(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2020년10월28일
(11) 등록번호 10-2170966
(24) 등록일자 2020년10월22일

(51) 국제특허분류(Int. Cl.)
G06F 9/38 (2006.01) *G06F 9/30* (2018.01)
(52) CPC특허분류
G06F 9/3855 (2013.01)
G06F 9/30145 (2013.01)
(21) 출원번호 10-2019-0146601
(22) 출원일자 2019년11월15일
심사청구일자 2019년11월15일
(56) 선행기술조사문헌
US20180014734 A1
US20160224351 A1
KR1020060022973 A
KR1020030042289 A

(73) 특허권자
연세대학교 산학협력단
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
노원우
서울특별시 강남구 삼성로51길 35, 201동 1202호 (대치동, 래미안 대치 팰리스(2단지))
정이품
서울특별시 서대문구 신촌로11길 11-52, M하우스 204호(창천동)
(74) 대리인
민영준

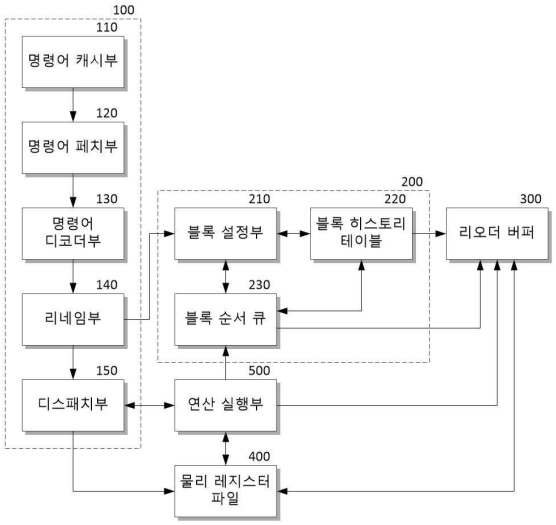
전체 청구항 수 : 총 17 항 심사관 : 최정권

(54) 발명의 명칭 **고성능 비순차 실행 코어의 리오더 버퍼 관리 장치 및 방법**

(57) 요 약

본 발명은 명령어가 반복 실행되는 프로그램의 특성을 활용하여, 다수의 명령어를 명령어 블록으로 그룹화하고, 블록 내부의 각 논리 레지스터의 값을 마지막으로 갱신하는 명령어에게만 리오더 버퍼 엔트리를 할당함으로써, 명령어 블록단위로 올바르게 구조적 상태를 갱신할 수 있도록 관리하여, 작은 크기의 리오더 버퍼로도 많은 명령어를 할당할 수 있으므로, 리오더 버퍼의 자원 효율성을 크게 향상시킬 수 있으며, 비용 및 전력 소모를 저감 시킬 수 있는 리오더 버퍼 관리 장치 및 방법을 제공할 수 있다.

대 표 도 - 도1



(52) CPC특허분류
G06F 9/3804 (2013.01)
G06F 9/3853 (2013.01)

공지예외적용 : 있음

명 세 서

청구범위

청구항 1

비순차 실행 코어의 리오더 버퍼 관리 장치에 있어서,

각각 다수의 명령어를 포함하는 적어도 하나의 명령어 블록을 엔트리로서 저장하는 블록 히스토리 테이블;

상기 코어로 인가되는 명령어가 포함된 명령어 블록이 상기 블록 히스토리 테이블의 엔트리로 저장되어 있는지 여부를 판별하여, 명령어 블록이 저장되어 있지 않은 것으로 판별되면, 인가되는 다수의 명령어를 기지정된 기준에 따라 명령어 블록으로 구분하고, 구분된 명령어 블록의 정보를 상기 블록 히스토리 테이블로 전달하여 저장하며, 인가된 명령어에 대응하여 블록 히스토리 테이블에 저장된 명령어 블록에서 동일한 논리 레지스터에 대해 최종 쓰기 작업을 수행하는 최종 쓰기 명령어를 선택하여 상기 리오더 버퍼의 엔트리로 할당하는 블록 설정부; 및

상기 코어에서 상기 명령어 블록에 포함된 명령어가 정상적으로 실행되는지 여부를 모니터링하여, 모든 명령어가 정상적으로 실행된 것으로 판별되면, 상기 리오더 버퍼에 할당된 명령어 블록에 대응하는 모든 명령어에 대해 커밋 승인하는 블록 순서 큐를 포함하는 리오더 버퍼 관리 장치.

청구항 2

제1 항에 있어서, 상기 블록 설정부는

상기 다수의 명령어 중 분기 명령어를 탐색하여 탐색된 분기 명령어를 기준으로 상기 다수의 명령어를 상기 명령어 블록으로 구분하는 리오더 버퍼 관리 장치.

청구항 3

제2 항에 있어서, 상기 블록 설정부는

명령어 블록 내에 포함되는 다수의 명령어 중 첫번째 명령어에 대한 메모리 주소를 나타내는 블록 정의자와 다수의 명령어 중 최종 쓰기 작업을 수행하는 명령어의 위치 정보를 나타내는 최종 쓰기 지시자 및 분기 명령어의 위치 및 예측 분기 경로를 나타내는 분기 지시자를 상기 명령어 블록의 정보에 포함하여 상기 블록 히스토리 테이블의 엔트리에 저장하는 리오더 버퍼 관리 장치.

청구항 4

제3 항에 있어서, 상기 블록 설정부는

상기 명령어 블록 내에서 동일한 논리 레지스터에 대해 최종 쓰기 작업을 수행하는 최종 쓰기 명령어와 함께 분기 명령어를 선택하여, 상기 리오더 버퍼의 엔트리로 할당하는 리오더 버퍼 관리 장치.

청구항 5

제4 항에 있어서, 상기 블록 설정부는

상기 블록 순서 큐로부터 연속하는 동일한 명령어 블록들에 대해 기지정된 기준 커밋 횟수 이상 연속하여 커밋 승인되면, 서로 인접한 명령어 블록을 병합하고, 병합된 명령어 블록의 정보를 상기 블록 히스토리 테이블에 재저장하는 리오더 버퍼 관리 장치.

청구항 6

제5 항에 있어서, 상기 블록 순서 큐는

상기 명령어 블록의 명령어 중 적어도 하나에서 오류가 발생된 것으로 판별되면, 리오더 버퍼에 할당된 명령어 블록에 대응하는 모든 명령어가 정상적으로 재실행될 때까지 할당된 상기 리오더 버퍼의 엔트리가 커밋되지 않도록 하는 리오더 버퍼 관리 장치.

청구항 7

제6 항에 있어서, 상기 블록 순서 큐는

상기 명령어 블록의 명령어 중 적어도 하나에서 오류가 발생된 것으로 판별되면, 명령어 블록 내에서 오류가 발생된 명령어의 위치 정보를 블록 설정부로 전달하는 리오더 버퍼 관리 장치.

청구항 8

제7 항에 있어서, 상기 블록 설정부는

상기 블록 순서 큐에서 전달된 오류가 발생된 명령어의 위치 정보에 기반하여 오류가 발생된 명령어 이전 분기 명령어를 기준으로 2개의 명령어 블록을 분할하고, 분할된 명령어 블록의 정보를 상기 블록 히스토리 테이블에 재저장하는 리오더 버퍼 관리 장치.

청구항 9

제8 항에 있어서, 상기 블록 설정부는

2개의 명령어 블록을 분할 시에 오류가 발생된 명령어 이전 분기 명령어의 분기 경로를 변경하는 리오더 버퍼 관리 장치.

청구항 10

비순차 실행 코어의 리오더 버퍼 관리 방법에 있어서,

상기 코어로 인가되는 명령어가 포함된 명령어 블록이 각각 다수의 명령어를 포함하는 적어도 하나의 명령어 블록을 엔트리로서 저장하는 블록 히스토리 테이블에 저장되어 있는지 판별하는 단계;

상기 명령어가 포함된 명령어 블록이 저장되어 있지 않은 것으로 판별되면, 인가되는 다수의 명령어를 기지정된 기준에 따라 명령어 블록으로 구분하는 단계;

구분된 명령어 블록의 정보를 상기 블록 히스토리 테이블에 저장하는 단계;

블록 히스토리 테이블에 저장된 명령어 블록 중 인가된 명령어에 대응하는 명령어 블록의 다수의 명령어 중 동일한 논리 레지스터에 대해 최종 쓰기 작업을 수행하는 최종 쓰기 명령어를 선택하여 상기 리오더 버퍼의 엔트리로 할당하는 단계; 및

상기 코어에서 상기 명령어 블록에 포함된 명령어가 정상적으로 실행되는지 여부를 모니터링하여, 모든 명령어가 정상적으로 실행된 것으로 판별되면, 상기 리오더 버퍼에 할당된 명령어 블록에 대응하는 모든 명령어에 대해 커밋 승인하는 단계를 포함하는 리오더 버퍼 관리 방법.

청구항 11

제10 항에 있어서, 상기 구분하는 단계는

상기 다수의 명령어 중 분기 명령어를 탐색하여 탐색된 분기 명령어를 기준으로 상기 다수의 명령어를 상기 명령어 블록으로 구분하는 리오더 버퍼 관리 방법.

청구항 12

제11 항에 있어서, 상기 명령어 블록의 정보는

명령어 블록 내에 포함되는 다수의 명령어 중 첫번째 명령어에 대한 메모리 주소를 나타내는 블록 정의자와 다수의 명령어 중 최종 쓰기 작업을 수행하는 명령어의 위치 정보를 나타내는 최종 쓰기 지시자 및 분기 명령어의 위치 및 예측 분기 경로를 나타내는 분기 지시자를 상기 명령어 블록의 정보에 포함하여 상기 블록 히스토리 테이블의 엔트리에 저장하는 리오더 버퍼 관리 방법.

청구항 13

제12 항에 있어서, 상기 리오더 버퍼의 엔트리로 할당하는 단계는

상기 명령어 블록 내에서 동일한 논리 레지스터에 대해 최종 쓰기 작업을 수행하는 최종 쓰기 명령어와 함께 분

기 명령어를 선택하여, 상기 리오더 버퍼의 엔트리로 할당하는 리오더 버퍼 관리 방법.

청구항 14

제13 항에 있어서, 상기 리오더 버퍼 관리 방법은

연속하는 동일한 명령어 블록들에 대해 기지정된 기준 커밋 횟수 이상 연속하여 커밋 승인되면, 서로 인접한 명령어 블록을 병합하는 단계; 및

병합된 명령어 블록의 정보를 상기 블록 히스토리 테이블에 재저장하는 단계를 더 포함하는 리오더 버퍼 관리 방법.

청구항 15

제13 항에 있어서, 상기 리오더 버퍼 관리 방법은

상기 명령어 블록의 명령어 중 적어도 하나에서 오류가 발생된 것으로 판별되면, 리오더 버퍼에 할당된 명령어 블록에 대응하는 모든 명령어가 정상적으로 재실행될 때까지 할당된 상기 리오더 버퍼의 엔트리가 커밋되지 않도록 하는 단계를 더 포함하는 리오더 버퍼 관리 방법.

청구항 16

제14 항에 있어서, 상기 리오더 버퍼 관리 방법은

상기 명령어 블록의 명령어 중 적어도 하나에서 오류가 발생된 것으로 판별되면, 명령어 블록 내에서 오류가 발생된 명령어의 위치 정보를 획득하는 단계;

획득된 오류가 발생된 명령어의 위치 정보에 기반하여 오류가 발생된 명령어 이전 분기 명령어를 기준으로 2개의 명령어 블록을 분할하는 단계; 및

분할된 명령어 블록의 정보를 상기 블록 히스토리 테이블에 재저장하는 단계를 더 포함하는 리오더 버퍼 관리 방법.

청구항 17

제16 항에 있어서, 상기 블록 히스토리 테이블에 재저장하는 단계는

2개의 명령어 블록을 분할 시에 오류가 발생된 명령어 이전 분기 명령어의 분기 경로를 변경하는 리오더 버퍼 관리 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 리오더 버퍼 관리 장치 및 방법에 관한 것으로, 비순차 실행 코어의 리오더 버퍼 관리 장치 및 방법에 관한 것이다.

배 경 기 술

[0002] 비순차 실행(Out-of-order execution: OoOE 또는 비순차적 명령어 처리라고도 함)은 고성능 마이크로프로세서가 특정한 종류의 지연으로 인해 낭비될 수 있는 명령 사이클을 이용하고자 하는 패러다임이다. 비순차 실행은 명령 실행 효율을 높이기 위해 명령을 순서에 따라 처리하지 않는 기법으로 현재 많은 프로세서에 채용되고 있다.

[0003] 최신 비순차실행 코어 구조는 비순차적으로 실행되는 명령어들이 코어의 구조적 상태(architectural state)를 원래의 프로그램 순서에 따라 올바르게 갱신(update)할 수 있도록 하기 위하여 리오더 버퍼(Reorder Buffer) 및 물리 레지스터 파일(Physical Register File)을 사용하고 있다.

[0004] 이중 리오더 버퍼는 디코딩(decoding)된 모든 명령어를 프로그램 순서에 따라 저장하고 있는 구조체로서 연산을 완료한 명령어를 원래의 순서대로 커밋(commit) 함으로써 코어의 구조적 상태가 올바르게 갱신될 수 있도록 한다. 이를 위해 각 명령어는 비순차 실행이 수행되기 이전 디스패치(dispatch) 단계에서 원래의 순서대로 리오더 버퍼 엔트리를 할당 받는다. 즉, 실행되는 모든 명령어는 연산을 완료하고 파이프라인에서 커밋되기 전까지

리오더 버퍼 엔트리를 할당 받고 있어야 한다.

[0005] 한편 최신 프로세서들은 코어 내부에서 프로그램의 실행 성능을 높이기 위해 프로그램에 내재된 명령어 수준의 병렬성(instruction-level parallelism, ILP)을 활용하여 데이터 의존성이 없는 다수의 명령어를 동시에 실행하는 방법을 사용하고 있다. 이때 리오더 버퍼의 용량이 적다면, 다른 구성 요소가 병렬성을 제공할 수 있더라도, 즉 파이프 라인 내부의 유휴 연산 자원이 충분하더라도, 연산 결과에 따른 코어의 상태를 올바르게 갱신할 수 없으므로, 병렬적으로 실행할 수 있는 명령의 수가 제한된다. 이에 최신 프로세서들은 점점 더 큰 용량의 리오더 버퍼를 채용하고 있으며, 이러한 추세는 앞으로 계속 이어질 것으로 전망된다.

[0006] 그러나 리오더 버퍼는 실행되는 모든 명령어에 의해 각 명령어 당 적어도 두 번 이상 접근되기 때문에, 크기가 커질수록 칩을 구현하기 위한 트랜지스터 개수 및 전력 소모량이 증가하게 되어 전체 시스템의 구성 비용 및 전력 효율에 큰 오버헤드가 될 수 있다.

선행기술문헌

특허문헌

[0007] (특허문헌 0001) 한국 등록 특허 제10-0496904호 (2005.06.14 등록)

발명의 내용

해결하려는 과제

[0008] 본 발명의 목적은 코어의 구조적 상태를 올바르게 갱신하는 동시에 리오더 버퍼의 엔트리를 특정 명령어들에 선택적으로 할당함으로써 작은 크기로 높은 수준의 명령어 실행 성능을 지원할 수 있는 리오더 버퍼 관리 장치 및 방법을 제공하는데 있다.

[0009] 본 발명의 다른 목적은 리오더 버퍼에 명령어를 선택적으로 할당하여 리오더 버퍼의 유효 크기를 증대시켜, 제조 비용 및 전력 소비를 저감시킬 수 있는 리오더 버퍼 관리 장치 및 방법을 제공하는데 있다.

과제의 해결 수단

[0010] 상기 목적을 달성하기 위한 본 발명의 일 실시예에 따른 비순차 실행 코어의 리오더 버퍼 관리 장치는 각각 다수의 명령어를 포함하는 적어도 하나의 명령어 블록을 엔트리로서 저장하는 블록 히스토리 테이블; 상기 코어로 인가되는 명령어가 포함된 명령어 블록이 상기 블록 히스토리 테이블의 엔트리로 저장되어 있는지 여부를 판별하여, 명령어 블록이 저장되어 있지 않은 것으로 판별되면, 인가되는 다수의 명령어를 기지정된 기준에 따라 명령어 블록으로 구분하고, 구분된 명령어 블록의 정보를 상기 블록 히스토리 테이블로 전달하여 저장하며, 인가된 명령어에 대응하여 블록 히스토리 테이블에 저장된 명령어 블록에서 동일한 논리 레지스터에 대해 최종 쓰기 작업을 수행하는 최종 쓰기 명령어를 선택하여 상기 리오더 버퍼의 엔트리로 할당하는 블록 설정부; 및 상기 코어에서 상기 명령어 블록에 포함된 명령어가 정상적으로 실행되는지 여부를 모니터링하여, 모든 명령어가 정상적으로 실행된 것으로 판별되면, 상기 리오더 버퍼에 할당된 명령어 블록에 대응하는 모든 명령어에 대해 커밋 승인하는 블록 순서 큐를 포함한다.

[0011] 상기 블록 설정부는 상기 다수의 명령어 중 분기 명령어를 탐색하여 탐색된 분기 명령어를 기준으로 상기 다수의 명령어를 상기 명령어 블록으로 구분할 수 있다.

[0012] 상기 블록 설정부는 명령어 블록 내에 포함되는 다수의 명령어 중 첫번째 명령어에 대한 메모리 주소를 나타내는 블록 정의자와 다수의 명령어 중 최종 쓰기 작업을 수행하는 명령어의 위치 정보를 나타내는 최종 쓰기 지시자 및 분기 명령어의 위치 및 예측 분기 경로를 나타내는 분기 지시자를 상기 명령어 블록의 정보에 포함하여 상기 블록 히스토리 테이블의 엔트리에 저장할 수 있다.

[0013] 상기 블록 설정부는 상기 명령어 블록 내에서 동일한 논리 레지스터에 대해 최종 쓰기 작업을 수행하는 최종 쓰기 명령어와 함께 분기 명령어를 선택하여, 상기 리오더 버퍼의 엔트리로 할당할 수 있다.

[0014] 상기 블록 설정부는 상기 블록 순서 큐로부터 연속하는 동일한 명령어 블록들에 대해 기지정된 기준 커밋 횟수 이상 연속하여 커밋 승인되면, 서로 인접한 명령어 블록을 병합하고, 병합된 명령어 블록의 정보를 상기 블록

히스토리 테이블에 재저장할 수 있다.

[0015] 상기 블록 순서 큐는 상기 명령어 블록의 명령어 중 적어도 하나에서 오류가 발생된 것으로 판별되면, 리오더 버퍼에 할당된 명령어 블록에 대응하는 모든 명령어가 정상적으로 재실행될 때까지 할당된 상기 리오더 버퍼의 엔트리가 커밋되지 않도록 할 수 있다.

[0016] 상기 블록 순서 큐는 상기 명령어 블록의 명령어 중 적어도 하나에서 오류가 발생된 것으로 판별되면, 명령어 블록 내에서 오류가 발생된 명령어의 위치 정보를 블록 설정부로 전달할 수 있다.

[0017] 상기 블록 설정부는 상기 블록 순서 큐에서 전달된 오류가 발생된 명령어의 위치 정보에 기반하여 오류가 발생된 명령어 이전 분기 명령어를 기준으로 2개의 명령어 블록을 분할하고, 분할된 명령어 블록의 정보를 상기 블록 히스토리 테이블에 재저장할 수 있다.

[0018] 상기 블록 설정부는 2개의 명령어 블록을 분할 시에 오류가 발생된 명령어 이전 분기 명령어의 분기 경로를 변경할 수 있다.

[0019] 상기 목적을 달성하기 위한 본 발명의 다른 실시예에 따른 비순차 실행 코어의 리오더 버퍼 관리 장치 및 방법은 상기 코어로 인가되는 명령어가 포함된 명령어 블록이 각각 다수의 명령어를 포함하는 적어도 하나의 명령어 블록을 엔트리로서 저장하는 블록 히스토리 테이블에 저장되어 있는지 판별하는 단계; 상기 명령어가 포함된 명령어 블록이 저장되어 있지 않은 것으로 판별되면, 인가되는 다수의 명령어를 기지정된 기준에 따라 명령어 블록으로 구분하는 단계; 구분된 명령어 블록의 정보를 상기 블록 히스토리 테이블에 저장하는 단계; 블록 히스토리 테이블에 저장된 명령어 블록 중 인가된 명령어에 대응하는 명령어 블록의 다수의 명령어 중 동일한 논리 레지스터에 대해 최종 쓰기 작업을 수행하는 최종 쓰기 명령어를 선택하여 상기 리오더 버퍼의 엔트리로 할당하는 단계; 및 상기 코어에서 상기 명령어 블록에 포함된 명령어가 정상적으로 실행되는지 여부를 모니터링하여, 모든 명령어가 정상적으로 실행된 것으로 판별되면, 상기 리오더 버퍼에 할당된 명령어 블록에 대응하는 모든 명령어에 대해 커밋 승인하는 단계를 포함한다.

발명의 효과

[0020] 따라서, 본 발명의 실시예에 따른 리오더 버퍼 관리 장치 및 방법은 명령어가 반복 실행되는 프로그램의 특성을 활용하여, 다수의 명령어를 명령어 블록으로 그룹화하고, 블록 내부의 각 논리 레지스터의 값을 마지막으로 갱신하는 명령어에게만 리오더 버퍼 엔트리를 할당함으로써, 명령어 블록단위로 올바르게 구조적 상태를 갱신할 수 있도록 관리한다. 따라서 작은 크기의 리오더 버퍼로도 많은 명령어를 할당할 수 있으므로, 리오더 버퍼의 자원 효율성을 크게 향상시킬 수 있으며, 비용 및 전력 소모를 저감 시킬 수 있다.

도면의 간단한 설명

[0021] 도 1은 본 발명의 일 실시예에 따른 비순차 실행 코어의 개략적 구조를 나타낸다.

도 2는 도 1의 비순차 실행 코어에서 리오더 버퍼의 유효 크기를 증가시키기 위해 리오더 버터 엔트리를 할당하는 개념을 설명하기 위한 도면이다.

도 3은 블록 크기에 따른 리오더 버퍼의 유효 크기 변화를 설명하기 위한 도면이다.

도 4는 도 1의 블록 설정부가 명령어 블록을 병합하거나 분할하는 과정을 설명하기 위한 도면이다.

도 5는 본 발명의 일 실시예에 따른 비순차 실행 코어의 리오더 버퍼 관리 방법을 나타낸다.

발명을 실시하기 위한 구체적인 내용

[0022] 본 발명과 본 발명의 동작상의 이점 및 본 발명의 실시에 의하여 달성되는 목적을 충분히 이해하기 위해서는 본 발명의 바람직한 실시예를 예시하는 첨부 도면 및 첨부 도면에 기재된 내용을 참조하여야만 한다.

[0023] 이하, 첨부한 도면을 참조하여 본 발명의 바람직한 실시예를 설명함으로써, 본 발명을 상세히 설명한다. 그러나, 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며, 설명하는 실시예에 한정되는 것이 아니다. 그리고, 본 발명을 명확하게 설명하기 위하여 설명과 관계없는 부분은 생략되며, 도면의 동일한 참조부호는 동일한 부재임을 나타낸다.

[0024] 명세서 전체에서, 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라, 다른 구성요소를 더 포함할 수 있는 것을 의미한다. 또한, 명세서에 기

재된 "...부", "...기", "모듈", "블록" 등의 용어는 적어도 하나의 기능이나 동작을 처리하는 단위를 의미하며, 이는 하드웨어나 소프트웨어 또는 하드웨어 및 소프트웨어의 결합으로 구현될 수 있다.

[0025] 도 1은 본 발명의 일 실시예에 따른 비순차 실행 코어의 개략적 구조를 나타내고, 도 2는 도 1의 비순차 실행 코어에서 리오더 버퍼의 유효 크기를 증가시키기 위해 리오더 버퍼 엔트리를 할당하는 개념을 설명하기 위한 도면이며, 도 3은 블록 크기에 따른 리오더 버퍼의 유효 크기 변화를 설명하기 위한 도면이다. 그리고 도 4는 도 1의 블록 설정부가 명령어 블록을 병합하거나 분할하는 과정을 설명하기 위한 도면이다.

[0026] 도 1을 참조하면, 비순차 실행 코어는 명령어 이슈부(100), 명령어 블록화부(200), 리오더 버퍼(300), 물리 레지스터 파일(400) 및 연산 실행부(500)를 포함한다.

[0027] 우선 명령어 이슈부(100)는 명령어 캐시부(110), 명령어 페치부(120), 명령어 디코더부(130), 리네임부(140) 및 디스패치부(150)를 포함할 수 있다.

[0028] 명령어 캐시(instruction cache)부(110)는 인가된 명령어를 임시로 저장하고, 명령어 페치(instruction fetch)부(120)는 명령어 캐시부(110)에 저장된 명령어를 순차적으로 인가받아 저장한다. 명령어 페치부(120)는 일종의 큐(queue)로서 선입 선출 방식으로 명령어를 인가받아 저장하고 출력할 수 있다. 즉 명령어 페치부(120)는 먼저 입력된 명령어를 우선 출력할 수 있다. 그리고 출력된 명령어는 명령어 페치부(120)에서 제거된다. 또한 명령어 페치부(120)는 명령어 페치 과정에서 다수의 명령어 중 분기 명령어에서 분기되는 경로에 대한 분기 예측(branch prediction)을 함께 수행할 수 있다.

[0029] 명령어 디코더부(130)는 명령어 페치부(120)에서 인가되는 명령어를 내부적으로 사용하는 오피 코드(Opcode)와 소스 및 목적 레지스터 주소들로 변환하여 디코딩한다. 여기에서 하나의 명령어가 가질 수 있는 목적 레지스터 수는 한 개이며, 소스 레지스터 수는 두 개이다. 변환된 명령어들은 리네임부(140)로 전달된다.

[0030] 리네임부(140)는 비순차 실행 코어에서 비순차적으로 실행되는 명령어들 사이의 데이터 의존성을 해결하기 위해 구비되는 구성으로, 논리적 레지스터를 물리 레지스터 파일(Physical Register File: 이하 PRF)(400)의 물리적 레지스터로 변환한다. 즉 명령어 디코더부(130)에서 인가된 명령 명령어의 논리적 목적 레지스터에 대해서 PRF(400)의 물리적 목적 레지스터(physical destination register)를 할당하며, 논리적 소스 레지스터에 대해서는 PRF의 대응되는 물리적 소스 레지스터(physical source register)로 변환하여 준다.

[0031] 디스패치부(150)는 리네임부(140)에 의해 논리 레지스터와 물리 레지스터가 변환된 명령어를 인가받아 저장하고, 저장된 명령어를 순차적으로 연산 실행부(500)로 디스패치(dispatch)한다. 디스패치부(150)는 리네임부(140)에서 전달되는 명령 각각을 엔트리로 저장하며, 각 엔트리에는 오피 코드, 물리 목적 레지스터 주소, 물리 소스 레지스터 주소들이 저장될 수 있다. 디스패치부(150)는 저장된 명령어에 대해서 소스 레지스터들의 데이터가 모두 준비되고 사용하고자 하는 연산 실행부(500)가 가용한 경우에, 명령어를 연산 실행부(500)로 이슈(issue)한다.

[0032] 명령어 블록화부(200)는 명령어 이슈부(100)에서 이슈되는 다수의 명령어를 분석하여 기지정된 방식에 따라 블록화하고, 블록화된 다수의 명령어에서 동일한 목적 레지스터에 대해 최종적으로 쓰기 동작을 수행하는 명령어를 탐색하여 리오더 엔트리로서 리오더 버퍼(300)로 전달한다.

[0033] 명령어 블록화부(200)는 블록 설정부(210), 블록 히스토리 테이블(220) 및 블록 순서 큐(230)를 포함할 수 있다. 블록 설정부(210)는 명령어 이슈부(100)로부터 연산 실행부(500)에서 실행될 명령어를 인가받아 분석한다. 일예로 블록 설정부(210)는 명령어 이슈부(100)의 리네임부(140)로부터 다수의 명령어를 인가받을 수 있으나, 경우에 따라 명령어 페치부(120), 명령어 디코더부(130) 또는 디스패치부(150)로부터 명령어를 인가받을 수 있다.

[0034] 블록 설정부(210)는 명령어가 인가되면, 블록 히스토리 테이블(Block History Table: 이하 BLHT)(220)을 탐색하여, 인가된 명령어가 포함된 명령어 블록이 BLHT(220)에 존재하는지 판별한다.

[0035] 만일 명령어 블록이 존재하지 않는 것으로 판별되면, 블록 설정부(210)는 인가되는 다수의 명령어들을 블록화하여 명령어 블록을 생성한다. 여기서 블록 설정부(210)는 인가되는 다수의 명령어들 중 분기(branch) 명령어를 탐색하고, 분기 명령어가 확인되면, 분기 명령어가 인가될 때까지의 이전 명령어 전체를 하나의 명령어 그룹으로 설정한다. 그리고 설정된 명령어 블록을 BLHT(220)에 BLHT 엔트리로 할당하여 저장한다. 여기서 브랜치 명령어를 기준으로 구분되는 명령어 그룹 각각을 기본 블록(basic block)이라 한다. 즉 기본 블록은 인접한 두개의 브랜치 명령어 사이에 위치하는 다수의 명령어를 그룹화한 명령어 그룹을 의미한다.

[0036] 이때, 블록 설정부(210)는 BLHT 엔트리에 명령어 블록을 정의하는 블록 정의자와 명령어 블록 내부의 다수의 명령어 중 최종 쓰기(last write) 작업을 수행하는 명령어를 지시하는 최종 쓰기 지시자를 포함시킬 수 있다.

[0037] 블록 설정부(210)는 우선 블록에 포함되는 다수의 명령어 중 첫번째 명령어에 대한 메모리 주소를 프로그램 카운터(program counter: 이하 PC)로부터 인가받아 블록 정의자로서 BLHT 엔트리에 포함시킬 수 있다. 그리고 다수의 명령어 중 최종 쓰기 작업을 수행하는 명령어 각각의 위치를 비트 벡터 형식으로 작성하여 최종 쓰기 지시자로서 BLHT 엔트리에 포함시킬 수 있다. 일예로 명령어 블록에 5개의 명령어가 포함되어 있고, 이중 3번째와 4번째 명령어가 최종 쓰기 명령어인 경우, 블록 설정부(210)는 "00110"와 같은 비트 벡터 형식으로 최종 쓰기 지시자를 작성하여 BLHT 엔트리에 포함시킬 수 있다. 또한 블록 설정부(210)는 블록 내부의 분기 명령어의 분기 경로(branch path)를 나타내는 분기 지시자를 BLHT 엔트리에 포함시킬 수도 있다. 파이프 라인 기법에서 분기 명령어의 분기 경로는 분기 명령어의 조건이 참으로 판정되어 분기 목적지로 이동하는 테이큰(taken)과 조건이 거짓으로 판정되어 다음 명령어를 실행하는 낫테이큰(not taken)으로 구분되며, 블록 설정부(210)는 블록 내의 분기 명령어의 분기 경로를 나타내는 분기 지시자를 비트값으로 표현하여 BLHT 엔트리에 포함시킬 수 있다.

[0038] 한편, 블록 설정부(210)는 인가된 명령어가 포함된 명령어 블록이 BLHT(220)에 이미 존재하는 것으로 판별되면, 블록 순서 큐(230)로부터 전달되는 커밋 승인 신호 또는 실행 오류 신호에 따라 BLHT(220)에 저장된 다수의 BLHT 엔트리, 즉 다수의 명령어 블록 중 인접한 블록을 병합하거나, 이전 병합된 명령어 블록에서 일부 명령어 블록을 별도로 분할할 수 있다.

[0039] BLHT(220)는 블록 설정부(210)에 의해 설정된 명령어 블록 각각을 BLHT 엔트리로서 저장한다. 그리고 BLHT(220)는 BLHT 엔트리에 기반하여 최종 쓰기 지시자와 분기 지시자에 대응하는 명령어들을 각각 ROB(300)로 전달하여 저장한다. 즉 BLHT 엔트리에 포함된 최종 쓰기 지시자와 분기 지시자에 대응하는 명령어들 각각을 ROB(300)의 엔트리로 저장한다. 또한 BLHT(220)는 블록 순서 큐(230)의 요청에 따라 저장된 BLHT 엔트리를 블록 순서 큐(230)로 전달한다.

[0040] 블록 순서 큐(230)는 블록 설정부(210)에서 생성되거나 탐색된 BLHT 엔트리를 BLHT(220)로부터 인가받는다. 블록 순서 큐(230)는 BLHT 엔트리에 대응하는 명령어 블록에 포함된 다수의 명령어 중 첫번째 명령어가 디스패치부(150)에서 디스패치되면, BLHT(220)에 요청하여 대응하는 명령어 블록을 인가받아 할당하고, 명령어 블록에 포함된 모든 명령어가 연산 실행부(500)에서 정상적으로 실행되는지 모니터링한다. 그리고 명령어 블록의 모든 명령어가 정상적으로 실행된 것으로 판별되면, 명령어 블록에 포함된 명령어에 대한 커밋 승인 신호를 ROB(300)와 디스패치부(150)로 전달한다. 즉 ROB(300)와 디스패치부(150)가 해당 명령어 블록에 포함된 명령어에 대한 할당을 해제하도록 한다.

[0041] 그러나 만일 명령어 블록의 명령어가 정상적으로 실행되지 않은 것으로 판단되면, 실행 오류 신호를 ROB(300)와 디스패치부(150)로 전달하여 해당 명령어 블록의 명령어의 실행을 무효화(pipeline flush)하고, 재실행되도록 한다. 즉 명령어 블록에 포함된 명령어들에 대한 할당이 유지되도록 한다.

[0042] 일예로 블록 순서 큐(230)는 BLHT 엔트리에 의해 지정된 명령어 블록의 명령어를 실행 중 분기 경로에 오류가 발생하거나 예외 상황이 발생되면, 실행 오류 신호를 ROB(300)와 디스패치부(150)로 출력하여 BLHT 엔트리, 즉 명령어 블록의 모든 명령어에 대한 실행 결과를 무효화하고, 해당 명령어 블록이 재실행되도록 한다.

[0043] 또한 블록 순서 큐(230)는 커밋 승인 신호 또는 실행 오류 신호를 블록 설정부(210)로 전송하여, 블록 설정부(210)가 커밋 승인 신호 또는 실행 오류 신호에 응답하여, 명령어 블록을 병합하거나 분할하도록 한다. 이때 블록 순서 큐(230)는 실행 오류 신호와 함께 명령어 블록에서 실행 오류가 발생된 명령어 정보(예를 들면 명령어 위치 정보)를 함께 블록 설정부(210)로 전달할 수 있다.

[0044] 이에 블록 설정부(210)는 동일한 명령어 블록에 대해 기지정된 기준 커밋 횟수(예를 들면 5회) 이상 커밋 승인 신호가 인가되면, 해당 명령어 블록과 인접한 다음 명령어 블록을 병합할 수 있다. 그러나 동일한 명령어 블록에 대해 기지정된 기준 오류 횟수(예를 들면 2회) 이상 실행 오류 신호가 인가되면, 해당 명령어 블록에서 실행 오류가 발생된 명령어에 인접한 이전 분기 명령어을 기준으로 명령어 블록을 분할할 수 있다. 이는 블록 설정부(210)가 상기한 바와 같이 분기 명령어를 기반으로 기본 블록을 생성하기 때문이다.

[0045] ROB(300)는 BLHT(220)에 저장된 다수의 BLHT 엔트리 중 블록 설정부(210)에 의해 지정된 BLHT 엔트리에 기반하여 최종 쓰기 지시자와 분기 지시자에 대응하는 명령어들 각각을 ROB 엔트리로 저장한다. ROB(300)는 선입 선출법에 따라 다수의 명령어를 ROB 엔트리로 할당하고, 할당해제 할 수 있다.

[0046] 여기서 ROB(300)가 블록 설정부(210)에 의해 지정된 BLHT 엔트리에 기반하여 최종 쓰기 지시자와 분기 지시자에 대응하는 명령어들 각각을 ROB 엔트리로 저장한다.

[0047] 본 실시예에서 블록화부(200)가 다수의 명령어를 명령어 블록으로 병합하고, 명령어 블록에 최종 쓰기 지시자를 포함시켜, BLHT 엔트리로 할당하고, ROB(300)가 최종 쓰기 지시자에 대응하는 명령을 ROB 엔트리로 할당하여 저장하는 것은 ROB(300)의 유효 크기(Effective size)를 증가시키기 위해서이다.

[0048] 도 2를 참조하면 (a)는 기존의 방식에 따라 블록화를 수행하지 않고, 명령어 이슈부(100)에 인가된 모든 명령어 각각을 ROB 엔트리로 할당하는 경우를 나타낸다. ROB(300)는 연산이 완료된 명령어를 원래의 순서대로 커밋하여 코어의 구조적 상태가 올바르게 갱신되도록 하기 위해 포함된 구성으로 기존에는 (a)에 도시된 바와 같이, 모든 명령어가 프로그램 순서에 따라, 즉 명령어가 인가된 순서에 따라 저장되어야 하였다.

[0049] 그러나 모든 명령어가 저장되지 않고 최종적으로 해당 레지스터의 쓰기 작업이 정상적으로 수행된 경우에 커밋하도록 구성되어도 코어의 구조적 상태는 올바르게 갱신될 수 있다. 본 실시예에서는 이점을 감안하여, ROB(300)에는 최종 쓰기 지시자(또는 분기 지시자)에 대응하는 명령어만이 포함되도록 함으로써, ROB에 더 많은 명령어가 엔트리될 수 있도록 한다. 즉 물리적인 용량의 증대 없이 ROB(300)의 유효 크기를 증가시킨다.

[0050] 즉 (b)에 도시된 바와 같이, (a)와 동일한 5개의 명령어(I_1 ~ I_5) 중 제1 및 제2 명령어(I_1, I_2)는 각각 제1 및 제2 레지스터(R1, R2)에 연산 결과를 쓰기 작업하는 명령어이지만, 이후 제3 및 제4 명령어(I_3, I_4) 또한 제1 및 제2 레지스터(R1, R2)에 연산 결과를 쓰기 작업한다. 즉 제1 및 제2 명령어(I_1, I_2)의 작업 결과에 대해 제3 및 제4 명령어(I_3, I_4)가 덮어쓰기(Overwrtie)를 수행하게 된다. 그리고 제5 명령어(I_5) 또한 이후 추가적으로 덮어쓰기되지 않는 제3 레지스터(R3)에 연산 결과를 쓰기 작업한다. 따라서 최종적으로 해당 레지스터에 쓰기 작업을 수행하는 명령어는 제3 내지 제5 명령어(I_3 ~ I_5)이다.

[0051] 이에 본 실시예에서 블록화부(200)는 (c)에 도시된 바와 같이 다수의 명령어를 블록화하여 명령어 블록을 생성하고, 블록 내의 다수의 명령어 중 최종 쓰기 지시자에 대응하는 명령어들이 ROB 엔트리로 저장되도록 함으로써, 명령어 블록에 포함되는 5개의 명령어(I_1 ~ I_5) 중 3개의 명령어(I_3 ~ I_5)만이 ROB(300)에 ROB 엔트리로서 할당되어 저장되도록 한다.

[0052] 결과적으로 기존에는 ROB(300)에 5개의 명령어(I_1 ~ I_5)가 저장되어야 하는데 반해, 본 실시예에서는 3개의 명령어(I_3 ~ I_5)만 저장되어도 동일하게 코어의 구조적 상태는 올바르게 갱신할 수 있다. 이는 ROB(300)가 더 적은 개수의 명령어만으로도 모든 명령어를 저장하는 것과 동일하게 코어의 구조적 상태를 유지할 수 있도록 하여 ROB(300)의 유효 크기를 크게 증가시키는 효과를 유발한다.

[0053] ROB(300)는 최종 쓰기 지시자(또는 분기 지시자)에 대응하는 다수의 명령어 각각을 일괄적으로 ROB 엔트리로 할당하여 저장할 수 있으며, 할당된 ROB 엔트리를 블록 순서 큐(230)에서 인가되는 커밋 승인 신호에 따라 명령어 블록에 포함되는 명령어들에 대한 할당을 일괄적으로 해제할 수 있다. 즉 ROB(300)는 동일한 명령어 블록에 해당하는 명령어들을 일괄적으로 할당하거나 할당 해제할 수 있다.

[0054] 블록 설정부(210)가 다수의 명령어를 블록화하여 명령어 블록을 생성하고, ROB(300)는 생성된 명령어 블록 내의 최종 쓰기 지시자(또는 분기 지시자)에 대응하는 명령어를 일괄적으로 ROB 엔트리로 할당하거나 할당 해제하는 것은 명령어 블록 단위로 ROB(300)에 명령어를 저장 및 제거함으로써, ROB(300)에 모든 명령어가 ROB 엔트리로 할당되지 않더라도 코어의 구조적 상태를 유지할 수 있도록 하기 위함이다.

[0055] 그리고 블록 설정부(210)가 다수의 명령어 블록을 병합하거나 분할하는 것은 명령어 블록에 포함되는 명령어의 수가 증가될수록, 명령어 블록에 포함되는 전체 명령어의 개수 대비, 최종 쓰기 지시자에 대응하는 명령어의 개수가 더 줄어 들게 되어 ROB(300)에 저장되어야 하는 명령어의 개수를 더 줄일 수 있기 때문이다. 즉 ROB(300)의 유효 크기를 더욱 증가시킬 수 있다.

[0056] 도 3을 참조하면, (a)는 각각 5개의 명령어((I_1 ~ I_5), (I_6 ~ I_{10}))를 포함하는 2개의 명령어 블록(B1, B2)에서 ROB 엔트리로 할당되는 명령어를 나타내고, (b)는 (a)에 도시된 2개의 명령어 블록(B1, B2)이 하나의 명령어 블록으로 병합된 경우에 ROB 엔트리로 할당되는 명령어를 나타낸다.

[0057] (a)에서는 제1 및 제2 명령어 블록(B1, B2) 각각에서 최종 쓰기 지시자와 분기 지시자에 대응하는 명령어들을 ROB 엔트리로 할당하므로, 제2 명령어(i2)와 제6 및 제7 명령어(i6, i7)의 세개의 명령어를 제외한 나머지 모든

명령어(i1, i3 ~ i5, i8 ~ i10)이 ROB 엔트리로 할당된다. 비록 제1 블록(B1)에서 제1 및 제3 명령어(i1, i3)의 제2 및 제3 레지스터(r2, r3)는 제2 블록(B2)의 제6 내지 제9 명령어(i6 ~ i9)에 의해 2회씩 덮어쓰기가 되지만, 제1 블록(B1)과 제2 블록(B2)으로 구분되어 있으므로, 제1 블록(B1)의 제1 및 제3 명령어(i1, i3)는 ROB 엔트리로서 ROB(300)에 할당되어야 한다.

[0058] 반면, (b)와 같이 2개의 명령어 블록(B1, B2)이 하나의 명령어 블록으로 병합된 경우에는 제1 및 제3 명령어(i1, i3)의 제2 및 제3 레지스터(r2, r3)가 제6 내지 제9 명령어(i6 ~ i9)에 의해 덮어쓰기되고, 최종적으로는 제8 및 제9 명령어(i8, i9)가 제2 및 제3 레지스터(r2, r3)에 쓰기 작업을 하므로, 제1, 제3, 제6 및 제7 명령어(i1, i3, i6, i7)이 모두 제외되고, 제8 및 제9 명령어(i8, i9)만이 ROB 엔트리로서 ROB(300)에 할당될 수 있다.

[0059] 즉 명령어 블록의 크기가 증가할수록 ROB(300)에 ROB 엔트리로서 할당되어야 하는 명령어의 개수가 더욱 줄어들게 되어 ROB(300)의 유효 크기를 증가시킬 수 있다.

[0060] 다만, 상기한 바와 같이 명령어 블록 내의 분기 명령의 분기 예측 오류나 예외 상황이 발생된 경우, 명령어 블록 전체에 대해 실행된 결과가 무효화되고 재실행되어야 하므로, 명령어 블록의 병합을 통해 명령어 블록의 크기를 증가시키게 되면 오히려 비효율적이게 되는 결과가 발생될 수 있다. 즉 프로그램에 포함된 모든 명령어를 하나의 명령어 블록으로 설정하면 ROB(300)에 할당되는 명령어 개수를 최대로 줄일 수 있으나, 정상적으로 실행되지 못하는 결과를 초래할 수 있다.

[0061] 이에 본 실시예에서는 프로그램 명령어가 일반적으로 반복 실행되므로, 이전 명령어 블록의 실행 결과를 기반으로 분기 명령어나 예외 상황을 모니터링하고, 모니터링 결과에 따라 명령어 블록을 병합 또는 분할하도록 함으로써 최적의 명령어 블록을 설정할 수 있도록 한다. 즉 코어의 실행 성능을 가능한 유지하면서 ROB(300)의 유효 크기를 최대화할 수 있도록 한다.

[0062] 도 4에서 (a)는 프로그램에서 분기 명령어를 기준으로 구분된 블록(B1 ~ B6)에 따른 제어 흐름 그래프를 나타낸다. (a)에 도시된 바와 같이, 프로그램에는 다수의 분기 명령어(B1 ~ B6)가 포함될 수 있으며, 다수의 분기 명령어(B1 ~ B6)는 다음 실행될 명령어에 대한 분기가 수행될 수 있다. (a)에서는 제1 분기 명령어(B1)가 낫테이큰(NT)되어 분기되고, 제3 분기 명령어(B3)에서는 테이큰(T)되어 분기되며, 제4 분기 명령어(B4)에서도 테이큰(T)되어 제6 분기 명령어(B6) 방향으로 분기되는 경로로 프로그램이 실행되는 경우를 나타내고 있다.

[0063] 그리고 블록 설정부(210)는 (b)의 좌측에 도시된 바와 같이, 분기 명령어(B1 ~ B6)를 기준으로 다수의 명령어 블록(iB1 ~ iB6)을 생성할 수 있다. 이때 다수의 명령어 블록(iB1 ~ iB6) 각각은 대응하는 하나의 분기 명령어(B1 ~ B6)가 포함된 기준 블록이며, 각 분기 명령어(B1 ~ B6)의 분기 경로는 미리 예측되어 있다.

[0064] 그리고 동일 분기 경로가 선택되어 동일한 명령어 블록에 대해 기지정된 기준 커밋 횟수(여기서는 일예로 5회) 이상 커밋 승인 신호가 인가되면, 분기 경로 상에서 서로 인접한 명령어 블록들을 서로 결합한다. (b)에서는 가운데 도시된 바와 같이, 제3 명령어 블록(iB3)과 제6 명령어 블록(iB3)이 각각 제1 명령어 블록(iB1)과 제4 명령어 블록(iB4)에 병합되었다. 그리고 이후에도 연속하여 동일 분기 경로로 커밋 승인 신호가 인가되면, (b)의 오른쪽에 도시된 바와 같이, 병합된 명령어 블록들(iB1, iB4)를 다시 하나의 명령어 블록(iB1)으로 병합할 수 있다.

[0065] 한편, 병합된 하나의 명령어 블록(iB1)을 반복하여 실행하는 동안 명령어 블록(iB1) 내의 특정 위치(여기서는 일예로 기존의 제4 명령어 블록(iB4)의 명령어)에서 기지정된 기준 오류 횟수(예를 들면 2회) 이상 실행 오류 신호가 인가되면, 오류가 발생된 위치의 명령어가 포함된 기준 블록(iB4) 이전 기준 블록(iB3)에서 병합된 명령어 블록(iB1)을 분할하고, 다른 분기 경로 상의 제5 명령어 블록(iB5)으로 분기 경로를 변경할 수 있다.

[0066] 도시하지 않았으나, 변경된 분기 경로에서 다시 기준 커밋 횟수 이상 커밋 승인 신호가 인가되면, 블록 설정부(210)는 제5 명령어 블록(iB5)을 병합된 명령어 블록(iB1)에 추가 병합할 수 있다.

[0067] 이러한 과정을 프로그램에서 요구되는 횟수동안 반복 실행함으로써, 블록 설정부(210)는 최적의 명령어 블록 조합을 획득할 수 있다.

[0068] 프로그램 명령어의 경우, 컴파일러가 컴파일을 수행하는 동안 다수의 명령어를 블록화 시키는 경우도 있으나, 컴파일러는 명령어를 실행하지 않기 때문에 실질적인 블록화가 수행될 수 없다. 즉 여러 분기 명령어나 예외 상황을 고려하여 블록화를 할 수 없어 최적의 명령어 블록을 획득할 수 없다는 한계가 있다. 그러나 상기한 바와 같이, 본 실시예에서는 프로그램 명령어가 반복 실행된다는 점에 기반하여, 이전 명령어 블록의 실행 결과를

기반으로 블록화를 수행하므로, 명령어 블록을 최적화할 수 있다.

[0069] PRF(400)은 연산 실행부(500)에서 연산이 완료된 명령어의 결과값을 저장하는 구조체로서, 비순차 실행 코어에서 연산 실행 시에 하나의 논리 레지스터에 다수의 물리 레지스터를 할당함으로써 동일한 논리 레지스터를 갱신하는 서로 다른 명령어 간의 비순차실행이 올바르게 이루어질 수 있도록 한다. PRF(400)은 디스패치부(150)가 논리 레지스터를 갱신하는 모든 명령어에 대하여 논리 레지스터를 물리 레지스터로 변환함으로써 할당한다. PRF(400)는 논리 레지스터를 갱신하는 명령어에 의해 물리 레지스터를 할당하며, 이후의 명령어에 의해 동일한 논리 레지스터가 갱신될 때까지 물리 레지스터를 할당 유지해야 한다.

[0070] 연산 실행부(500)는 디스패치부(150)로부터 이슈되는 디스패치 엔트리에 따라 연산 동작을 수행한다. 연산 실행부(500)는 디스패치 엔트리에 포함된 오피 코드와 물리 목적 레지스터 주소, 물리 소스 레지스터 주소 중 물리 소스 레지스터 주소에 따라 지정되는 PRF(400)의 레지스터 값을 읽어 피연산값을 획득하고, 획득된 피연산값을 오피 코드에 따라 연산하여 PRF(400)에서 물리 목적 레지스터 주소에 의해 지정되는 레지스터의 값을 할당하여 갱신한다.

[0071] 여기서는 설명의 편의를 위하여 연산 실행부(500)를 단일 블록으로 도시하였으나, 연산 실행부(500)는 다수의 연산 기능부로 구성될 수 있으며, 다수의 연산 기능부 각각은 오피 코드에 따라 서로 다른 기능을 수행하도록 구성될 수 있다. 연산 실행부(500)는 파이프 라인 기법에 따라 다수의 연산을 병렬로 수행할 수 있다.

[0072] 결과적으로 도 1에서 블록화부(200)는 리오더 버퍼 관리 장치로서 명령어 이슈부(100)로 인가된 다수의 명령어에 대해 블록화를 수행하여 다수의 명령어 블록을 생성하고, 다수의 명령어 블록 각각에 포함된 다수의 명령어 중 최종 쓰기 작업 또는 분기 작업을 수행하는 명령어만이 리오더 버퍼(300)의 리오더 엔트리로 할당되어 저장되도록 하여 리오더 버퍼(300)에 저장되어야 하는 명령어의 수를 줄임으로써, 리오더 버퍼(300)의 물리적 용량을 증가시키지 않고서도 용량을 증대시킨 것과 유사한 효과를 획득할 수 있다. 즉 리오더 버퍼(300)의 유효 크기를 증가시킨다.

[0073] 또한 리오더 버퍼 관리 장치는 생성된 명령어 블록을 프로그램이 실행되는 실행 시간 동안 기지정된 기준에 따라 병합 및 분할하여 최적의 명령어 블록을 탐색함으로써, 리오더 버퍼(300)에 저장되어야 하는 명령어의 개수를 더욱 줄일 수 있다.

[0074] 도 5는 본 발명의 일 실시예에 따른 비순차 실행 코어의 리오더 버퍼 관리 방법을 나타낸다.

[0075] 도 1 내지 도 4를 참조하여, 도 5의 리오더 버퍼 관리 방법을 설명하면, 우선 미리 작성된 프로그램에 따른 다수의 명령어가 인가된다(S11). 인가된 다수의 명령어는 명령어 이슈부(100)에 의해 페치되고 디코딩 및 리네이밍된 후 명령어에 대응하는 연산이 연산 실행부(500)에 의해 수행되도록 디스패치된다.

[0076] 그리고 명령어가 디스패치되는 동안 리오더 버퍼 관리 장치인 블록화부(200)는 인가된 명령어가 포함된 대응하는 명령어 블록이 기작성된 블록 히스토리 테이블(BLHT)에 존재하는지 판별한다(S12).

[0077] 만일 블록 히스토리 테이블(BLHT)내에 대응하는 명령어 블록이 존재하지 않으면, 인가되는 다수의 명령어를 분기 명령을 기준으로 구분하여 명령어 블록을 생성한다(S13). 여기서 생성된 명령어 블록을 기본 블록이라 할 수 있다.

[0078] 그리고 생성된 명령어 블록에 대한 정보를 획득하여, 블록 히스토리 테이블(Block History Table: 이하 BLHT)의 엔트리로 할당하여 저장한다(S14). 여기서 BLHT 엔트리에는 명령어 블록에 포함되는 다수의 명령어 중 첫번째 명령어에 대한 메모리 주소를 나타내는 블록 정의자와 명령어 블록 내부의 다수의 명령어 중 최종 쓰기 작업을 수행하는 명령어를 지시하는 최종 쓰기 지시자 및 블록 내부의 분기 명령어와 분기 경로를 나타내는 분기 지시자가 포함될 수 있다.

[0079] 여기서 명령어에 대한 메모리 주소는 PC로부터 인가될 수 있다.

[0080] 그리고 BLHT 엔트리를 기반으로 명령어 블록에 포함되는 다수의 명령어 중 최종 쓰기 지시자 및 분기 지시자에 대응하는 명령어들을 추출하여, 각각 리오더 버퍼(ROB)의 엔트리로 할당한다(S15).

[0081] 반면, 블록 히스토리 테이블(BLHT) 내의 BLHT 엔트리에서 인가된 명령어가 포함된 대응하는 BLHT 엔트리가 탐지되면, 탐지된 BLHT 엔트리를 기반으로 명령어 블록에 포함되는 다수의 명령어 중 최종 쓰기 지시자 및 분기 지시자에 대응하는 명령어들을 추출하여, 각각 리오더 버퍼(ROB)의 ROB 엔트리로 할당한다(S15).

[0082] 이후 연산 실행부(500)에서 인가된 다수의 명령어에 대응하는 연산의 실행 상태를 모니터링하여, 연산이 정상적

으로 수행되는지 판별한다(S16). 만일 연산이 정상적으로 수행되면, 명령어 블록에 대응하여 리오더 버퍼(ROB)에 할당된 ROB 엔트리에 대한 커밋을 승인한다(S17). 그리고 해당 명령어 블록에 대해 연속하여 커밋 승인된 횟수가 기지정된 기준 커밋 횟수 이상인지 판별한다(S18). 만일 커밋 승인된 횟수가 기준 커밋 횟수 이상이면, 커밋 승인된 명령어 블록과 인접한 명령어 블록을 병합한다(S19). 그리고 병합된 명령어 블록에 대해 다시 정보를 획득하여, BLHT 엔트리로 재할당한다(S23).

[0083] 한편, 연산이 정상적으로 수행되지 않은 것으로 판별되면, 명령어 블록에서 오류가 발생된 명령어를 판별한다(S20). 그리고 해당 명령어 블록에 대해 연속하여 실행 오류가 발생된 횟수가 기지정된 기준 오류 횟수 이상인지 판별한다(S21). 만일 실행 오류 횟수가 기준 오류 횟수 이상이면, 오류가 발생된 명령어가 포함된 기준 블록을 확인하고, 확인된 기준 블록을 명령어 블록에서 분할하고, 이전 기준 블록에서 분기 명령어에서의 분기 경로를 분할된 명령어 블록과 다른 경로로 전환한다(S22). 그리고 분할된 명령어 블록에 대해 다시 정보를 획득하여, BLHT 엔트리로 재할당한다(S23).

[0084] 이후 프로그램이 수행 완료되었는지 여부를 판별한다(S24). 만일 프로그램이 수행 완료되지 않은 것으로 판별되면, 다시 프로그램에 따른 명령어를 인가받는다(S11).

[0085] 본 발명에 따른 방법은 컴퓨터에서 실행시키기 위한 매체에 저장된 컴퓨터 프로그램으로 구현될 수 있다. 여기서 컴퓨터 판독가능 매체는 컴퓨터에 의해 액세스 될 수 있는 임의의 가용 매체일 수 있고, 또한 컴퓨터 저장 매체를 모두 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터와 같은 정보의 저장을 위한 임의의 방법 또는 기술로 구현된 휘발성 및 비휘발성, 분리형 및 비분리형 매체를 모두 포함하며, ROM(판독 전용 메모리), RAM(랜덤 액세스 메모리), CD(컴팩트 디스크)-ROM, DVD(디지털 비디오 디스크)-ROM, 자기 테이프, 플로피 디스크, 광데이터 저장장치 등을 포함할 수 있다.

[0086] 본 발명은 도면에 도시된 실시예를 참고로 설명되었으나 이는 예시적인 것에 불과하며, 본 기술 분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시예가 가능하다는 점을 이해할 것이다.

[0087] 따라서, 본 발명의 진정한 기술적 보호 범위는 첨부된 청구범위의 기술적 사상에 의해 정해져야 할 것이다.

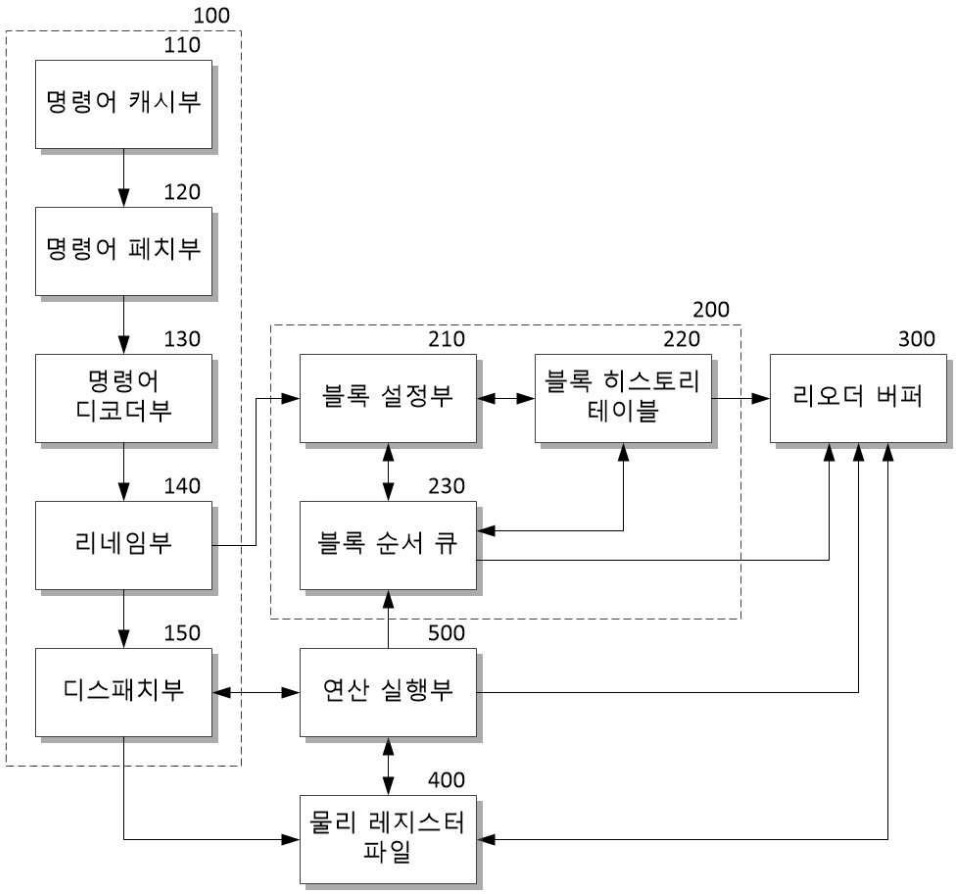
부호의 설명

[0088]

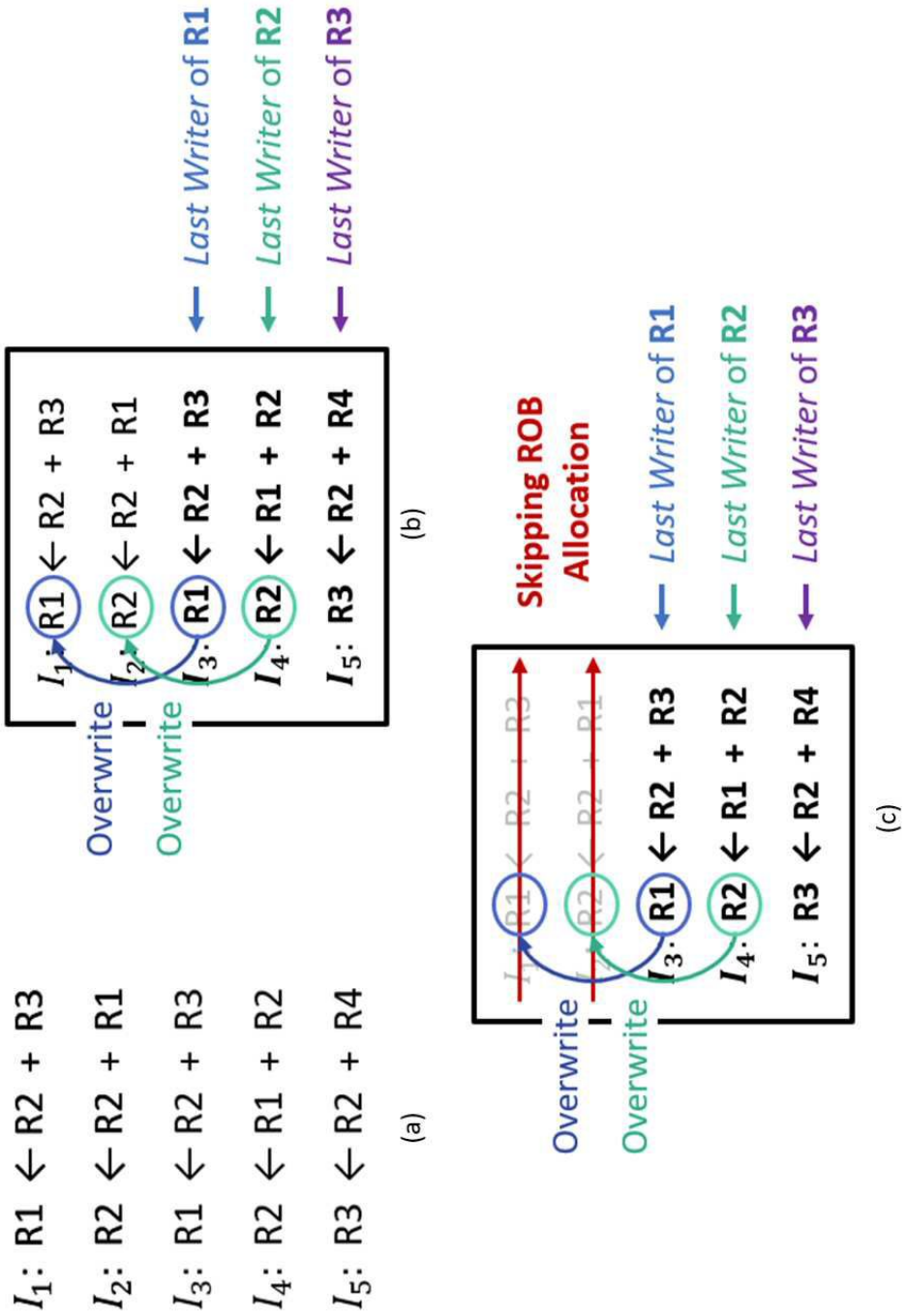
100: 명령어 이슈부
200: 명령어 블록화부
300: 리오더 버퍼
400: 물리 레지스터 파일
500: 연산 실행부
110: 명령어 캐시부
120: 명령어 페치부
130: 명령어 디코더부
140: 리네임부
150: 디스패치부
210: 블록 설정부
220: 블록 히스토리 테이블
230: 블록 순서 큐

도면

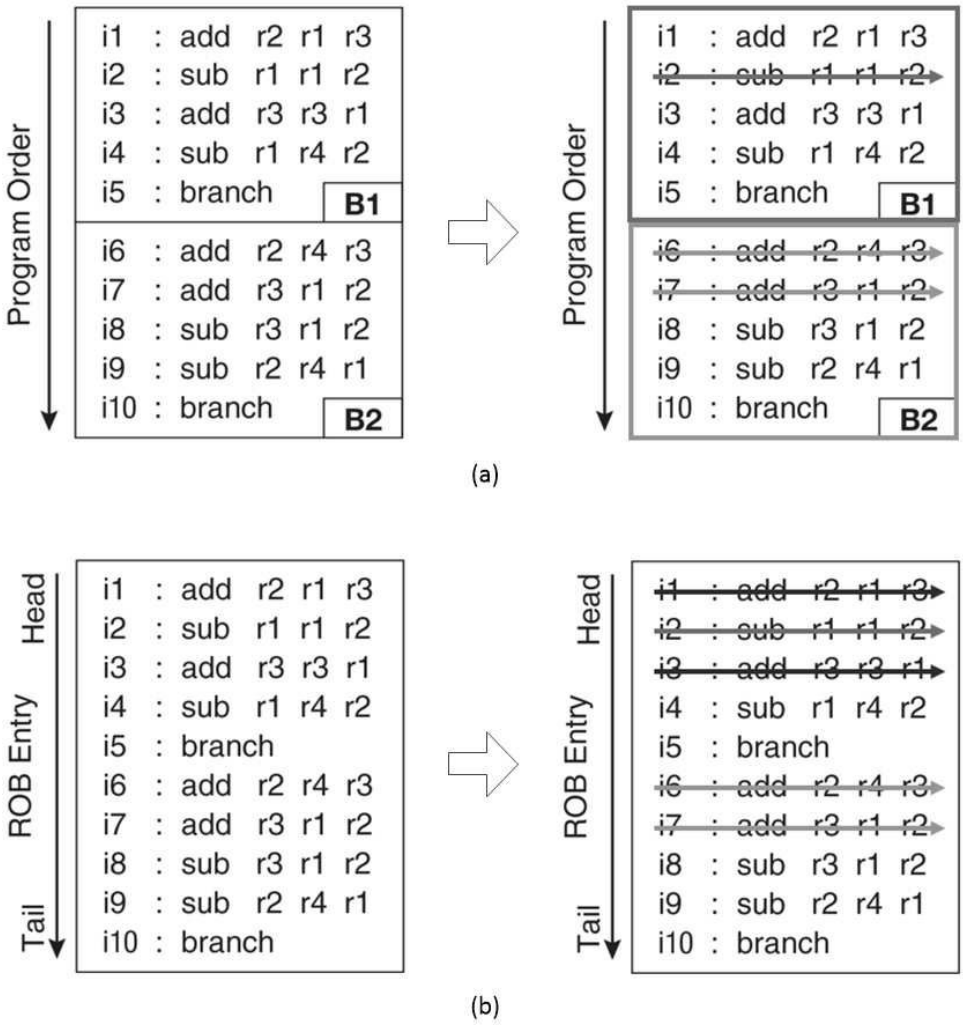
도면1


100
110
명령어 캐시부
120
명령어 페치부
130
명령어
디코더부
140
리네임부
150
디스패치부
200
210
블록 설정부
220
블록 히스토리
테이블
300
리오더 버퍼
230
블록 순서 큐
500
연산 실행부
400
물리 레지스터
파일

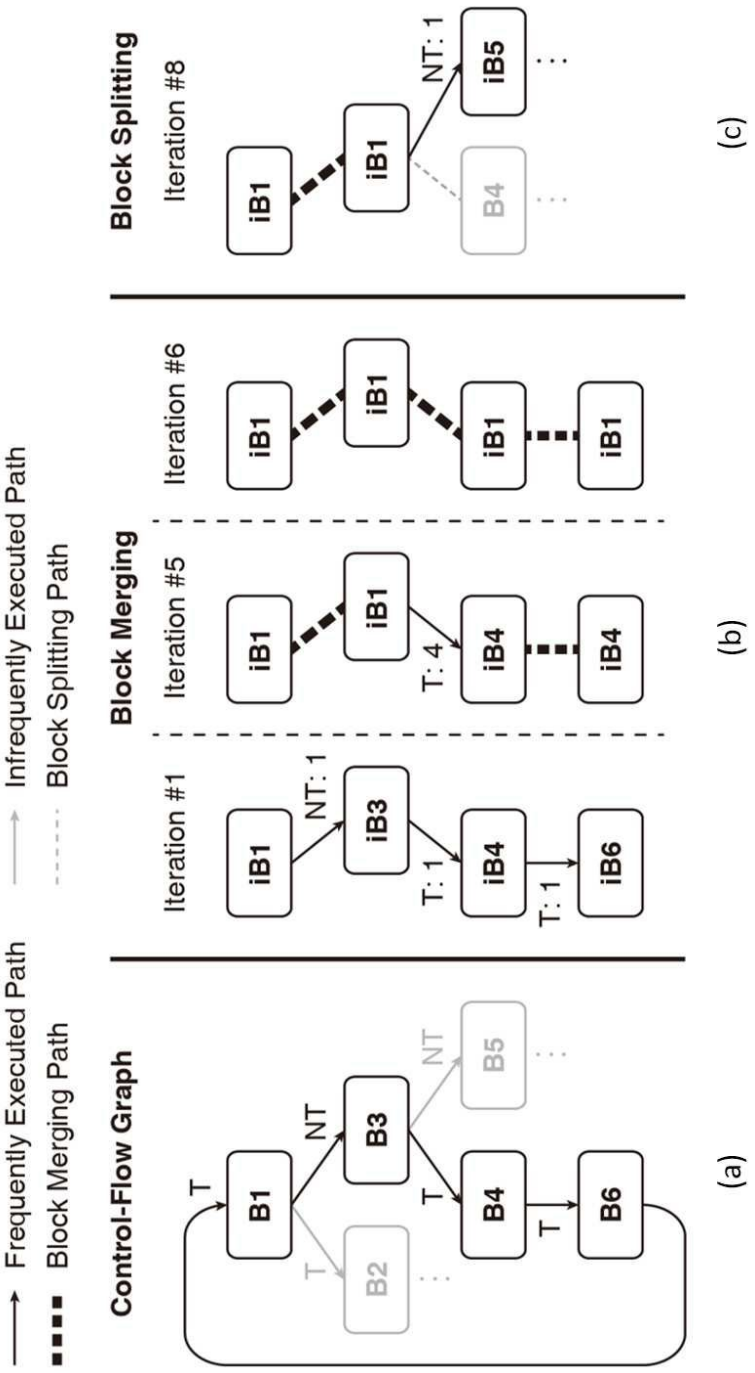
도면2


I1: R1 ← R2 + R3
I2: R2 ← R2 + R1
I3: R1 ← R2 + R3
I4: R2 ← R1 + R2
I5: R3 ← R2 + R4
(a)
Overwrite
Overwrite
Last Writer of R1
Last Writer of R2
Last Writer of R3
(b)
Skipping ROB Allocation
Overwrite
Overwrite
Last Writer of R1
Last Writer of R2
Last Writer of R3
(c)

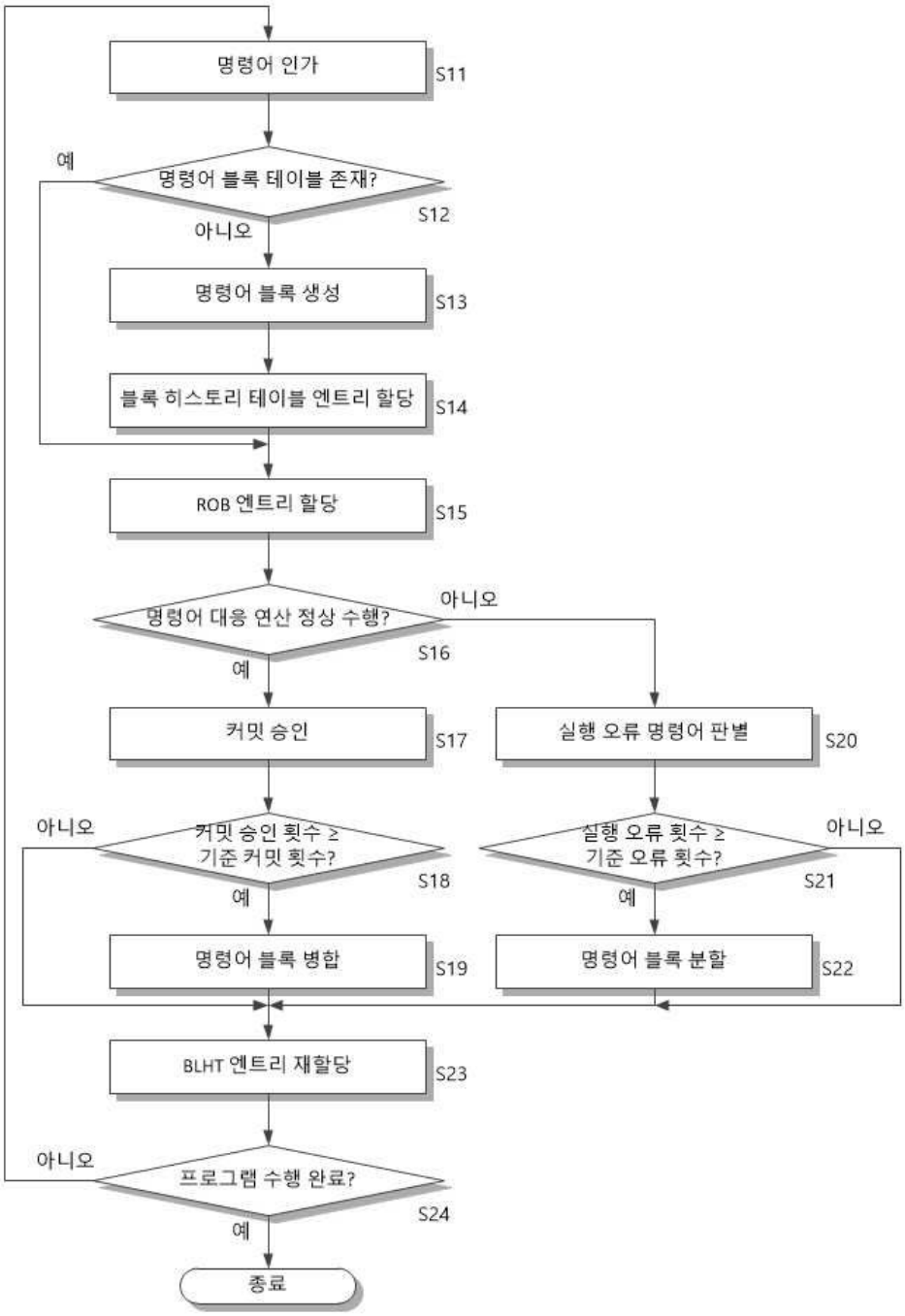
도면3


Program Order
i1 : add r2 r1 r3
i2 : sub r1 r1 r2
i3 : add r3 r3 r1
i4 : sub r1 r4 r2
i5 : branch
B1
i6 : add r2 r4 r3
i7 : add r3 r1 r2
i8 : sub r3 r1 r2
i9 : sub r2 r4 r1
i10 : branch
B2
(a)
Head
ROB Entry
Tail
i1 : add r2 r1 r3
i2 : sub r1 r1 r2
i3 : add r3 r3 r1
i4 : sub r1 r4 r2
i5 : branch
i6 : add r2 r4 r3
i7 : add r3 r1 r2
i8 : sub r3 r1 r2
i9 : sub r2 r4 r1
i10 : branch
(b)

도면4


Frequently Executed Path
Block Merging Path
Infrequently Executed Path
Block Splitting Path
Control-Flow Graph
B1
B2
B3
B4
B5
B6
T
NT
Block Merging
Iteration #1
Iteration #5
Iteration #6
iB1
iB3
iB4
iB6
NT: 1
T: 1
T: 4
Block Splitting
Iteration #8
NT: 1
(a)
(b)
(c)

도면5


명령어 인가
S11
예
명령어 블록 테이블 존재?
S12
아니오
명령어 블록 생성
S13
블록 히스토리 테이블 엔트리 할당
S14
ROB 엔트리 할당
S15
명령어 대응 연산 정상 수행?
아니오
S16
예
커밋 승인
S17
실행 오류 명령어 판별
S20
아니오
커밋 승인 횟수 ≥
기준 커밋 횟수?
S18
예
실행 오류 횟수 ≥
기준 오류 횟수?
아니오
S21
예
명령어 블록 병합
S19
명령어 블록 분할
S22
BLHT 엔트리 재할당
S23
아니오
프로그램 수행 완료?
S24
예
종료