
도면4


인터럽트 리포트를 수신한다
510
CU로 리포트를 전송한다
515
RJN을 적재한다
520
SJN을 적재한다
525
재시작을 발생시킨다
530
RJN 후에 JOBS를 버린다
535

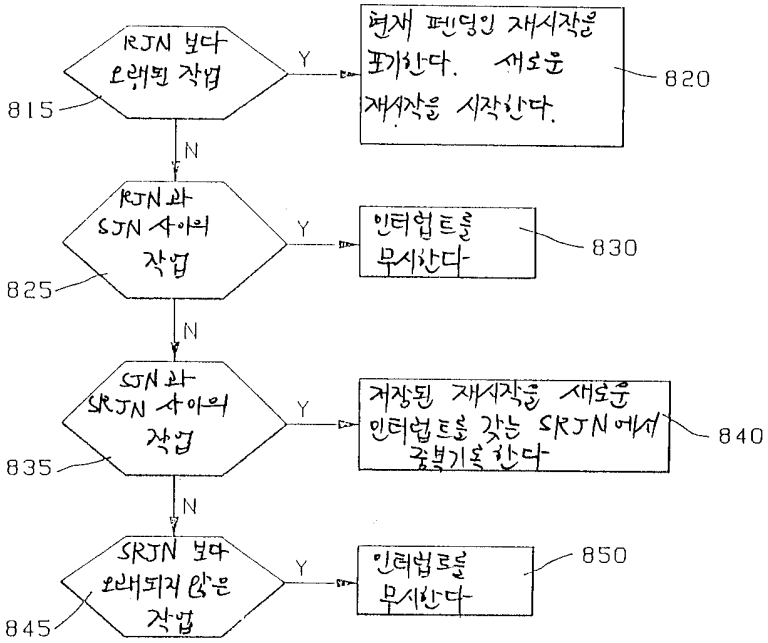
도면5


작업 A
작업 B
경로 B
예정된 브랜치
작업 C
경로 A
작업 D
올바르지 않은 브랜치 선택
작업 E
작업 F

도면6


RJN 보다 오래된 작업
715
Y
현재의 재시작을 포기한다
720
N
RJN과 SJN 사이의 작업
725
Y
인터럽트를 무시한다
730
N
SJN과 동일하거나 오래되지 않은 작업
735
Y
740
JN내에 SRJN을 적재한다 RJN으로의 재시작인 때 처리 SRJN을 완성시킨다

도면7


RJN 보다 오래된 작업
815
Y
현재 펜딩인 재시작을 포기한다. 새로운 재시작을 시작한다.
820
N
RJN과 SJN 사이의 작업
825
Y
인터럽트를 무시한다
830
N
SJN과 SRJN 사이의 작업
835
Y
저장된 재시작을 새로운 인터럽트를 갖는 SRJN에서 중복기록한다
840
N
SRJN 보다 오래되지 않은 작업
845
Y
인터럽트를 무시한다
850

도면8

