(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2008-0098683
(43) 공개일자 2008년11월11일

(51) Int. Cl.
G06F 11/14 (2006.01) *G06F 9/30* (2006.01)
G06F 11/07 (2006.01)
(21) 출원번호 10-2008-7023877
(22) 출원일자 2008년09월29일
심사청구일자 2008년09월29일
번역문제출일자 2008년09월29일
(86) 국제출원번호 PCT/ES2006/070041
국제출원일자 2006년03월31일
(87) 국제공개번호 WO 2007/113346
국제공개일자 2007년10월11일

(71) 출원인
인텔 코오퍼레이션
미합중국 캘리포니아 산타클라라 미션 칼리지 블러바드 2200
(72) 발명자
베라, 사비어
스페인 이-08028 바르셀로나 2-2 87-89비 그란 비아 찰스 3세
에르긴, 오구즈
스페인 이-08034 바르셀로나 토레 기로나 파쎄이 그 델스 틸러스 19 레지던시아 유니버시타리아 에이103
(뒷면에 계속)
(74) 대리인
양영준, 백만기

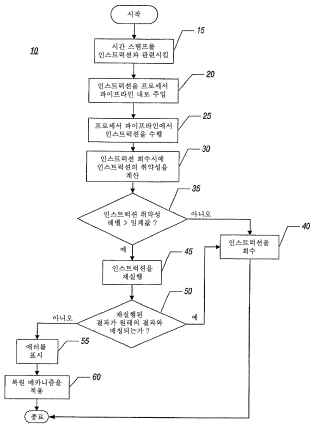
전체 청구항 수 : 총 29 항

(54) 새로운 선택적 구현에 의한 일시적 에러의 검출

(57) 요 약

일실시예에서, 본 발명은 프로세서에 의해 구현된 인스트럭션에 대한 취약성의 정도 또는 레벨을 결정하고, 취약성의 레벨이 소정의 임계값보다 큰 경우에 인스트럭션을 재구현하는 방법을 포함한다. 취약성의 레벨은 인스트럭션이 프로세서 내에 존재하는 동안 인스트럭션에 대한 논리 에러 가능성에 대응할 수 있다. 다른 실시예들이 기술되고 청구된다.

대 표 도 - 도1



(72) 발명자

언살, 오스만

스페인 이-08019 바르셀로나 피티에이. 1 피2 펠라이레스 1

아벨라, 자움

스페인 이-08034 바르셀로나 6-3 11 레이나 엘리센다 데 몬트카다

곤잘레즈, 안토니오

스페인 이-08028 바르셀로나 1에이 4오 11 조안 구엘

특허청구의 범위

청구항 1

프로세서에서 실행된 인스트럭션(instruction)에 대한 취약성 레벨(vulnerability level)을 결정하는 단계 - 상기 취약성 레벨은 상기 인스트럭션에 대한 소프트 에러 가능성에 대응함 - ; 및

상기 취약성 레벨이 임계값보다 큰 경우 상기 인스트럭션을 재실행하는 단계

를 포함하는 방법.

청구항 2

제1항에 있어서,

상기 인스트럭션이 실행되었던 프로세서의 제1 실행 유닛이 아닌, 상기 프로세서의 다른 실행 유닛에서 상기 인스트럭션을 재실행하는 단계를 더 포함하는 방법.

청구항 3

제1항에 있어서,

상기 취약성 레벨이 상기 임계값보다 작은 경우 상기 인스트럭션을 재실행하지 않는 단계를 더 포함하는 방법.

청구항 4

제1항에 있어서,

상기 인스트럭션에 의해 점유된 상기 프로세서의 영역 및 상기 인스트럭션의 수명과 관련된 시간 기간 중 적어도 하나에 근거하여 상기 취약성 레벨을 결정하는 단계를 더 포함하는 방법.

청구항 5

제4항에 있어서,

상기 인스트럭션과 관련된 시간 스탬프를 이용하여 상기 시간 기간을 결정하는 단계를 더 포함하는 방법.

청구항 6

제1항에 있어서,

상기 재실행된 인스트럭션의 결과가 상기 인스트럭션의 결과와 매칭되는지를 결정하는 단계; 및

매칭되지 않는 경우 상기 프로세서를 플러싱(flushing)하는 단계를 더 포함하는 방법.

청구항 7

적어도 하나의 레지스터 파일 및 상기 적어도 하나의 레지스터 파일에 연결된 적어도 하나의 실행 유닛; 및

상기 실행 유닛에 의해 실행된 인스트럭션들의 소프트 에러에 대한 취약성을 결정하는 인스트럭션 검증 유닛

을 포함하는 장치.

청구항 8

제7항에 있어서,

상기 인스트럭션 검증 유닛은,

인스트럭션들 및 관련된 소스 데이터 태그들을 저장하는 버퍼; 및

상기 버퍼에 연결되어 취약한 인스트럭션들을 재실행하는 적어도 하나의 실행 유닛을 포함하는 장치.

청구항 9

제8항에 있어서,

상기 인스트럭션 검증 유닛은 상기 인스트럭션에 대한 영역 값 및 상기 인스트럭션에 대한 시간 값 중 적어도 하나에 근거하여 인스트럭션에 대한 취약성 레벨을 결정하는 로직(logic)을 더 포함하는 장치.

청구항 10

제9항에 있어서,

상기 로직은 상기 인스트럭션과 관련된 시간 스탬프에 근거하여 상기 시간 값을 결정하는 장치.

청구항 11

제9항에 있어서,

상기 로직은 상기 취약성 레벨이 임계값보다 큰 경우 재실행을 위해 상기 인스트럭션을 상기 적어도 하나의 실행 유닛에 제공하는 장치.

청구항 12

제11항에 있어서,

상기 임계값은 선택된 성능 매트릭(metric)에 근거하여 조절가능하고, 상기 임계값은 보다 높은 레벨의 성능에 대해 보다 높게 설정되는 장치.

청구항 13

제11항에 있어서,

상기 인스트럭션의 재실행이 상기 인스트럭션의 원래의 실행과 매칭되는 경우 상기 인스트럭션을 회수(retire)하는 인스트럭션 회수 유닛을 더 포함하는 장치.

청구항 14

인스트럭션들을 포함하는 머신-판독가능한 저장 매체를 포함하는 물품에 있어서,

상기 인스트럭션들은 머신에 의해 실행되는 경우 상기 머신으로 하여금,

원래의 결과를 얻기 위해 프로세서에서 인스트럭션을 실행하는 단계;

상기 인스트럭션이 소프트 에러에 대해 취약한 경우 재실행된 결과를 얻기 위해 상기 프로세서에서의 상기 인스트럭션을 재실행하는 단계; 및

상기 원래의 결과를 상기 재실행된 결과와 비교하는 단계

를 포함하는 방법을 수행하도록 하는, 머신-판독가능한 저장 매체를 포함하는 물품.

청구항 15

제14항에 있어서,

상기 방법은 인스트럭션 회수 단계에서 상기 인스트럭션을 회수하고, 상기 인스트럭션이 상기 소프트 에러에 대해 취약하지 않은 경우 상기 인스트럭션을 재실행하지 않는 단계를 더 포함하는, 머신-판독가능한 저장 매체를 포함하는 물품.

청구항 16

제14항에 있어서,

상기 방법은 상기 비교가 미스매칭을 나타내는 경우 상기 프로세서를 플러싱하는 단계를 더 포함하는, 머신-판독가능한 저장 매체를 포함하는 물품.

청구항 17

제14항에 있어서,

상기 방법은 상기 인스트럭션에 의해 소모된 영역 및 상기 프로세서에서의 상기 인스트럭션의 펜딩 시간(pending time) 중 적어도 하나에 근거하여, 상기 인스트럭션이 상기 소프트 에러에 대해 취약한지 여부를 결정하는 단계를 더 포함하는, 머신-판독가능한 저장 매체를 포함하는 물품.

청구항 18

제17항에 있어서,

상기 방법은,

상기 영역 및 상기 펜딩 시간을 이용하여 취약성 값을 계산하는 단계; 및

상기 취약성 값과 임계값을 비교하고, 상기 비교에 근거하여 상기 인스트럭션을 재실행할지 여부를 결정하는 단계를 더 포함하는, 머신-판독가능한 저장 매체를 포함하는 물품.

청구항 19

인스트럭션을 실행하는 적어도 하나의 실행 유닛 및 상기 인스트럭션이 소프트 에러에 대해 취약한 경우 상기 인스트럭션을 재실행하는 중복 실행 유닛을 포함하는 프로세서; 및

상기 프로세서에 연결된 DRAM(dynamic random access memory)

을 포함하는 시스템.

청구항 20

제19항에 있어서,

상기 프로세서는 상기 소프트 에러에 대한 상기 인스트럭션의 취약성을 정량화하는 인스트럭션 검증기를 더 포함하는 시스템.

청구항 21

제20항에 있어서,

상기 인스트럭션 검증기는 상기 프로세서에서의 상기 인스트럭션의 크기 및 상기 인스트럭션의 수명 중 적어도 하나에 근거하여 상기 인스트럭션을 재실행할지의 여부를 결정하는 시스템.

청구항 22

제21항에 있어서,

상기 인스트럭션 검증기는 상기 인스트럭션의 상기 크기 및 수명에 근거하여 상기 인스트럭션의 취약성 값을 임계값과 비교하고, 상기 취약성 값이 상기 임계값보다 큰 경우 상기 재실행을 트리거링하는 시스템.

청구항 23

제22항에 있어서,

상기 임계값은 적응적 임계값이며, 상기 시스템은 선택된 성능 레벨에 근거하여 상기 임계값을 조절하는 시스템.

청구항 24

제22항에 있어서,

상기 시스템은 상기 프로세서가 인스트럭션 취약성 감소 기술을 구현하는 경우에 상기 임계값을 감소시키는 시스템.

청구항 25

제20항에 있어서,

상기 인스트럭션 검증기는 상기 인스트럭션의 회수 단계에서 상기 인스트럭션을 재실행할지의 여부를 결정하는 시스템.

청구항 26

제20항에 있어서,

상기 인스트럭션 검증기는 상기 프로세서에서 실행된 인스트럭션들에 대응하는 엔트리들을 저장하는 버퍼를 포함하는 시스템.

청구항 27

제26항에 있어서,

상기 엔트리들은 대응하는 상기 인스트럭션을 상기 프로세서내로 삽입하는 시간에 대응하는 시간 스탬프 필드를 포함하는 시스템.

청구항 28

제27항에 있어서,

상기 인스트럭션 검증기는 상기 시간 스탬프 필드를 이용하여 상기 인스트럭션을 재실행할지의 여부를 결정하는 시스템.

청구항 29

제19항에 있어서,

상기 시스템은 상기 재실행된 인스트럭션의 결과가 상기 인스트럭션의 결과와 매칭되지 않는 경우 상기 프로세서의 적어도 일부분을 플러싱하는 시스템.

명 세 서

기 술 분 야

<1> 본 발명의 실시예들은 반도체 디바이스에서의 에러 검출에 관한 것으로서, 특히, 프로세서에서의 에러 검출에 관한 것이다.

배 경 기 술

<2> 종종 소프트 에러(soft error)들이라고 지칭되는 일시적인 에러들은, 프로세서에서의 증가되는 에러 소스이다. 디바이스의 감소된 크기 및 디바이스가 동작하는 감소된 전압으로 인해, 이들 디바이스는 우주 입자 충격(cosmic particle strikes) 및 파라미터 변화에 더욱 취약하다. 그러한 이벤트들은, 임의적으로 발생되며, 프로세서의 적절한 실행에 영향을 미칠 수 있는 일시적인 에러를 초래할 수 있다. 반도체 제조 기술의 각 세대에 따라, 소프트 에러에 대한 민감성이 증가할 것으로 예상된다.

<3> 소정의 메카니즘들이 소프트 에러 정정을 시도하는데 이용되어 왔다. 전형적으로, 이들 방안들은 데이터에 대한 중복 동작들을 위한 중복 경로들을 제공하는 것을 포함한다. 그러나, 그러한 중복 경로들은 프로세서의 크기 및 전력 소모를 크게 증가시켜, 성능 저하를 초래한다. 더욱이, 일부 방안들은 에러를 검출하기 위해 동시 멀티스레딩(simultaneous multithreading; SMT)을 이용한다. 그러한 방안들에서, 처리는 2개의 분리된 실행 경로들(예를 들면, SMT 코어에서의 2개의 스레드)에 대해 스케쥴링된다. 그 다음, 결과적인 데이터는 동일성(identity) 확인을 위해 비교된다. 결과가 상이한 경우, 이것은 소프트 에러의 표시이며, 그러한 에러가 검출된다. 그러나, 일부 하드웨어의 경우 다른 처리들을 실행하는 대신에 에러 검출에 집중하며, 결과 비교 및 스레드 조정을 지원하는데 있어 복잡성이 존재하기 때문에, 성능의 저하가 현저하다.

발명의 상세한 설명

<9> 다양한 실시예들에서, 프로세서에서의 소프트 에러가 검출될 수 있으며, 그러한 에러를 정정하기 위해 적절한 조치를 취할 수 있다. 그러한 소프트 에러 검출은 최소의 복잡도 또는 추가 전력 소모로 수행될 수 있다. 더욱이, 실시예들은 현존하는 프로세서 구조를 이용하여 에러 검출을 수행할 수 있다. 대안적으로, 소프트 에로

검출을 수행하기 위해, 최소량의 추가적인 하드웨어가 구현될 수 있다.

<10> 소프트 에러 검출을 수행하기 위해, 상이한 파라미터들에 근거하여 프로세서 파이프라인에서의 인스트럭션들이 선택적으로 복제되거나 또는 재실행될 수 있다. 예를 들어, (예컨대, 프로세서에서의 크기 및/또는 시간 길이에 근거하여) 소프트 에러에 특히 좌우될 것 같은 인스트럭션들만이 선택적으로 복제될 수 있다. 이러한 방식으로, 성능에 최소한으로 영향을 주면서, 많은 양의 소프트 에러를 검출할 수 있다.

<11> 본 발명의 실시예에 따른 소프트 에러 검출은 상이한 방식들로 구현될 수 있다. 몇몇 실시예에서, 현존하는 프로세서 구조를 이용하여, 에러 검출을 위한 하나 이상의 알고리즘을 통해 소프트 에러 검출을 수행할 수 있다. 다른 실시예에서, 소프트 에러 검출을 처리하기 위해, 추가적인 제어기들, 로직(logic)들 및/또는 기능 유닛들이 프로세서에 제공될 수 있다. 하이 레벨에서, 소프트 에러 검출은 소프트 에러에 특히 취약한 프로세서 파이프라인에서의 인스트럭션들을 식별하고, 그러한 인스트럭션들을 재실행함으로써 구현될 수 있다. 원래의 인스트럭션 및 복제된 인스트럭션의 결과들이 매칭되는 경우, 소프트 에러는 표시되지 않는다. 만약, 그렇지 않고 결과들이 상이한 경우, 소프트 에러가 표시되며, 에러를 해결하기 위해 복원 메카니즘이 적용될 수 있다.

<12> 인스트럭션 취약성은 대부분, 인스트럭션이 프로세서에서 이용하는 영역 및 인스트럭션이 프로세서 내에서 소모하는 시간에 의존한다. 예를 들어, 많은 인스트럭션들이 수행(commitment) 이전에 프로세서에서 많은 수의 사이클을 소모하는 반면, 다른 인스트럭션들이 단일의 사이클 동안 지연없이 파이프라인을 이동한다. 더욱이, 모든 인스트럭션들이 동일한 하드웨어 자원들을 이용하는 것은 아니다. 소프트 에러 검출 커버리지를 위해, 가장 취약한 인스트럭션들은, 성능에 최소한으로 영향을 미치면서, 최대한으로 가능한 에러 커버리지를 제공하도록 복제될 수 있다.

<13> 일실시예에서, 취약한 인스트럭션들은 인스트럭션이 파이프라인을 떠나기 전의 인스트럭션 수행시에(즉, 인스트럭션 회수(retirement)시에) 복제될 수 있다. 이러한 실시예에서, ALU(arithmetic logic unit)들의 세트가 프로세서 내에 포함되어, 예를 들면, 인스트럭션들의 재실행에 의해 인스트럭션들이 ROB(reorder buffer)의 최상부에 도달할 때, 취약한 인스트럭션의 출력들을 검증(validate)할 수 있다.

<14> 상이한 인스트럭션들이 그들의 수명 동안 프로세서에서 상이한 양의 저장부를 점유하고, 상이한 양의 시간을 소모하기 때문에, 스프트 에러에 대한 각 인스트럭션의 취약성은 상이하다. 그러한 보다 취약한 인스트럭션들을 식별함으로써, 최소 레벨의 인스트럭션 복제로, 높은 커버리지의 소프트 에러 검출을 달성할 수 있다. 이러한 방식으로, 최소의 하드웨어 자원 및 전력 소모를 이용하여, 성능에 최소한의 영향을 미치면서, 최대량의 에러 검출 커버리지를 달성할 수 있다. 다양한 실시예들에서, 인스트럭션의 소프트 에러 취약성은 인스트럭션 타입(예를 들면, 로드, 저장, 브랜치, 산술), 프로세서 (또는, 특정 프로세서 구성요소) 내에서 인스트럭션에 의해 소모된 시간, 및 인스트럭션의 다른 특성들(예를 들면, 소스 데이터가 준비중이고, 즉각적인 필드가 좁기 때문에, 감소된 수의 비트를 가짐)에 의존한다.

<15> 모든 인스트럭션들이 프로세서 구성요소 내에서 동일한 공간을 점유하는 것은 아니다. 예를 들어, 로드 및 저장 인스트럭션들은 그들의 엔티티 내에 메모리 순서 버퍼 인덱스(memory order buffer index)를 저장할 수 있지만, 다른 인스트럭션들은 이러한 필드를 이용하지 않는다. 마찬가지로, 저장 및 브랜치 인스트럭션들은 어떠한 결과도 생성하지 않고, 따라서, 그들에게 지정된 어떠한 목적지 레지스터도 갖지 않기 때문에, 목적지 레지스터 인덱스를 저장하는 이슈 큐 필드(issue queue field)를 이용하지 않는다. 이용되지 않은 이들 비트는 입자 공격에 대해 취약하지 않으므로, 인스트럭션 취약성을 감소시킨다. 또한, 이슈 큐 엔트리 내의 일부 비트들의 취약성 상태는 수퍼스칼라(superscalar) 파이프라인의 입력 변화 또는 역학 관계(dynamics)에 의존적일 수 있다. 인스트럭션이 이슈 큐로 디스패치될 때 인스트럭션의 소스들이 준비되어 있다면, 이슈 큐 내의 소스 태그 필드들은 취약하다. 유사하게, 즉각적 필드에 대해 좁은 오퍼랜드 식별 기법을 이용하는 것은 인스트럭션의 즉각적 필드의 큰 부분이 취약하지 않게 하여, 인스트럭션의 전체 취약성을 감소시킨다.

실 시 예

<16> 이제, 도 1을 참조하면, 본 발명의 일실시예에 다른 방법의 흐름도가 도시되어 있다. 도 1에 도시된 바와 같이, 방법(10)은 시간 스탬프를 인스트럭션과 관련시킴으로써 시작될 수 있다(블록 15). 예를 들어, 프로세서의 전단은 입력 인스트럭션이 ROB와 같은 버퍼 내에 배치됨에 따라, 시간 스탬프를 입력 인스트럭션과 관련시킬 수 있다. 다양한 실시예들에서, 에러 검출이 다른 실시예들에서 상이한 인스트럭션 입자도 레벨들(instruction granularity levels)로 구현될 수 있지만, 인스트럭션들은 마이크로오퍼레이션들(microoperations)(μops)에 대응할 수 있다. 본 명세서에서는 특정 프로세서 구조에 대해 기술되었지만, 본 발명의 영역은 그렇게 제한되

지 않으며, 소프트 에러 검출은 다른 위치들에서 구현될 수 있음을 이해할 것이다. 더욱이, 인스트럭션이 프로세서의 전단으로 들어감에 따라, 시간 스탬프를 인스트럭션과 관련시키는 것으로서 기술되었지만, 그 대신에, 시간 스탬프는 프로세서 파이프라인에서의 다른 포인트들에서의 인스트럭션과 관련될 수 있다.

<17> 도 1을 여전히 참조하면, 다음, 인스트럭션이 프로세서 파이프라인 내로 주입된다(블록 20). 따라서, 인스트럭션이 실행을 위해 스케쥴링될 때, 인스트럭션은 프로세서 파이프라인에서 수행될 수 있다(블록 25). 실행후에, 인스트럭션 회수시에 인스트럭션의 취약성이 계산될 수 있다(블록 30). 예를 들어, ROB에서의 인스트럭션의 상태는, 그것이 실행을 종료할 때, 수행할 준비가 된 것으로서 설정되며, 그것이 회수될 (즉, 인스트럭션에서 시간상 가장 오래된) ROB의 최상부에 도달할 때까지 대기한다. 그 때, 소프트 에러에 대한 인스트럭션의 취약성이 계산될 수 있다. 그러한 계산은 상이한 실시예들에서 많은 상이한 형태들을 취할 수 있다. 계산은 인스트럭션이 프로세서 내에 존재하는 시간의 길이 뿐만 아니라, 인스트럭션에 의해 소비된 프로세서의 영역 둘다를 고려할 수 있다. 따라서, (인스트럭션, 소스 정보 등의 다양한 필드들을 포함하는) 인스트럭션 폭 뿐만 아니라 시간 스탬프에 근거한 정보가 고려될 수 있다.

<18> 다음, 결정된 인스트럭션 취약성 레벨이 임계값보다 큰지의 여부가 결정될 수 있다(다이아몬드 35). 이러한 임계값은 사용자 설정될 수 있고, 몇몇 실시예에서에서, 프로세서에 대한 원하는 성능 레벨에 근거한 적응적 임계값일 수 있다. 인스트럭션 취약성 레벨이 임계값보다 낮다면, 제어는 블록(40)으로 전달되며, 여기서 인스트럭션이 회수될 수 있다. 따라서, 방법(10)은 종료된다.

<19> 그 대신에, 다이아몬드(35)에서, 인스트럭션 취약성 레벨이 임계값보다 큰 것으로 결정된다면, 제어는 블록(45)으로 전달된다. 여기서, 인스트럭션은 재실행될 수 있다(블록 45). 몇몇 실시예에서, 인스트럭션은 복제되어, 동일한 프로세서 파이프라인에서 재실행될 수 있지만, 다양한 실시예에서, 하나 이상의 추가적인 기능 유닛들이 재실행의 수행을 위해 제공될 수 있다.

<20> 재실행후에, 원래의 결과는 재실행된 결과와 비교되어, 결과들의 매칭 여부가 결정될 수 있다(다이아몬드 50). 만약 매칭된다면, 이것은 소프트 에러가 존재하지 않는다는 표시이며, 따라서, 제어는 (전술한) 블록(40)으로 전달되며, 여기서 인스트럭션이 회수된다. 만약, 그 대신에, 다이아몬드(50)에서, 결과들이 상이하다고 결정된다면, 제어는 블록(55)으로 전달된다. 따라서, 블록(55)에서는, 소프트 에러가 표시된다. 그러한 표시는 프로세서 또는 다른 그러한 위치의 소정의 제어 로직에 대한 신호를 포함하는 많은 상이한 형태들을 취할 수 있다. 표시된 에러에 근거하여, 적절한 복원 메카니즘이 적용될 수 있다(블록 60). 복원 메카니즘은 상이한 실시예들에 있어서 변할 수 있으며, 인스트럭션의 재실행, 다양한 프로세서 자원들의 일부 또는 전부의 플러싱(flushing), 또는 다른 그러한 복원 메카니즘을 포함할 수 있다. 도 1의 실시예에서의 이러한 특정 구현으로 기술되었지만, 본 발명의 영역은 그렇게 제한되지 않음을 이해할 것이다.

<21> 전술한 바와 같이, 실시예들은 많은 상이한 프로세서 아키텍쳐들로 구현될 수 있다. 예시를 위해, 도 2는 본 발명의 일실시예에 따른 일반적 프로세서 아키텍쳐의 블록도를 도시한다. 도 2에 도시된 바와 같이, 프로세서(100)는 아웃-오브-오더(out-of-order) 프로세서일 수 있다. 그러나, 본 발명의 영역은 그렇게 제한되지 않으며, 다른 실시예들이 인-오더(in-order) 머신에서 구현될 수 있다. 프로세서(100)는 인스트럭션 정보를 수신하고, 실행을 위해 그러한 정보를 하나 이상의 마이크로오퍼레이션들로 디코딩할 수 있는 전단부(front end)(110)를 포함한다. 전단부(110)는 다수의 실행 유닛들(130) 중 선택된 하나에 대해 실행을 위한 인스트럭션들을 스케쥴링할 수 있는 인스트럭션 스케쥴링 유닛(120)에 연결된다. 그러한 유닛들은 변할 수 있지만, 소정의 실시예에서, 정수, 부동점(floating-point), SIMD(single instruction multiple data), 어드레스 생성 유닛들 및 다른 그러한 실행 유닛들이 제공될 수 있다. 더욱이, 몇몇 실시예에서, 하나 이상의 추가적인 중복 실행 유닛들을 제공하여, 본 발명의 실시예에 따른 소프트 에러 검출을 수행할 수 있다.

<22> 도 2를 여전히 참조하면, 인스트럭션들이 선택된 실행 유닛에서 실행되었을 때, 인스트럭션들은 인스트럭션 회수/검증 유닛(140)에 제공될 수 있다. 유닛(140)은 프로세서 파이프라인 내에서 상이한 순서들로 수행된 인스트럭션들을, 프로그램 순서에 따라 인-오더 회수로 다시 회수하는데 이용될 수 있다. 유닛(140)은 본 발명의 실시예에 따른 소프트 에러 검출을 수앵하도록 더 적응될 수 있다. 특히, 회수시에, 주어진 임계값보다 큰 레벨에서 소프트 에러에 취약할 것으로 생각되는 각각의 인스트럭션이, 예를 들면, 실행 유닛들(130) 내의 추가적인 실행 유닛에서 재실행될 수 있다. 그러한 재실행의 결과에 근거하여, 원래의 인스트럭션의 결과가 확인되어 인스트럭션이 회수되거나 또는 소프트 에러가 표시되고, 적절한 복원 메카니즘이 구현된다. 도 2에서의 하이 레벨 아키텍쳐로 기술되었지만, 본 발명의 영역은 그렇게 제한되지 않으며, 특정한 변형들이 고려됨을 이해할 것이다.

<23> 이제, 도 3을 참조하면, 본 발명의 일실시예에 따른 프로세서의 보다 상세한 블록도가 도시되어 있다. 도 3에 도시된 바와 같이, 프로세서(200)는 인스트럭션들을 수행하기 위한 다양한 자원들을 포함할 수 있다. 도 3에 도시된 프로세서(200)는 단일 코어 프로세서에 대응하거나, 또는, 다른 실시예에서, 대안적으로 멀티코어 또는 다수코어 프로세서 중 하나의 코어일 수 있다.

<24> 도 3에 도시된 바와 같이, 프로세서(200)는 다양한 자원들을 포함하는 전단부(210)를 포함할 수 있다. 도 3의 실시예에서, 인스트럭션들을 취하여, 인스트럭션들 내의 논리적 레지스터를 프로세서의 레지스터 파일들 내의 보다 큰 수의 물리적 레지스터들 내로 리네이밍(renaming)하는 리네이머 유닛(renamer unit)(220)에 연결되는 ROB(215)가 제공될 수 있다. 리네이머(220)로부터, 인스트럭션들이 트레이스 캐시(225)에 연결될 수 있으며, 트레이스 캐시(225)는 실행의 브랜치들의 예측시에 도움을 주기 위해 브랜치 예측기(230)에 연결된다. MITE(microinstruction translation engine)(238)이 번역된 인스트럭션들을 마이크로 시퀀서(235)에 제공하도록 연결되며, 마이크로 시퀀서(235)는 단일화된 캐시 메모리(240)(예를 들면, 레벨 1 또는 레벨 2 캐시)에 연결된다. 데이터 캐시(268) 또한 단일화된 캐시 메모리(240)에 연결될 수 있으며, 로드 큐(267a) 및 저장 큐(267b)에 연결될 수 있다.

<25> 도 3을 여전히 참조하면, 리네이밍된 인스트럭션들이 실행 유닛(250)에 제공될 수 있으며, 실행 유닛(250)은 입력 인스트럭션들을 수신하고, 저장을 위해 그것을 큐에 위치시키며, 다수의 기능 유닛들 중 하나로 스케쥴링하는 이슈 큐(252)를 포함한다. 이슈 큐(252)는 레지스터 파일들, 즉, 부동점(floating-point; FP) 레지스터 파일(254) 및 정수 레지스터 파일(256)의 쌍에 더 연결된다. 인스트럭션에 대해 필요한 소스 데이터가 선택된 레지스터 파일에 제공되는 경우, 인스트럭션은 다수의 기능 유닛들 중 하나에 대해 스케쥴링된 것으로서 실행될 수 있으며, 다수의 기능 유닛들 중 2개가 예시의 용이함을 위해 도 3에 도시된다. 특히, 부동점 논리 유닛 및 정수 논리 유닛에 각각 대응할 수 있는, 제1 실행 유닛(260) 및 제2 실행 유닛(265)이 제공될 수 있다. 인스트럭션들의 실행 결과가, 상호접속 네트워크(270)를 통해 다양한 위치들로 송신될 수 있다. 예를 들어, 주어진 아키텍처에서, 결과 데이터가 레지스터 파일들(254, 256), 로드 큐(267a) 및 저장 큐(267b)에 다시 제공되고/되거나, ROB(215)에 다시 제공될 수 있다.

<26> 더욱이, 결과 데이터는 인스트럭션 검증 유닛(280)에 제공될 수 있다. 유닛(280)은 본 발명의 실시예에 따라 소프트 에러 검출을 수행할 수 있다. 도시된 바와 같이, 인스트럭션 검증 유닛(280)은 상호접속 네트워크(270)에 연결될 수 있고, ROB(215)에 더 연결될 수 있다. 다양한 실시예에서, 인스트럭션 검증 유닛(280)은, ROB(215)의 버퍼와 유사하게 배열된 버퍼를 포함하는 다양한 자원들을 포함할 수 있다. 일실시예에서, 인스트럭션 검증 유닛(280)은 ROB(215)의 확장일 수 있는 RSB(recheck source buffer)를 포함할 수 있다. 즉, RSB는 ROB(215)와 동일한 수의 엔트리들을 포함하는 FIFO(first-in-first-out) 버퍼일 수 있다. 다른 실시예에서, 인스트럭션 검증 목적을 위한 버퍼는 단순히 ROB(215)를 이용할 수 있다. 일실시예에서, ROB(215)의 각각의 엔트리 (그리고, 만약 존재하는 경우, 인스트럭션 검증 유닛(280)에서의 버퍼)는 아래의 표 1에 도시된 형태를 취할 수 있다.

표 1

<27>

인스트럭션	소스 태그들	시간 스탬프	결과

<28> 표 1에 도시된 바와 같이, ROB(215)의 각각의 엔트리는 다양한 필드들을 포함할 수 있다. 특히, 표 1에 도시된 바와 같이, 각각의 엔트리는 마이크로오퍼레이션에 대응할 수 있는 인스트럭션 필드를 포함할 수 있다. 더욱이, 각각의 엔트리는 필요한 데이터의 위치를 식별하기 위한 소스 태그들을 포함할 수 있다. 또한, 각각의 엔트리는 ROB(215)에서 엔트리가 생성된 시간을 나타내는 시간 스탬프를 포함할 수 있다. 마지막으로, 표 1에 도시된 바와 같이, 각각의 엔트리는 인스트럭션의 결과를 저장하기 위한 결과 필드를 포함할 수 있다. 표 1의 실시예에서의 이러한 특정 구현으로 기술되었지만, 본 발명의 영역은 그렇게 제한되지 않으며, ROB 또는 RSB에서의 엔트리들은 추가적이거나 또는 상이한 필드들을 포함할 수 있다.

<29> 인스트럭션 검증 유닛(280)은 선택된 인스트럭션을 재실행하기 위해 그 자신의 전용 기능 유닛들(285)을 더 포함할 수 있다(또는, 대안적으로, 이미 이용가능한 기능 유닛들을 이용할 수 있다). RSB는 모든 인스트럭션들, 인스트럭션 오피코드들 및 소스 태그들에 의해 생성된 결과 값을 유지하여, 인스트럭션이 취약한 것으로서 식별되는 경우, 수행(commit) 시간에 인스트럭션의 결과 (또는, 그것이 메모리 또는 브랜치 동작인 경우, 어드레스)를 검증할 수 있다.

<30> 인스트럭션 검증 유닛(280)은 소프트 에러 검출을 수행하기 위한 마이크로제어기, 로직 또는 다른 자원들을 더 포함할 수 있다. 특히, 자원은, 예를 들면, 수행 시간에 인스트럭션을 수신하여, 인스트럭션에 대한 취약성 측정치를 결정할 수 있다. 이러한 취약성 측정치는 많은 상이한 형태를 취할 수 있으며, 일실시예에서, 인스트럭션 검증 유닛(280)은 이하의 수학식에 따라 인스트럭션 취약성 레벨을 결정하기 위한 계산을 수행할 수 있다.

수학식 1

<31> 인스트럭션 취약성 레벨 = 점유된 비트 영역 X 소모 시간

<32> 인스트럭션 취약성 레벨은 소스 식별자들, 목적지 식별자들 등의 그의 다양한 필드들을 포함하는 인스트럭션의 점유된 비트 영역에, 예를 들면, ROB(215)로의 초기 인스트럭션 삽입으로부터 인스트럭션 수행까지의 시간 스탬프에 의해 측정된 것으로서의, 인스트럭션이 프로세서에서 소모한 시간에 대응할 수 있는 프로세서에서의 소모 시간을 승산한 것에 근거할 수 있다. 다른 실시예에서, 취약성 측정치는 점유된 비트 영역 및 소모 시간 중 단지 하나, 또는 이들 값들의 상이한 결합들에 근거할 수 있다.

<33> 이러한 인스트럭션 취약성 레벨은 임계값과 비교될 수 있다. 동작의 예로서, 임계값 1000을 가정한다. 평균적으로, 인스트럭션이 50 비트의 취약한 비트 공간을 커버한다면, 이슈 큐에서 20 사이클보다 많이 소모하는 임의의 인스트럭션이 선택적 복제를 통해 재실행된다. 그러한 지연들은 부동점 분할과 같은 긴 의존성 체인 또는 긴 대기시간 동작 때문일 수 있다. 다양한 실시예에서, 임계값은 광범위하게 변할 수 있다. 예를 들어, 임계값 0은 모든 인스트럭션의 재실행을 초래할 것이며, 큰 임계값(예를 들어, 5000)은 단지 작은 백분율(예를 들면, 10% 미만)의 인스트럭션들의 재입력을 초래할 것이다. 일실시예에서, 임계값 1000은 양호한 트레이드오프를 나타낼 수 있다.

<34> 취약성 레벨이 임계값보다 작은 경우, 인스트럭션은 정상적으로 수행되고, 인스트럭션 검증 유닛(280)은 더 이상의 동작을 취하지 않는다. 그 대신, 인스트럭션 취약성 레벨이 선택된 임계값보다 크다고 결정되는 경우, 인스트럭션은 소프트 에러에 더욱 취약할 수 있다. 인스트럭션 취약성 값이 높다면, 이것은 인스트럭션이 다수의 비트를 점유하고/하거나, 프로세서 구성요소에서 긴 시간을 소모하여, 그것을 더욱 취약하게 만든다는 것을 의미하는 것이다. 다양한 실시예에서, 선택된 취약성 임계값보다 높은 인스트럭션들만이 복제된다. 이러한 방식으로, 최대량의 에러 커버리지는, 최소수의 인스트럭션들을 복제함으로써 영향을 받는다. 인스트럭션 수행에서 인스트럭션 취약성 분석을 수행함으로써, 수행 단계에 도달하기 전에 파이프라인으로부터 제거된 구조적으로 정확한 실행(architecturally correct execution; ACE)이 아닌 인스트럭션들이 필터링된다. 또한, 인스트럭션이 프로세서 내부에서 소모한 시간량은 인스트럭션이 점유한 공간의 양과 더불어 정확하게 알려지기 때문에, 인스트럭션 취약성 정보가 수행 시간에 보다 정확하게 수집될 수 있다.

<35> 다양한 실시예에서, 인스트럭션 검증 유닛(280)은 인스트럭션의 정보를 취하고, 인스트럭션의 소스 데이터를 획득하여, 그것을 인스트럭션 검증 유닛(280)과 관련된 하나 이상의 추가적인 기능 유닛들(285)에 제공함으로써, 인스트럭션을 재실행할 수 있다. 다양한 실시예에서, 취약한 인스트럭션은 RSB 내에 저장된 정보를 이용함으로써 재실행된다. 하나의 구현에서, 소스 레지스터 태그들이 RSB에 저장되고, 검증을 위해서 소스 데이터를 수집하기 위해 레지스터 파일들(254, 256)을 액세스하는데 이용된다. 이들 레지스터 파일들은 검증 목적을 위해 2개의 추가적인 판독 포트들(각각의 소스 오퍼랜드에 대해 하나의 포트임)를 포함할 수 있다. 따라서, 선택적 재실행을 위해 레지스터 파일들(254, 256)을 액세스하는 것은, 인스트럭션이 이슈 큐(252) 내에 있고 소스 태그들이 입자 충돌에 취약한 동안 발생되는 에러들을 커버할 수 있다. 대안적으로, 소스 값들은 그것들이 처음에 판독될 때 RSB 내에 저장됨으로써, 레지스터 파일들 상에 추가적인 포트들을 갖는 것을 회피할 수 있다.

<36> 재실행된 인스트럭션의 결과는 인스트럭션 검증 유닛(280)으로 다시 전달되며, 여기서, 재실행된 결과가 원래의 결과와 비교된다. 2개의 결과들이 매칭된다면, 인스트럭션 검증 유닛(280)에 의해 추가의 동작이 취해지지 않으며, 인스트럭션은 회수된다. 그러나, 결과들이 매칭되지 않는다면, 소프트 에러가 표시되며, 인스트럭션 검증 유닛(280)은 프로세서(200) 내의 하나 이상의 위치들에 소프트 에러를 시그널링한다. 이 때, 프로세서(200)는 에러 복원 메카니즘을 수행할 수 있다. 예를 들어, 2개의 결과들이 미스매칭된다면, 인스트럭션 검증 유닛(280)은 결함이 있는 인스트럭션으로부터 시작하는 실행을 재개하는, 프로세서의 플러시(flush)을 개시할 수 있다. 도 3의 실시예에서의 특정 구현으로 기술되었지만, 본 발명의 영역은 그렇게 제한되지 않음을 이해할 것이다.

<37> 소정의 실시예들에서, 몇몇 최적화가 가능하다. ROB의 헤드 또는 다른 회수 위치에서의 인스트럭션만을 검증하는 대신에, 매 사이클마다 하나보다 많은 인스트럭션에 대한 검증이 실행될 수 있다. 더욱이, 검증 때문에 프

로세서가 성능을 잃을 때의 시간 기간을 식별하기 위해 다수의 임계값들 및 성능 매트릭이 구현될 수 있다. 그러한 기간들 동안, 매우 취약한 인스트럭션들만이 복제될 수 있다. 예로서, 낮은 성능의 시간들 동안, 임계값은 재실행되는 인스트럭션의 수를 감소시키기 위해 보다 높게 설정될 수 있다. 더욱이, 프로세서 상태(예를 들면, 에러율, 성능, 전력 등)에 따라 변경되는 적응적 임계값 레벨이 이용될 수 있다. 따라서, 주어진 프로세서 상태에 따라, 다수의 상이한 취약성 임계값들 중 하나가, 계산된 인스트럭션 취약성 값들과의 비료를 위해 선택될 수 있다.

<38> 몇몇 실시예에서, 본 발명의 실시예에 따른 에러 검출은, 플러시 및 재시작 또는 좁은 값 식별과 같은 취약성 감소 기법들과 함께 이용될 수 있다. 예를 들면, 오프-칩 캐시 미스(off-chip cache miss)에 따라 파이프라인을 플러싱 및 재시작하는 것은, 이슈 큐 내에서 인스트럭션들이 소모한 시간을 감소시킴으로써 많은 인스트럭션들의 소프트 에러 취약성을 감소시킬 것이며, 따라서, 소프트 에러 검출을 통해 재실행된 인스트럭션들의 수를 감소시킬 수 있다. 더욱이, 보다 낮은 임계값 레벨들이, 플러시 및 재시작 메카니즘 또는 다른 취약성 감소 기법이 적절한 경우, 에러 커버리지를 증가시키도록 설정될 수 있다.

<39> 실시예들은 여러 가지 상이한 시스템 유형들로 구현될 수 있다. 이제, 도 4를 참조하면, 본 발명의 실시예에 따른 멀티프로세서 시스템의 블록도가 도시되어 있다. 도 4에 도시된 바와 같이, 멀티프로세서 시스템은 점-대-점(point-to-point) 상호접속 시스템이며, 제1 프로세서(470)와, 비록 다른 실시예들에서는 다른 종류의 상호접속들이 이용될 수 있지만, 점-대-점 상호접속(450)을 통해 연결된 제2 프로세서(480)를 포함한다. 도 4에 도시된 바와 같이, 프로세서들(470, 480) 각각은, 제1 및 제2 프로세서 코어들(즉, 프로세서 코어들(474a, 474b) 및 프로세서 코어들(484a, 484b))을 포함하는 멀티코어 프로세서들일 수 있다. 예시의 용이성을 위해 도시되지 않았지만, 제1 프로세서(470) 및 제2 프로세서(480) (그리고, 보다 구체적으로, 그 내부의 코어들)은 취약 인스트럭션 식별 및 검증 로직을 포함하여, 본 발명의 실시예에 따라 소프트 에러들을 검출할 수 있다. 제1 프로세서(470)는 메모리 제어기 허브(memory controller hub; MCH)(472) 및 점-대-점(P-P) 인터페이스들(476, 478)을 더 포함한다. 마찬가지로, 제2 프로세서(480)는 MCH(482) 및 P-P 인터페이스들(486, 488)을 포함한다. 도 4에 도시된 바와 같이, MCH들(472, 482)은 프로세서들을 각각의 메모리들, 즉, 각각의 프로세서들에 국부적으로 부착된 주 메모리의 부분들인 메모리(432) 및 메모리(434)에 연결한다.

<40> 제1 프로세서(470) 및 제2 프로세서(480)는 P-P 상호접속들(452, 454) 각각을 통해 칩셋(490)에 연결될 수 있다. 도 4에 도시된 바와 같이, 칩셋(490)은 P-P 인터페이스들(494, 498)을 포함한다. 더욱이, 칩셋(490)은, 칩셋(490)을 고성능 그래픽 엔진(438)과 연결하는 인터페이스(492)를 포함한다. 일실시예에서, AGP(Advanced Graphics Port) 버스(439)는 그래픽 엔진(438)을 칩셋(490)에 연결하는데 이용될 수 있다. AGP 버스(439)는 캘리포니아주, 산타 클라라 소재의 Intel Corporation에 의해, 1998년 5월 4일에 간행된 Accelerated Graphics Port Interface Specification, Revision 2.0을 따를 수 있다. 대안적으로, 점-대-점 상호접속(439)이 이들 구성요소들을 연결할 수 있다.

<41> 칩셋(490)은 인터페이스(496)를 통해 제1 버스(416)에 연결될 수 있다. 일실시예에서, 제1 버스(416)는 PCI Local Bus Specification, Production Version, Revision 2.1(1995년 6월)과 같은 PCI(Peripheral Component Interconnect) 버스, 또는 PCI Express 버스 또는 제3 세대 입/출력(I/O) 상호접속 버스와 같은 버스일 수 있으나, 본 발명의 영역은 그렇게 제한되지 않는다.

<42> 도 4에 도시된 바와 같이, 다양한 I/O 장치들(414)이, 제1 버스(416)를 제2 버스(420)와 연결하는 버스 브리지(418)와 함께, 제1 버스(416)에 연결될 수 있다. 일실시예에서, 제2 버스(420)는 LPC(low pin count) 버스일 수 있다. 다양한 장치들은, 예를 들면, 키보드/마우스(422), 통신 장치(426), 및 일실시예에서 코드(430)를 포함할 수 있는 데이터 저장 유닛(428)을 포함하는 제2 버스(420)에 연결될 수 있다. 더욱이, 오디오 I/O(424)는 제2 버스(420)에 연결될 수 있다. 다른 아키텍처들이 가능함을 주지해야 한다. 예를 들어, 도 4의 점-대-점 아키텍처 대신에, 시스템은 멀티-드롭(multi-drop) 버스 또는 다른 그러한 아키텍처를 구현할 수 있다.

<43> 더욱이, 다양한 실시예에서, 상이한 프로세서 구조들 및 선택적 인스트럭션 복제를 수행하는 상이한 방법들이 실현될 수 있음을 이해할 것이다. 예를 들어, 몇몇 실시예에서, 인스트럭션들은 그것들이 회수가능한 때와 그것들이 실제로 회수되는 때 사이의 시간 윈도우 동안 선택적으로 재실행될 수 있다. 인스트럭션들의 그러한 선택적 재발행을 수행하는 상이한 방법들이 실현될 수 있다. 이제, 도 5를 참조하면, 선택적 인스트럭션 재실행을 위한 본 발명의 일시예에 따른 프로세서의 일부분의 블록도가 도시되어 있다.

<44> 도 5에 도시된 바와 같이, 프로세서(500)는 내부 버스(505)를 통해 입력 인스트럭션들을 수신하도록 연결되는 이슈 큐(510)를 포함한다. 도 5에 더 도시된 바와 같이, 선택적 큐(520)가 입력 인스트럭션들을 수신하기 위해

더 연결될 수 있다. 또한, 인스트럭션들은 ROB(515)에 제공될 수도 있다.

<45> 도 5에 더 도시된 바와 같이, 이슈 큐(510) 및 선택적 큐(520)는, 레지스터 파일(530)에 전달하기 위해 이들 큐들 중 하나로부터 인스트럭션을 선택할 수 있는 선택기(525)에 연결되며, 레지스터 파일(530)은 하나 이상의 실행 유닛들(540)에 연결될 수 있다. 도 5에 도시된 바와 같이, 실행 유닛(들)(540)은 ROB(515)에 더 연결될 수 있다. 실행을 위해 포트가 이용가능한 경우, 선택적 큐(520) 내의 인스트럭션(이슈 큐(510) 내의 그의 대응부는 이미 발행됨)이 발행되어 실행된다. 인스트럭션이 그의 실행을 종료한 경우, 결과들이 ROB(515)에 저장된다. 복제 인스트럭션 실행이 종료된 경우, 그 결과는 검증 목적을 위해, 저장된 원래의 결과와 비교된다. ROB(515)의 헤드가 수행될 준비는 되었지만 검증되지 않은 경우, 수행은 중지된다.

<46> 몇몇 실시예에서, ROB(515)는 소정의 필드들(예를 들면, 비교 비트, 검증 비트, 에러 검출 비트, 및 결과를 저장하기 위한 비트들)을 각각의 엔트리에 부가할 수 있다. 그러나, 다른 실시예에서, 그러한 저장을 위해 다른 어레이가 이용될 수 있다. ROB 엔트리가 새로운 인스트럭션에 할당될 때, 엔트리 내의 이들 보조의 필드들은 리세트된다. 인스트럭션들이 그들의 실행을 종료한 경우, 그들의 결과들은 ROB(515)에 기록된다. 원래의 인스트럭션이 종료되는 경우, 결과가 기록되고, 비교 비트가 설정된다. 그 다음, 제2 인스트럭션(즉, 복제)은, 실행되는 경우, 설정된 비교 비트를 찾아서, 저장된 값이 재실행된 결과와 비교되도록 하고, ROB 엔트리에서 검증된 비트를 설정한다. 비교 결과는 에러 검출 비트에 저장되어, 결과들이 매칭되는지 여부를 나타낸다. 비교 비트에 대한 한 가지 대안은, 그것이 원래의 인스트럭션인지 또는 복제 인스트럭션인지 여부를 식별하기 위한 인스트럭션과 관련된 비트일 수 있다.

<47> 전술한 바와 같이, 인스트럭션이 이슈 큐(510) 내에 위치될 때, 그것은 선택적 큐(520) 내에도 저장된다. 선택적 큐(520) 내의 각각의 엔트리는 오피코드, (레지스터 파일(530)로부터 소스들을 판독하기 위한) 소스 태그들, ROB 엔트리 식별자, 및 재발행을 위한 준비가 되었는지 여부를 나타내는 준비 비트를 포함할 수 있다. 이슈 큐(510) 내의 엔트리는, 그의 대응하는 엔트리를 ROB(515)에 저장할 뿐만 아니라, 그의 대응하는 엔트리를 선택적 큐(520)에 저장할 수도 있음을 주지해야 한다. 원래의 인스트럭션이 이슈 큐(510)로부터 발행되면, 그것은 선택적 큐(520) 내의 대응하는 엔트리에게 신호를 송신하여, 준비 비트를 설정한다. 선택적 큐(520) 내의 전체 엔트리 수는 ROB(515) 내의 엔트리 수와 동일할 수 있다. 선택적 큐(520)의 대기시간은 성능에 대해 중요하지 않으므로, 보다 느리고 전력 효율적인 설계 및 심지어는 저전력 트랜지스터로 구현될 수 있음을 주지해야 한다.

<48> 선택기(525)는 이슈 큐(510) 및 선택적 큐(520)로부터의 인스트럭션들 사이에서 선택하는 멀티플렉서일 수 있다. 다양한 실시예에서, 선택기(525)는 이슈 큐(510)로부터의 인스트럭션들을 우선순위화할 수 있다. 선택적 큐(520)는 설정된 준비 비트를 갖는 것들로부터 선택기(525)로 전달하기 위한 인스트럭션을 선택할 수 있다. 이것은 게이트들의 체인(그것은 중요 경로에 있지 않으며, 사이클 시간에 영향을 미치지 않기 때문임)으로 수행되거나, 또는 멀티뱅킹(multibanking)에 의해, 즉, 뱅크 내의 가장 오래된 인스트럭션만이 포트를 완료하도록, 이슈 포트만큼 많은 뱅크들을 갖는 선택적 큐(520)를 형성함으로써 수행될 수 있다. 자유(free) 포트가 존재한다면, 선택적 큐(520)로부터의 인스트럭션이 발행되며, 그렇지 않은 경우, 대기한다.

<49> 전술한 바와 같이, 인스트럭션의 소프트 에러 취약성은 프로세서에서 인스트럭션이 점유하는 영역 및 소모된 시간에 의존할 수 있다. 선택적 큐(520) 내의 엔트리가 설정된 준비 비트를 갖는다면, 그것의 취약성 값은 임계값과 비교될 수 있다. 인스트럭션이 이슈 큐(510) 및 선택적 큐(520)에 동시에 할당되기 때문에, 이슈 큐(510)에서 인스트럭션이 얼마나 많은 사이클을 소모하는지 알려진다는 것을 주지해야 한다. 선택적 큐(520)에서의 배치와 준비 비트를 설정하기 위한 신호의 수신 사이의 경과 시간은 소모된 시간의 측정치로서 이용될 수 있다. 취약성 값이 임계값보다 작은 경우, 선택적 큐(520) 내의 엔트리는 자유롭게 될 수 있고, 대응하는 ROB 엔트리 내의 검증된 비트는 설정되고, ROB 엔트리에서의 에러 검출된 비트는 리세트된다. 몇몇 실시예에서, (예를 들면, 시간 스탬프들을 통한) 시간은 영역과 시간의 곱 대신에, 취약성 값으로서 이용될 수 있다.

<50> 검증 목적을 위해, ROB(515) 내의 각각의 엔트리는, 위에서 설명된, 검증된 비트를 포함할 수 있다. 검증된 비트가 설정되지 않는다면, 수행은 검증을 대기하는 것을 중지할 수 있다. 검증된 비트가 설정된다면, 인스트럭션은, 그의 에러 검출된 비트가 리세트되는 경우에만 수행할 준비가 된다. 에러 검출된 비트가 설정된다면, 상이한 동작들이 취해질 수 있다. 예컨대, 파이프라인이 플러싱되어 결함 인스트럭션을 재실행하거나, 또는 예외가 발생될 수 있다. 잘못된 브랜치 예측(branch misprediction)시에, 엔트리들은 이슈 큐(510)에서 더 이상 유효하지 않으며, 선택적 큐(520)가 제거될 수 있다. 이슈 큐(510)에서 이용된 동일한 메카니즘이, 선택적 큐(520)에서의 엔트리들을 제거하는데 이용될 수 있다.

<51> 중복 하드웨어 자체가 입자 충돌에 취약할 수 있음을 주지해야 한다. 그러나, 그것의 취약성은 0이다. 즉, 선

택적 큐(520)에 대한 충돌이 존재한다면, 잘못된 인스트럭션이, 잘못된 포지티브를 발생시킬 것 같은 것을 실행할 것이다. ROB(515)의 헤드는 검증되도록 대기할 것이므로, 충돌이 ROB 엔트리 식별자 또는 ROB(515) 내의 검증된 비트를 히트(hit)하여, 데드락(deadlock)을 초래할 수 있다. 이것은 선택적 큐(520) 내의 ROB 엔트리 식별자를 패리티 보호하거나 또는 워치도그 타이머(watchdog timer)를 부가함으로써 해결될 수 있으며, 수행이 주어진 수의 사이클들보다 많이 지연된다면, 인스트럭션은 스쿼싱(squashing) 및 재시작될 수 있다.

<52> 실시예들은 코드 내에서 구현될 수 있으며, 시스템이 인스트럭션들을 수행하도록 프로그래밍하는데 이용될 수 있는 인스트럭션들을 저장한 저장 매체 상에 저장될 수 있다. 저장 매체는, 제한적인 것은 아니지만, 플로피 디스크, 광학 디스크, CD-ROM(compact disk read-only memory), CD-RW(compact disk rewritable) 및 자기 광학 디스크(magneto-optical disk)를 포함하는 임의의 유형의 디스크와, ROM(read-only memory), DRAM(dynamic random access memory), SRAM(static random access memory)와 같은 RAM, EPROM(erasable programmable read-only memory), 플래시 메모리, EEPROM(electrically erasable programmable read-only memory), 자기 또는 광학 카드와 같은 반도체 디바이스, 또는 전자 인스트럭션들을 저장하기에 적합한 임의의 다른 유형의 매체를 포함할 수 있다.

<53> 본 발명은 제한된 수의 실시예들에 대하여 기술되었지만, 당업자라면 그로부터의 다양한 변경 및 수정이 가능함을 이해할 것이다. 첨부된 특허청구범위는 그와 같은 모든 변경 및 수정을 커버하며, 본 발명의 진정한 사상 및 영역 내에 속하는 것으로서 의도된다.

도면의 간단한 설명

<4> 도 1은 본 발명의 일실시예에 다른 방법의 흐름도이다.

<5> 도 2는 본 발명의 일실시예에 따른 일반적 프로세서 아키텍처의 블록도이다.

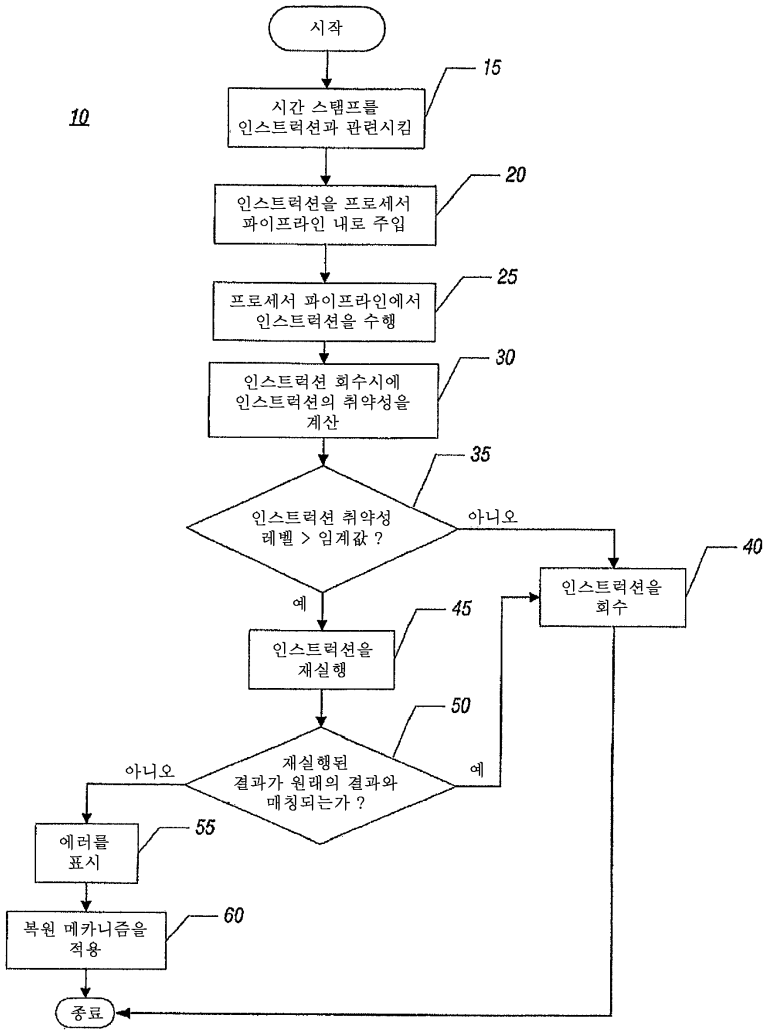
<6> 도 3은 본 발명의 일실시예에 따른 프로세서의 블록도이다.

<7> 도 4는 본 발명의 실시예에 따른 멀티프로세서 시스템의 블록도이다.

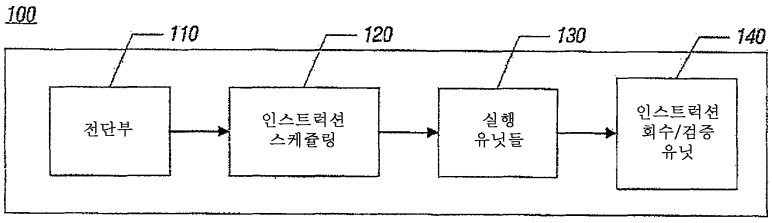
<8> 도 5는 선택적 인스트럭션 재실행을 위한 본 발명의 실시예에 따른 프로세서의 일부분의 블록도이다.

도면

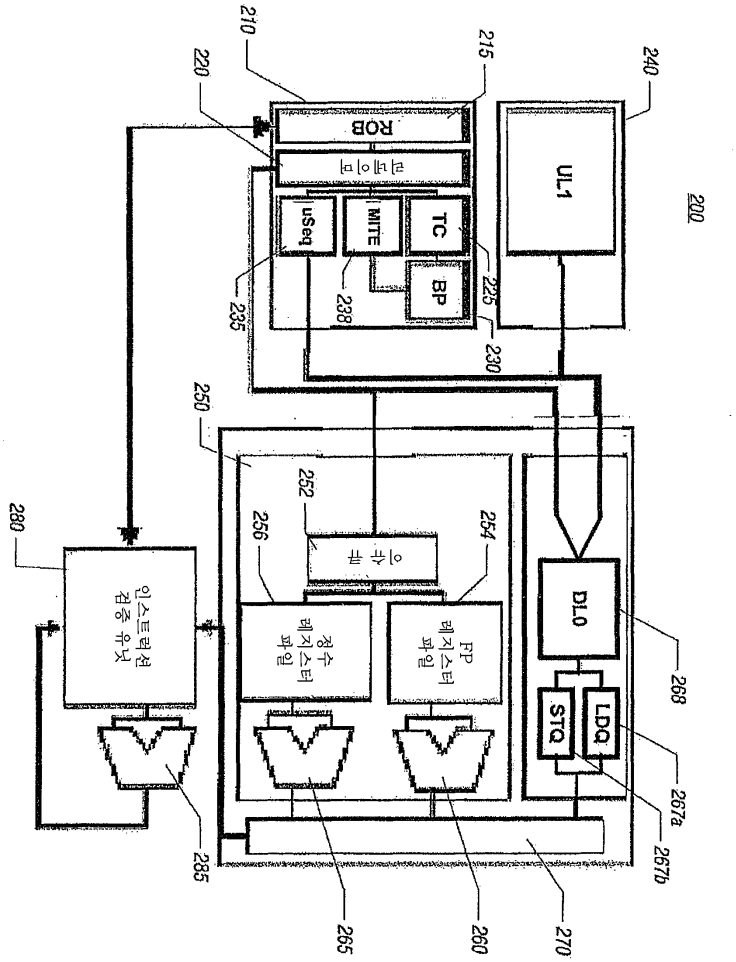
도면1


10
시작
15
시간 스탬프를 인스트럭션과 관련시킴
20
인스트럭션을 프로세서 파이프라인 내로 주입
25
프로세서 파이프라인에서 인스트럭션을 수행
30
인스트럭션 회수시에 인스트럭션의 취약성을 계산
35
인스트럭션 취약성 레벨 > 임계값 ?
아니오
예
40
인스트럭션을 회수
45
인스트럭션을 재실행
50
재실행된 결과가 원래의 결과와 매칭되는가 ?
아니오
예
55
에러를 표시
60
복원 메카니즘을 적용
종료

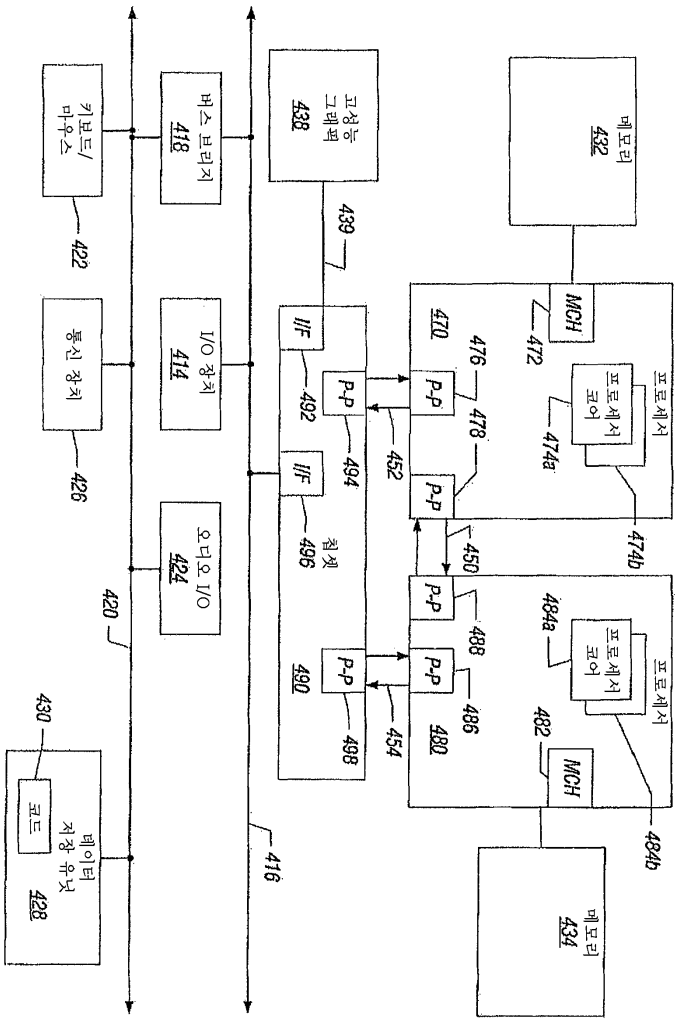
도면2


100
110
120
130
140
전단부
인스트럭션 스케쥴링
실행 유닛들
인스트럭션 회수/검증 유닛

도면3


210
220
215
240
200
ROB
리네이머
uSeq
MITE
TC
BP
UL1
235
238
225
230
250
252
256
254
인큐
정수 레지스터 파일
FP 레지스터 파일
280
인스트럭션 검증 유닛
DL0
STQ
LDQ
268
267a
267b
285
265
260
270

도면4


메모리
432
MCH
472
프로세서
474a
474b
프로세서
코어
470
476
478
P-P
P-P
P-P
450
452
484a
484b
482
MCH
프로세서
코어
488
P-P
P-P
P-P
486
480
454
490
498
I/F
I/F
칩셋
492
494
496
439
고성능
그래픽
438
416
버스 브리지
418
I/O 장치
414
오디오 I/O
424
420
키보드/
마우스
422
통신 장치
426
430
코드
데이터
저장 유닛
428
메모리
434

도면5