(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.[8]
G06F 9/40 (2006.01)

(11) 공개번호 10-2006-0004953
(43) 공개일자 2006년01월16일

(21) 출원번호 10-2005-7020027
(22) 출원일자 2005년10월21일
번역문 제출일자 2005년10월21일
(86) 국제출원번호 PCT/GB2004/001824
국제출원일자 2004년04월28일
(87) 국제공개번호 WO 2004/097631
국제공개일자 2004년11월11일

(30) 우선권주장 0310180.5 2003년05월02일 영국(GB)
0326437.1 2003년11월13일 영국(GB)

(71) 출원인 트랜시티브 리미티드
영국, 런던 이씨2브이 7큐제이, 노블 스트리트 10, 엘더 캐슬 5층

(72) 발명자 오웬, 다니엘
영국, 엠3 2이쥐, 맨체스터, 블랙프라이어스스트리트, 메이브룩하우스, 트랜시티브 리미티드
앤드류스, 조나단, 제이
영국, 엠3 2이쥐, 맨체스터, 블랙프라이어스스트리트, 메이브룩하우스, 트랜시티브 리미티드
하우슨, 마일즈, 필립
영국, 엠3 2이쥐, 맨체스터, 블랙프라이어스스트리트, 메이브룩하우스, 트랜시티브 리미티드
하이켄, 데이비드
영국, 엠3 2이쥐, 맨체스터, 블랙프라이어스스트리트, 메이브룩하우스, 트랜시티브 리미티드

(74) 대리인 정홍식
고일남

심사청구 : 없음

(54) 프로그램 코드 변환을 위한 중간 표현들을 발생하기 위한개선된 아키텍처

요약

본 발명은 프로그램 코드 변환을 위한 개선된 아키텍처 및 프로그램 코드 변환을 위한 중간 표현들을 발생하기 위한 방법에 관한 것이다. 프로그램 코드 변환 장치는 번역될 서브젝트 코드의 중간 표현으로 어느 타입의 IR 노드들을 발생시켜야 하는가를 결정한다. 상기 변환에 연관된 특정 서브젝트 및 타깃 계산 환경들에 의존하여, 프로그램 코드 변환 장치는 중간 표현을 발생하는 데 있어서, 베이스 노드들, 복합 노드들, 폴리모픽 노드들, 및 아키텍처 특정 노드들 모두, 또는 이들의 몇몇 조합들을 이용한다.

대표도

도 5

명세서

기술분야

본 발명은 일반적으로 컴퓨터들 및 컴퓨터 소프트웨어 분야에 관한 것으로, 특히 예를 들면 코드 번역기들, 에뮬레이터들 및 가속기들에 유용한 프로그램 코드 변환 방법들 및 장치에 관한 것이다.

배경기술

내장 및 비내장 CPU 시장에 걸쳐, 사람은 성능이 "가속(Accelerated)"되거나, 더 나은 비용/성능 이점들을 제공할 수 있는 많은 가능한 프로세서들로, 이들이 관련 소프트웨어를 명백하게 액세스할 수 있다면, "번역(Translated)"될 수 있는 대용량의 소프트웨어가 존재하는 우세한 명령 세트 아키텍쳐들(ISAs들)을 발견한다. 사람은 또한 이들 ISA에 제 때에 로크되고, 성능 또는 시장 범위(market reach)에서 발전할 수 없고 "합성 CPU" 코-아키텍쳐("Synthetic CPU" co-architecture)로부터 이익을 얻을 수 있는 지배적인 CPU 아키텍쳐들을 발견한다.

제 2 타입("타깃" 프로세서) 상에서 제 1 타입("서브젝트" 프로세서)의 컴퓨터 프로세서를 위해 쓰여진 프로그램을 실행하는 것이 종종 바람직하다. 여기서, 에뮬레이터 또는 번역기는 프로그램 코드 번역을 수행하기 위해 사용되므로, 서브젝트 프로그램은 타깃 프로세서 상에서 실행할 수 있다. 에뮬레이터는 서브젝트 프로세서를 에뮬레이팅함으로써, 서브젝트 프로그램이 서브젝트 프로세서 상에서 본래 실행되는 것처럼, 가상 환경을 제공한다.

과거에, 서브젝트 코드는, 이 출원서의 도 1 내지 도 5와 관련하여, 명칭이 프로그램 코드 변환(Program Code Conversion)인 WO 00/22521에 기재된 것과 같이, 소위 베이스 노드들을 이용하여 런-타임 번역 중 컴퓨터 프로그램의 중간 표현으로 변환된다. 중간 표현(intermediate representatin; "IR")은 프로그램이 표현될 수 있는 요약 컴퓨터 언어의 형태를 가리키기 위해 컴퓨터 산업에서 널리 사용되는 용어이지만, 임의의 특정 프로세서에 특정되지 않으며, 임의의 특정 프로세서 상에서 직접 실행될 수 있도록 의도되지 않는다. 중간 표현들을 이용하여 이와 같은 가속, 번역 및 공동-아키텍쳐 능력들을 용이하게 하는 프로그램 코드 변환 방법들 및 장치들은, 예를 들면 상기 공보 WO 00/22521에 제기되어 있다.

발명의 상세한 설명

본 발명에 따라, 첨부 청구항들에 기술된 장치 및 방법이 제공되어 있다. 본 발명의 최선의 특징들은 독립 청구항들, 및 다음에 이어지는 설명으로부터 명백하게 될 것이다.

다음은 본 발명에 따른 프로그램 코드 변환을 위한 개선된 아키텍쳐의 여러 실시예들에 따라 실현 가능한 여러 양태들 및 이점들의 요약이다. 그것은 이 기술 분야에서 숙련된 사람이 여기에 첨부된 청구항들의 범위를 제한하지 않고 어떤 식으로든 제한하려고 의도되지 않은 다음에 계속되는 본 발명의 상세 논의를 더 신속하게 이해하도록 돕기 위한 소개로서 제공된다.

이하에 기술된 여러 실시예들은 서브젝트 컴퓨팅 환경에서 실행 가능한 서브젝트 코드를 타깃 컴퓨팅 환경에서 실행 가능한 타깃 코드로 변환하기 위한 프로그램 변환 장치 및 관련 방법을 위한 개선된 아키텍쳐들에 관한 것이다. 프로그램 코드 변환 장치는 타깃 코드를 더욱 효율적으로 발생하기 위해 타깃 컴퓨팅 환경을 위해 나중에 최적화될 수 있는 서브젝트 코드의 중간 표현("IR")을 생성한다. 변환과 관련된 서브젝트 및 타깃 컴퓨팅 환경들의 특정 아키텍쳐에 의존하여, 일 실시예의 프로그램 코드 변환 장치는 다음과 같은 타입들의 IR 노드들 즉 베이스 노드들, 복합 노드들, 폴리모픽 노드들, 및 아키텍쳐-특정 노드들 중 어느 것이 중간 표현에서 발생하는 가를 결정한다. 프로그램 코드 변환 아키텍쳐는, 상기 타입들의 노드들 중 다른 하나가 실행될 특정 변환에 더 적용 가능할 수 있는 것으로 결정되지 않는다면, 중간 표현을 생성할 때 디폴트로 베이스 노드들을 발생할 것이다.

베이스 노드들은 서브젝트 코드를 실행하는 임의의 서브젝트 아키텍쳐의 시맨틱을 표현하는 데 요구되는 노드들의 최소 세트(즉, 요약 표현식들)를 제공하므로, 베이스 노드들은 RISC와 같은 기능을 제공한다. 복합 노드들은 베이스 노드들보다 더 컴팩트한 표현으로 서브젝트 코드를 실행하는 서브젝트 아키텍쳐의 CISC와 같은 시맨틱을 표현하는 일반 노드들이다. 모든 복합 노드들은 동일한 시맨틱을 갖는 베이스 노드 표현들로 분해될 수 있지만, 복합 노드들은 번역기의 성능을 향상시키기 위해 단일 IR 노드에 복합 명령들의 시맨틱을 보유한다. 복합 노드들은 필수적으로 서브젝트 코드 내의 CISC와 같

은 명령들을 위한 베이스 노드들의 세트를 증가시킨다. 베이스 노드들 및 복합 노드들은 모두 가능한 서브젝트 및 타깃 아키텍쳐들의 넓은 범위에 걸쳐 일반적으로 사용되고 있으므로, 일반 최적화들이 베이스 노드들 및 복합 노드들로 구성된 대응하는 IR 트리들에 대해 수행될 수 있게 허용한다.

타깃 컴퓨팅 환경의 특징들이 특정 서브젝트 명령의 시맨틱으로 하여금 일반IR 노드로서 실현되면 상실되게 할 경우 프로그램 변환 장치는 중간 표현에 폴리모픽 노드들을 이용한다. 폴리모픽 노드들은 소스 코드에서 특정 서브젝트 명령에 특정인 타깃 컴퓨팅 환경의 펑크션에 대한 펑크션 포인터를 포함한다. 프로그램 코드 변환 장치는 또한 특정 타깃 환경들에 대해 전용 코드 발생 펑크션들을 수행하기 위한 타깃-전용 변환 성분들을 제공하기 위해 아키텍쳐-특정 노드들을 이용한다.

이하에 기술되는 개선된 IR 발생 방법들은 최적 수준의 성능을 유지하고 번역 속도를 최대화하면서 프로그램 코드 변환 장치가 임의의 서브젝트와 타깃 프로세서 아키텍쳐 페어링으로 구성 가능하게 허용한다.

본 발명의 더 나은 이해를 위해 그리고 본 발명의 실시예들이 어떻게 실행될 수 있는 지를 보이기 위해, 첨부 개략도들을 예로서 참조할 것이다.

도면의 간단한 설명

도 1은 서브젝트 및 타깃 컴퓨팅 환경들을 포함하는 예시적 컴퓨팅 환경을 도시한 도면.

도 2는 최선의 프로그램 코드 변환 장치를 도시한 도면.

도 3은 서브젝트 코드-타깃 코드로의 번역을 설명하는 예시적인 컴퓨팅 환경의 개략도.

도 4는 본 발명의 최선의 실시예에 따라 프로그램 코드 변환 장치에 의해 실현되는 다양한 중간 표현들의 개략도.

도 5는 최선의 프로그램 코드 변환 장치의 상세 개략도.

도 6은 베이스 노드들 및 복합 노드들을 이용하여 발생된 예시적 IR 트리들을 도시한 도면.

도 7은 가속기에서의 본 발명의 구현을 위한 ASN 발생의 예를 도시한 개략도.

도 8은 번역기에서의 본 발명의 구현을 위한 ASN 발생의 예를 도시한 개략도.

도 9는 본 발명의 최선의 실시예에 따라 ASN들을 이용할 때의 번역 처리의 동작 흐름도.

도 10은 번역 처리 및 상기 처리 중 발생된 대응하는 IR의 예를 도시한 개략도.

도 11은 번역 처리 및 상기 처리 중 발생된 대응하는 IR의 다른 예를 도시한 개략도.

도 12는 번역 처리 및 상기 처리 중 발생된 대응하는 IR의 또 다른 예를 도시한 개략도.

실시예

다음의 설명은 이 기술분야에서 숙련된 어떤 사람이라도 본 발명을 만들고 이용하고 이들 발명을 행하는 발명자들에 의해 시도된 최선의 형태들을 설명할 수 있도록 제공된다. 그러나, 본 발명의 일반적인 원리들이 특히 프로그램 코드 변환 장치에 대해 개선된 아키텍쳐를 제공하기 위해 본원에 정의되었기 때문에, 다양한 변형예들은 이 기술분야에서 숙련된 사람들에게 용이하게 명백한 채로 있을 것이다.

도 1를 참조하면, 서브젝트 컴퓨팅 환경(1) 및 타깃 컴퓨팅 환경(2)을 포함하는 예시적인 컴퓨팅 환경이 도시되어 있다. 서브젝트 환경(1)에 있어서, 서브젝트 코드(10)는 고유하게 서브젝트 프로세서(12)에 대해 실행 가능하다. 서브젝트 프로세서(12)는 한 세트의 서브젝트 레지스터들(14)을 구비한다. 여기서, 서브젝트 코드(10)는, 이 기술분야에서 숙련된 사람에게 친숙한 것과 같이, 서브젝트 코드(10)와 서브젝트 프로세서(12) 사이의 중간 계층들(즉, 컴파일러)을 갖는 임의의 적합한 언어로 표현될 수 있다.

한 세트의 타깃 레지스터들(24)을 이용하여 타깃 프로세서(22)를 제공하는 타깃 컴퓨팅 환경(2)에서 서브젝트 코드(10)를 실행하는 것이 요망된다. 이들 2개의 프로세서들(12, 22)은 고유하게 비호환성일 수 있으므로, 이들 2개의 프로세서들은 상이한 명령 세트들을 이용한다. 그러므로, 프로그램 코드 변환 아키텍처(30)는 비호환 환경에서 서브젝트 코드(10)를 실행하기 위해, 타깃 컴퓨팅 환경(2)에 제공된다. 프로그램 코드 변환 아키텍처(30)는 번역기, 에뮬레이터, 가속기, 또는 하나의 프로세서 타입을 위해 설계된 프로그램 코드를 다른 프로세서 타입에 대해 실행 가능한 프로그램 코드로 변환하는 데 적합한 임의의 다른 아키텍처를 포함할 수 있다. 이후 이어지는 본 발명의 논의를 위해, 프로그램 코드 변환 아키텍처(30)는 "번역기(translator; 30)"라 한다. 2개의 프로세서들(12, 22)은 또한 가속기의 경우에서와 같이 동일한 아키텍처 타입으로 될 수 있음을 유의해야 한다.

번역기(30)는 서브젝트 코드(10)에 대해 번역 처리를 수행하고 타깃 프로세서(22)에 의해 실행하기 위한 번역된 타깃 코드(20)를 제공한다. 적합하게, 번역기(30)는 2진 번역을 수행하고, 여기서 서브젝트 프로세서(12)에 적합한 실행 가능한 2진 코드의 형태의 서브젝트 코드(10)는 타깃 프로세서(22)에 적합한 실행 가능한 2진 코드로 번역된다. 번역은 정적으로 또는 동적으로 수행될 수 있다. 정적 번역에 있어서, 전체 프로그램은 타깃 프로세서에 대해 번역된 프로그램을 실행하기 전에 번역된다. 이것은 중요한 지연을 수반한다. 그러므로, 번역기(30)는 바람직하게는 타깃 프로세서(22)에 대해 즉시 실행하기 위한 서브젝트 코드(10)의 작은 섹션들을 동적으로 번역한다. 이것은 훨씬 더 효율적인 데, 그 이유는 서브젝트 코드(10)의 큰 섹션들은 실제로 이용되지 않을 수 있거나 단지 드물게 이용될 수 있기 때문이다.

도 2를 참조하면, 번역기(30)의 최선의 실시예가 프론트 엔드(31), 커널(32) 및 백 엔드(33)를 포함해서 더 상세하게 도시되어 있다. 프론트 엔드(31)는 서브젝트 코드와 연관된 서브젝트 프로세서(12)에 특정으로 구성된다. 프론트 엔드(31)는 서브젝트 코드(10)의 미리 정해진 섹션을 취하고 일반 중간 표현(generic intermediate representation)의 블록("IR 블록")을 제공한다. 커널(32)은, 이 기술 분야에서 숙련된 사람들에게 용이하게 알려진 것과 같이, 최적화 기술들을 채용함으로써 프론트 엔드(31)에 의해 발생된 각 IR 블록을 최적화한다. 백 엔드(33)는 커널(32)로부터 최적화된 IR 블록들을 취하고 타깃 프로세서(22)에 의해 실행 가능한 타깃 코드(20)를 생성한다.

적합하게는, 프론트 엔드(31)는 서브젝트 코드(10)를 기본 블록들로 분할하고, 여기서 각각의 기본 블록은 유일한 입구 포인트에서의 제 1 명령과 유일한 출구 포인트에서의 최후 명령(예를 들면 점프, 호 또는 분기 명령) 간의 명령들의 순차 세트(sequential set)이다. 커널(32)은 단일 유닛으로서 함께 처리될 2개 이상의 기본 블록들을 포함하는 그룹 블록을 선택할 수 있다. 더욱이, 프론트 엔드(31)는 상이한 입력 조건들 하에서 서브젝트 코드의 동일한 기본 블록을 나타내는 이소-블록들(iso-blocks)을 형성할 수 있다. 사용시, 서브젝트 코드(10)의 제 1의 미리 정해진 섹션이 예를 들어 기본 블록으로서 식별되고, 번역 모드에서 타깃 프로세서(22) 상에서 실행하는 번역기(30)에 의해 번역된다. 타깃 프로세서(22)는 이때 타깃 코드(20)의 대응하는 최적화 및 번역된 블록을 실행한다.

번역기(30)는 커널(32)에 적합하게 제공되는 복수의 요약 레지스터들(34)을 구비하며, 이 레지스터들은 서브젝트 코드(10)를 실행하기 위해 서브젝트 프로세서(12) 내에서 이용될 수 있는 물리적 서브젝트 레지스터들(14)을 표현한다. 요약 레지스터들(34)은 서브젝트 프로세서 레지스터들에 대한 서브젝트 코드 명령들의 예측 효과들을 나타냄으로써 에뮬레이팅되는 서브젝트 프로세서(12)의 상태를 정의한다.

이와 같은 구현을 채용하는 구조가 도 3에 도시되어 있다. 도시된 것과 같이, 컴파일링된 원시 서브젝트 코드는 적합한 컴퓨터 메모리 저장 매체(100)에 상주하는 것으로 도시되어 있고, 특히 그리고 대안의 메모리 저장 메카니즘들이 이 기술 분야에서 잘 알려져 있다. 소프트웨어 성분들은 번역기 코드, 번역된 코드로 번역될 원시 서브젝트 코드 및 운영 시스템을 구비한다. 번역기 코드, 즉 번역기를 구현하는 소스 코드의 컴파일링된 버전은 유사하게 적합한 컴퓨터 메모리 저장 매체(102) 상에 상주한다. 번역기는 예를 들면 타깃 프로세서(106), 전형적으로 마이크로프로세서 또는 다른 적합한 컴퓨터 상에서 실행하는 UNIX와 같은 메모리-저장 운영 시스템(104)과 관련하여 실행한다. 도 3에 도시된 구조는 단지 예시적인 것이며 예를 들면 본 발명에 따른 방법들 및 처리들은 운영 시스템과 함께 또는 운영 시스템 아래에서 상주하는 코드로 구현될 수 있다는 것이 이해될 것이다. 번역된 코드는 적합한 컴퓨터 메모리 저장 매체(108)에 상주하는 것으로 도시되어 있다. 서브젝트 코드, 번역기 코드, 운영 시스템, 번역된 코드 및 저장 메카니즘들은 이 기술 분야에 숙련된 사람들에게 잘 알려진 것과 같은, 아주 다양한 형태들 중 어느 하나일 수 있다.

본 발명의 최선의 실시예에 있어서, 프로그램 코드 변환은 번역된 프로그램이 타깃 컴퓨팅 환경에서 실행하고 있는 동안, 런-타임에서 동적으로 수행된다. 번역기(30)는 번역된 프로그램과 인라인으로 실행한다. 번역된 프로그램의 실행 경로는, 서브젝트 코드의 블록을 번역된 코드로 번역하는 번역기 코드를 실행하는 단계; 이후 그 번역된 코드의 블록을 실행하는

단계들을 포함하는 제어 루프이고, 번역된 코드의 각 블록의 단부는 제어를 번역기 코드로 다시 복귀시키는 명령들을 포함한다. 즉, 서브젝트 코드를 번역한 다음 실행하는 단계들이 인터레이싱되므로, 서브젝트 프로그램의 부분들만이 동시에 번역된다.

번역기(30)의 번역 기본 유닛은 기본 블록이며, 이것은 번역기(30)가 서브젝트 코드를 하나의 기본 블록으로 동시에 번역하는 것을 의미한다. 기본 블록은 정확하게 하나의 입구 포인트와 정확하게 하나의 출구 포인트를 갖는 코드 섹션으로서 형식적으로 정의되고, 이것은 블록 코드를 단일 제어 경로로 제한한다. 이 때문에, 기본 블록들은 제어 흐름의 기본 유닛이다.

중간 표현(IR) 트리들

번역된 코드를 발생하는 처리에 있어서, 중간 표현("IR") 트리들은 서브젝트 명령 시퀀스에 기초하여 발생된다. IR 트리들은 계산된 표현식들의 요약 표현들 및 서브젝트 프로그램에 의해 수행되는 동작들인 노드들을 포함한다. 번역된 코드는 이후 IR 트리들에 기초하여 발생한다. 본원에 기재된 IR 노드들의 컬렉션들은 구어체로 "트리들"이라 한다. 우리는 형식적으로 이와 같은 구조들은 트리들이 아닌, 실제로 방향성 비순환 그래프들(directed acyclic graphs; DAGs)임을 알고 있다. 트리의 형식 정의는 각 노드가 많아야 하나의 페어런트(parent)를 가질 것을 요구한다. 상기 실시예들은 IR 발생 중 공통 서브표현식 제거(common subexpression elimination)를 이용하기 때문에, 노드들은 종종 다중 페어런트들을 가질 것이다. 예를 들면, 플래그-영향 명령 결과(flag-affecting 명령 result)의 IR은 2개의 요약 레지스터들에 의해 참조될 것이며, 이들은 목적지 서브젝트 레지스터 및 플래그 결과 패러미터에 대응한다.

예를 들면, 서브젝트 명령(add %rl, %r2, %r3)은 서브젝트 레지스터들 %r2 및 %r3의 콘텐츠의 부가를 수행하고 서브젝트 레지스터 %rl에 그 결과를 저장한다. 따라서, 이러한 명령은 요약 표현식 "%rl = %r2 + %3"에 대응한다. 이러한 예는 명령 오퍼랜드들 %rl 및 %r2를 나타내는 2개의 서브표현식들을 포함하는 부가 표현식을 갖는 요약 레지스터 %rl의 정의를 포함한다. 서브젝트 프로그램과 관련하여, 이들 서브표현식들은 다른 이전의 서브젝트 명령들에 대응할 수 있고, 또는 이들은 중간 상수 값들과 같은 현재의 명령의 상세를 나타낼 수 있다.

"부가(add)" 명령이 파싱되면, 부가를 위한 요약 산술 연산자에 대응하는 새로운 'Y' IR 노드가 발생된다. 'Y' IR 노드는 오퍼랜드들(서브표현식 트리로서 표현된, 서브젝트 레지스터들에 유지된)을 표현하는 다른 IR 노드들에 대한 참조들을 저장한다. 'Y' 노드 자체는 적합한 서브젝트 레지스터 정의(요약 레지스터 %rl, 명령의 목적지 레지스터)에 의해 참조된다. 이 기술 분야에서 숙련된 사람들이 알 수 있는 것과 같이, 일 실시예에 있어서, 번역기는 C++와 같은 객체-지향 프로그래밍 언어를 이용하여 구현된다. 예를 들면, IR 노드는 C++ 오브젝트로서 구현되고, 다른 노드들에 대한 참조들이 이들 다른 노드들에 대응하는 C++ 오브젝트들에 대한 C++ 참조들로서 구현된다. 그러므로 IR 트리는 서로에 대해 다양한 참조들을 포함하는, IR 노드 오브젝트들의 컬렉션으로서 구현된다.

요약 레지스터들(Abstract Registers)

더욱이, 논의하에 있는 실시예에 있어서, IR 발생은 한 세트의 요약 레지스터들(34)을 이용한다. 이들 요약 레지스터들(34)은 서브젝트 아키텍처의 특정 특징들에 대응한다. 예를 들면, 서브젝트 아키텍처(12) 상에 각 물리 레지스터(14)에 대해 유일한 요약 레지스터(34)가 있다. 요약 레지스터들(34)은 IR 발생 중 IR 트리들에 대한 플레이스홀더들(placeholders)로서 작용한다. 예를 들면, 서브젝트 명령 시퀀스의 주어진 포인트에서의 서브젝트 레지스터 %r2의 값은 특정 IR 표현 트리에 의해 표현되고, 이것은 서브젝트 레지스터 %r2에 대한 요약 레지스터(34)와 연관된다. 일 실시예에 있어서, 요약 레지스터(34)는 C++ 오브젝트로서 구현되고, 이것은 그 트리의 루트 노드 오브젝트에 대한 C++ 참조를 통해 특정 IR 트리와 연관된다.

상기한 예시적인 명령 시퀀스에 있어서, 번역기(30)는 "add" 명령에 선행하는 서브젝트 명령들을 파싱하는 동안 %r2 및 %r3의 값에 대응하는 IR 트리들을 이미 발생했다. 즉, %r2 및 %r3의 값을 계산하는 서브표현식들은 IR 트리들로서 이미 표현된다. "add %rl, %r2, %r3 명령에 대한 IR 트리를 발생할 때, 새로운 'Y'노드는 %r2 및 %r3에 대한 IR 서브트리들에 대한 참조들을 포함한다.

요약 레지스터들(34)의 구현은 번역기(30) 및 번역된 코드 모두로 성분들 사이에서 분할된다. 번역기와 관련하여, 요약 레지스터는 IR 발생 과정에서 이용되는 플레이스홀더이므로, 요약 레지스터(34)는 특정 요약 레지스터(34)가 대응하는 서브젝트 레지스터(14)의 값을 계산하는 IR 트리와 연관된다. 그러한 것으로, 번역기 내의 요약 레지스터들(34)은 IR 노드 오브젝트 (즉, IR 트리)에 대한 참조를 포함하는 C++ 오브젝트로서 구현될 수 있다. 번역된 코드와 관련하여, 요약 레지스터(34)는 요약 레지스터 스토어 내의 특정 위치이며, 그 특정 위치로 및 위치로부터 서브젝트 레지스터(14) 값들이 실제 타깃

레지스터들(24)과 동기화된다. 대안으로, 어떤 값이 요약 레지스터 스토어로부터 로딩된 경우, 번역된 코드 내의 요약 레지스터(34)는 레지스터 스토어로 다시 세이브되기 전에, 번역된 코드의 실행 중 서브젝트 레지스터 값을 임시로 유지하는 타깃 레지스터(26)인 것으로 이해될 수 있다.

전술한 프로그램 번역의 예가 도 4에 도시되어 있다. 도 4는 x86 명령들의 2개의 기본 블록의 번역, 및 번역 처리에서 발생된 대응하는 IR 트리들을 도시한다. 도 4의 좌측은 번역 도중의 에뮬레이터의 실행 경로를 도시한다. 번역기(30)는 서브젝트 코드의 제 1 기본 블록(153)을 타깃 코드로 번역하고(151) 이후 그 타깃 코드를 실행한다(155). 타깃 코드가 실행을 종료하면, 제어는 에뮬레이터(157)로 복귀한다. 번역기(30)는 이후 서브젝트 코드(159)의 다음 기본 블록을 타깃 코드로 번역하고(157) 그 타깃 코드를 실행하는(161) 등의 과정을 계속한다.

서브젝트 코드의 제 1 기본 블록(153)을 타깃 코드로 번역하는(151) 처리 중, 번역기(30)는 그 기본 블록에 기초하여 IR 트리(163)를 발생한다. 이 경우, IR 트리(163)는 소스 명령 "add %eex, %edx,"으로부터 발생되고, 이 소스 명령은 플래그-영향 명령(flag-affecting instruction)이다. IR 트리(163)를 발생하는 중에, 4개의 요약 레지스터들이 이러한 명령, 즉 목적지 서브젝트 레지스터 %ecx(167), 제 1 플래그-영향 명령 패러미터(169), 제 2 플래그-영향 명령 패러미터(171), 및 플래그-영향 명령 결과(173)에 의해 정의된다. 상기 "add" 명령에 대응하는 IR 트리는 단순한 'Y'(산술 부가) 연산자(175)이며, 그 오퍼랜드들은 서브젝트 레지스터들 %ecx(177) 및 %edx(179)이다.

제 1 기본 블록의 에뮬레이션은 패러미터들 및 플래그-영향 명령들의 결과들을 저장함으로써 플래그들을 펜딩 상태로 놓는다. 플래그-영향 명령은 "add %ecx, %edx"이다. 명령의 패러미터들은 에뮬레이팅된 서브젝트 레지스터들 %ecx(177) 및 %edx (179)의 현재 값들이다. 이들 특정 서브젝트 레지스터들은 현재의 기본 블록에 의해 미리 로딩되지 않기 때문에, "@" 심볼 선행 서브젝트 레지스터 사용들((177, 179)은 서브젝트 레지스터들의 값들이 글로벌 레지스터 스토어로부터, %ecx 및 %edx에 대응하는 위치들로부터 각각 검색되는 것을 나타낸다. 부가 연산(175)의 결과는 플래그 결과 요약 레지스터(173)에 저장된다.

IR 트리가 발생된 후, 대응하는 타깃 코드는 IR에 기초하여 발생된다. 일반 IR로부터 타깃 코드를 발행하는 과정은 이 기술 분야에서 잘 알 수 있다. 플래그 결과(173) 및 플래그 패러미터들(169, 171)에 대한 것들을 포함해서 요약 레지스터들을 글로벌 레지스터 스토어에 세이브하기 위해 번역된 블록의 끝에 타깃 코드가 삽입된다. 타깃 코드가 발생된 후, 그것은 이후 실행된다(155).

서브젝트 코드(159)의 제 2 기본 블록을 번역(157)하는 도중에, 번역기(30)는 기본 블록에 기초하여 IR 트리(165)를 발생한다. IR 트리(165)는 소스 명령 "pushf"으로부터 발생되고, 이 명령은 플래그-이용 명령(flag-using 명령)이다. "pushf" 명령의 시맨틱은 모든 조건 플래그들의 값들을 스택에 저장하는 것이며, 이것은 각 플래그가 명확하게 계산될 것을 요구한다. 그러한 것으로서, 4개의 조건 플래그 값들에 대응하는 요약 레지스터들은 IR 발생, 즉 제로 플래그("ZF"; 181), 싸인 플래그(sign flag)("S17"; 183), 캐리 플래그(carry flag)("CF"; 185), 및 오버플로 플래그(overflow flag)("OF"; 187) 도중 정의된다. 노드(195)는 "unsigned less-than" 산술 비교 연산자이다. 조건 플래그들의 계산은 이전 플래그-영향 명령으로부터의 정보에 기초하며, 이 명령은 이 경우 제 1 기본 블록(153)으로부터의 "add %ecx, %edx" 명령이다. 조건 플래그 값들(165)을 계산하는 IR은 플래그-영향 명령의 결과(189) 및 패러미터들(191, 193)에 기초하고 있다. 상기한 바와 같이, "@" 심볼 선행 플래그 패러미터 라벨들은 에뮬레이터를 타깃 코드에 삽입하여 이들의 사용 전에 글로벌 레지스터 스토어로부터 이들 값들을 로딩하는 것을 나타낸다.

따라서, 제 2 기본 블록은 플래그 값들이 강제로 정규화되게 한다. 플래그 값들이 계산되고 사용된 후(타깃 코드 에뮬레이팅 "pushf" 명령에 의해), 이들은 글로벌 레지스터 스토어에 저장될 것이다. 동시에, 펜딩 플래그 요약 레지스터들(패러미터들 및 결과)은 플래그 값들이 명확하게 저장된다는(즉, 플래그들이 정규화된다는) 사실을 반영하기 위해 미정의 상태로 바뀐다.

도 5는 번역에 사용될 수 있는 수개의 상이한 타입의 IR 노드들을 발생할 수 있고, 어떻게 이들 상이한 타입들의 IR 노드들의 구현들이 번역기(30)의 프론트엔드(31), 커널(32), 및 백엔드(33) 성분들 간에 분배되는 가를 설명할 수 있는 본 발명의 최선의 실시예에 따라 형성된 번역기(30)를 도시한다. 용어 "리얼라이즈(실현하다; realize)"는 IR 발생을 일컬으며, 이것은 서브젝트 코드(10)의 서브젝트 명령들이 디코딩(즉 파싱)될 때 프론트엔드(31)에서 수행된다. 용어 "플랜트(plant)"는 타깃 코드 발생을 일컬으며, 이것은 백엔드(33)에서 수행된다.

번역 처리는 단일 서브젝트 명령의 관점에서 이하에 기술되지만, 이들 동작들은 실제로 상기한 바와 같이 동시에 서브젝트 명령들의 전체 기본 블록에 대해 발생함을 유의하라. 즉, 전체 기본 블록은 초기에 디코딩되어 IR 포레스트(forest)를 발생하고, 이후 커널(32)은 최적화들을 전 IR 포레스트에 적용한다. 끝으로, 백엔드(33)는 동시에 최적화된 IR 포레스트의 하나의 노드에 대해 타깃 코드 발생을 수행한다.

기본 블록에 대한 IR 포레스트를 발생할 때, 원하는 번역기 성능 및 소스 프로세서의 특정 아키텍쳐 및 타깃 프로세서 페어링에 기초하여, 번역기(30)는 베이스 노드들, 복합 노드들, 폴리모픽 노드들, 또는 아키텍쳐 특정 노드들(ASN), 또는 이들의 임의의 조합 중 어느 하나를 발생할 수 있다.

베이스 노드(Base Nodes)

베이스 노드들은 임의의 서브젝트 아키텍쳐의 시맨틱의 요약 표현들(즉, 표현식들, 계산들, 및 동작들)이고, 서브젝트 아키텍쳐의 시맨틱을 표현하는 데 요구되는 표준 또는 기본 노드들의 최소 세트를 제공한다. 그러한 것으로서, 베이스 노드들은 예를 들면 "add" 연산과 같은, 단순 RISC(Reduced Instruction Set Computer)와 같은 기능을 제공한다. 다른 타입의 노드들과는 대조적으로, 각각의 베이스 노드는 감소할 수 없고, 이것은 다른 IR 노드들로 추가로 내려가는 것을 막을 수 없다. 이들의 단순성으로 인해, 베이스 노드들은 또한 모든 백엔드들(33)(즉, 타깃 아키텍쳐) 상에서 타깃 명령들로 번역기(30)에 의해 용이하게 번역될 수 있다.

베이스 IR 노드들만을 이용할 경우, 번역 처리는 도 5의 상부(즉, "베이스 IR" 블록(204)을 통해 이동하는 경로들)에서 전적으로 일어난다. 프론트-엔드(31)는 디코드 블록(200)에서 서브젝트 프로그램 코드(10)로부터의 서브젝트 명령을 디코딩하고 실현 블록(202)에서 베이스 노드들로 만들어진 대응하는 IR 트리를 실현한다(발생한다). IR 트리는 이후 프론트-엔드(31)로부터 커널(32) 내의 베이스 IR 블록(204)으로 넘어가고, 여기서 최적화들이 전 IR 포레스트에 적용된다. 베이스 IR 블록(204)에 의해 최적화된 IR 포레스트는 베이스 노드들만으로 구성되기 때문에, 그것은 전체적으로 임의의 프로세서 아키텍쳐에 일반적이다. 최적화된 IR 포레스트는 이후 커널(32) 내의 베이스 IR 블록(204)으로부터 백엔드(33)로 넘어가고, 이 백엔드는 플랜트 블록(206)에서 각 IR 노드에 대해 타깃 코드 명령들을 플랜트한다(발생한다). 타깃 코드 명령들은 이후 타깃 프로세서에 의한 실행을 위해 인코드 블록(208)에 의해 인코딩된다.

상기한 바와 같이, 베이스 노드들은 모든 백엔드들(33) 상에서 타깃 명령들로 용이하게 번역되고, 번역된 코드는 전형적으로 베이스 노드들의 독점적 이용을 통해 전적으로 발생될 수 있다. 베이스 노드들의 독점적 이용이 번역기(30)를 구현하는 데 매우 빠르기는 하지만, 그것은 번역된 코드에 차선의 성능을 가져온다. 번역된 코드의 성능을 증가시키기 위해, 번역기(30)는 복합 노드들, 폴리모픽 노드들, 및 아키텍쳐-특정 노드들(ASNs)과 같은, 다른 타입의 IR 노드들을 이용함으로써 타깃 프로세서 아키텍쳐의 특징들을 이용하는 데 전용될 수 있다.

복합 노드들(Complex Nodes)

복합 노드들은 베이스 노드들보다 더 컴팩트한 표현으로 서브젝트 아키텍쳐의 시맨틱을 표현하는 일반 노드들이다. 복합 노드들은 예를 들면 "add-imm"(레지스터 및 즉치 상수(immediate constant)를 부가)와 같은, CISE(Complex Instruction Set Computer)와 같은 기능을 제공한다. 특히, 복합 노드들은 전형적으로 즉치 상수 필드들을 갖는 명령들을 표현한다. 즉치형 명령들(immediate-type instructions)은 상수 오퍼랜드 값이 "즉치" 필드 내의 명령 자체로 인코딩되는 명령이다. 즉치 필드들에 충분히 적합할 수 있을 만큼 작은 상수값들에 대해, 이와 같은 명령들은 상수를 유지하기 위해 하나의 레지스터를 사용하는 것을 회피한다. 복합 명령들에 대해, 복합 노드들은 동일한 시맨틱을 특징짓는 등가의 베이스 노드보다 훨씬 적은 노드들을 갖는 복합 명령들의 시맨틱을 표현할 수 있다. 복합 노드들은 기본적으로 동일한 시맨틱을 가진 베이스 노드 표현들로 분해될 수 있지만, 복합 노드들은 즉치형 명령들을 단일 IR 노드에 보존하는 데 유용하고, 따라서, 번역기(30)의 성능을 향상시킨다. 더욱이, 어떤 상황에서, 복합 명령들의 시맨틱은 베이스 노드들로 복합 명령들을 표현함으로써 잃을 수도 있고, 따라서 복합 노드들은 필수적으로 이와 같은 "CISC-같은" 명령들에 대해 IR 노드들을 포함하도록 베이스 노드 세트를 증가시킨다.

도 6을 참조하여, 베이스 노드들의 것에 비교되는 복합 노드를 이용함으로서 달성되는 효율의 예가 설명될 것이다. 예를 들면, MIPS 부가-측치 명령 "addirl,#10"의 시맨틱은 10을 레지스터 r1에 유지된 값에 더한다. 상수 값(10)을 레지스터에 로딩한 후 2개의 레지스터들을 부가하는 것보다는 오히려, addi 명령은 단지 상수 값 10을 명령 필드 자체로 직접 인코딩하고, 따라서, 제 2 레지스터를 이용할 필요성을 회피한다. 베이스 노드들을 엄격하게 이용하는 이들 시맨틱의 중간 표현을 발생할 때, 이러한 명령에 대한 베이스 노드 표현은 먼저 상수 값 10을 (#10) 노드(60)로부터 레지스터 노드 r(x)(61)로 로딩하고, 이후 add 노드(63)를 이용하여 레지스터 노드 r1(61) 및 레지스터 노드 r(x)(61)의 부가를 수행할 수 있다. 복합

노드 표현은 노드(70)의 부분(72)에서 상수 값(10)을 포함하는 단일의 "add to immediate"IR 노드(70) 및 레지스터 r1(74)에 대한 참조로 구성된다. 베이스 노드 시나리오에 있어서, "add to immediate" 타깃 명령을 인식하고 발생하기 위해, 백엔드(33)는, 도 6에 도시된 것과 같이, 4개의 노드 패턴을 인식할 수 있는 이디엄 인식을 수행할 필요가 있을 수 있다. 이디엄 인식이 없는 경우, 백엔드(33)는 레이지터-레지스터 부가를 수행하기 전에 상수 값 10을 레지스터로 로딩하기 위해 잉여 명령(extra instruction)을 방출할 수 있다.

복합 노드들은 벡엔드(33)에서 이디엄 인식 요구를 감소시킨다. 그 이유는 복합 노드들이 이들의 베이스 노드 등가물들보다 더 많은 시맨틱 정보를 포함하기 때문이다. 특히, 복합 노드들은 상수 오퍼랜드들의 백엔드(33) 이디엄 인식 요구를 회피한다. 비교에 의해, 즉치형 서브젝트 명령이 베이스 노드들(및 즉치형 명령들에도 포함된 타깃 아키텍쳐)로 분해되면, 이때 번역기(30)는 다중 노드 클러스터를 즉치 명령 후보로서 식별하기 위해 고가의 백엔드(33) 이디엄 인식을 요구하거나, 필요한 것보다 더 많은 타깃 레지스터들을 이용하여, 비효율적인 타깃 코드(즉, 필요한 것보다 더 많은 명령들)를 발생할 수 있다. 즉, 베이스 노드들을 단독으로 이용함으로써, 성능이 번역기(30)(이디엄 인식을 통해) 또는 번역된 코드(이디엄 인식 없는 잉여의 발생된 코드를 통해)에서 상실된다. 더욱 일반적으로, 복합 노드들이 더 컴팩트한 표현의 시맨틱 정보이기 때문에, 이들은 번역기(30)가 생성, 횡단(traverse), 및 삭제해야 할 IR 노드들의 수를 감소시킨다.

즉치형 명령들은 많은 아키텍쳐들에 공통이다. 그러므로, 복합 노드들은 이들이 아키텍쳐들의 범위에 걸쳐 재사용 가능한 점에서 일반적이다. 그러나, 모든 복합 노드는 모든 번역기의 IR 노드 세트에 존재하지 않는다. 번역기의 어떤 일반적인 특징들이 구성 가능하며, 이것은 번역기가 특정 쌍의 소스 및 타깃 구조들에 대해 컴파일링되고 있을 때, 상기 번역기 구성에 적용하지 않는 특징들은 컴파일레이션(compilation)으로부터 제외될 수 있다는 것을 의미한다. 예를 들면, MIPS MIPS (MIPS to MIPS) 번역기에 있어서, 임의의 MIP 명령들의 시맨틱과 매칭하지 않는 복합 노드들은 IR 노드로부터 제외되는데, 그 이유는 이들이 결코 이용될 수 없기 때문이다.

복합 노드들은 인 오더 트래버설(in order traversal)을 이용하여 발생된 타깃 코드의 성능을 더욱 향상시킬 수 있다. 인 오더 트래버설은 IR 트리 내의 IR 노드들이 타깃 코드로 발생되는 순서로 결정하는 수개의 다른 IR 트래버설 알고리즘들 중 하나이다. 특히, 인 오더 트래버설은 그것이 먼저 횡단될 때 각 IR 노드를 발생하고, 이것은 전 IR 트리에 걸쳐 별도의 최적화 패스의 부재로 인한 백엔드(33) 이디엄 인식을 방해한다. 복합 노드들은 베이스 노드들보다 노드 당 더 많은 시맨틱 정보를 표현하고, 따라서, 이디엄 인식의 작업 중 일부는 복합 노드들 내에 그들 자신들이 포함된다. 이것은 그것이 베이스 노드들 단독으로 사용될 때 타깃 코드 성능에 있어서 많은 페널티(penalty)를 받지 않고 번역기(30)가 인 오더 트래버설을 사용할 수 있게 한다.

번역기(30)가 복합 노드들(즉, 도 5에서 복합 IR 블록(210)을 통해 이동하는 경로들)을 발생할 경우, 번역 처리는 베이스 노드들에 대해 위에서 기술한 번역 처리과 유사하다. 단지 차이점은 복합 노드의 시맨틱과 매칭하는 서브젝트 명령들이 베이스 노드들보다는 오히려 리얼라이즈 블록(리얼라이즈 블록 202을 분리하는 점선으로 도시된 것과 같은)에서 복합 노드들로서 실현된다는 것이다. 복합 노드들은 또한 광범위의 아키텍쳐들에 걸쳐 일반적이고, 이것은 커널(32) 최적화들이 전 IR 포레스트에 적용하는 것을 가능하게 한다. 더욱이, CISC 타입 타깃 아키텍쳐들 상의 복합 노드들에 대한 타깃 코드 발생은 베이스 노드 등가물들보다 더 효율적일 수 있다.

폴리모픽 노드들(폴리모픽 Nodes)

도 5에 도시된 것과 같은 번역기(30)의 최선의 실시예는 또한 폴리모픽 중간 표현을 이용할 수 있다. 폴리모픽 중간 표현은 백엔드(33)가 특정, 성능 결정 서브젝트 명령들에 대해 타깃 아키텍쳐 특징들을 효율적으로 이용하도록 전용 코드 발생을 제공할 수 있는 메카니즘이다. 폴리모픽 메카니즘은 백엔드(33) 코드 발생 펑크션에 대한 펑크션 포인터를 포함하는 일반 폴리모픽 노드로서 구현된다. 각각의 펑크션 포인터는 특정 서브젝트 명령에 전용된다. 이러한 폴리모픽 메카니즘은 표준 프론트엔드(31) IR 발생 메카니즘을 대신하고, 표준 프론트엔드(31) IR 발생 메카니즘은 그렇지 않으면 서브젝트 명령을 베이스 또는 복합 노드들로 디코딩할 수 있다. 폴리모픽 메카니즘 없이, 이들 베이스 노드들의 발생은, 백엔드(33)에서, 차선의 타깃 코드로 될 수 있고 또는 서브젝트 명령의 시맨틱을 재구성하기 위해 고가의 이디엄 인식을 필요로 할 수 있다.

각각의 폴리모픽 펑크션은 특정 서브젝트 명령 및 타깃 아키텍쳐 펑크션 페어링에 특정이다. 폴리모픽 노드들은 이들의 펑크션에 대한 최소 정보를 커널(32)에 노출시킨다. 폴리모픽 노드들은 표현식 공유 및 표현식 폴딩(expression folding)과 같은 통상의 커널(32) 최적화들에서의 부분을 취할 수 있다. 커널(32)은 2개의 폴리모픽 노드들이 동일한지의 여부를 결정하기 위해 펑크션 포인터를 이용할 수 있다. 폴리모픽 노드들은 서브젝트 명령의 어떤 시맨틱 정보도 가지고 있지 않지만, 이와 같은 시맨틱 정보는 펑크션 포인터로부터 추정될 수 있다.

폴리모픽 노드들은 서브젝트 명령들에 대해 이용되며, 이 서브젝트 명령들은 일련의 주의깊게 선택된 타깃 명령들에 의해 표현될 수 있어, 최상의 타깃 명령들이 런-타임인지의 여부를 결정하기 위한 커널(32)에 대한 요구를 제거한다. 폴리모픽 노드들이 베이스 노드들을 이용하는 프론트엔드(31)에 의해 실현되지 않을 경우, 커널(32)은 이들 노드들을 폴리모픽 노드로서 실현하기 위해 선택할 수 있다.

더욱이, 폴리모픽 노드들은 레지스터 할당 힌트들(allocation hints)을 포함할 수 있다. 타깃 명령들이 알려져 있기 때문에, CISC 아키텍쳐들에 대해 필요로 될 수 있는 각각의 레지스터들도 알려질 수 있다. 폴리모픽 노드들은 이들의 오퍼랜드 및 결과들이 IR 구성 시간에서 선택된 레지스터들에 나타나도록 허용한다.

폴리모픽 노드들(즉, 도 5에서 폴리모픽 IR 블록(212)을 통해 이동하는 경로)을 번역기(30)가 이용하도록 하기 위해, 백엔드(33)는 서브젝트 명령 타깃 펑크션 포인터 쌍들의 리스트를 프론트엔드(31)에 제공한다. 상기 제공된 리스트 상에 있는 서브젝트 명령들은 대응하는 백엔드(33) 펑크션 포인터를 포함하는 폴리모픽 노드들로서 실현된다. 상기 리스트 상에 있지 않은 서브젝트 명령들은 상기한 바와 같이 복합 또는 기본 IR 트리들로서 실현된다. 도 5에 있어서, 백엔드(33)로부터 프론트엔드(31)로 화살표(214)로 나타낸 경로는 프론트엔드(31)에서 리얼라이즈 블록(215)에 제공되어 있는 서브젝트 명령 타깃 펑크션 포인터 쌍들의 리스트를 나타낸다. 프론트엔드(31)가 리얼라이즈 블록(215)에서 리얼라이제이션(즉, IR 노드들로의 서브젝트 명령들의 맵핑)을 수행하지만, 상기 처리는 백엔드(33)로부터 경로(214)를 통해 수신된 정보에 의해 변형된다.

커널(32)의 폴리모픽 IR 블록(212)에 있어서, 폴리모픽 노드들은 또한 일반 최적화들에 참여할 수 있는 데, 그 이유는 커널(32)이 그들의 시맨틱을 각 노드의 펑크션 포인터들로부터 추정할 수 있기 때문이다. 백엔드(33)에 있어서, 타깃 코드 발생 펑크션들을 가리키는 타깃 펑크션 포인터들은 단지 참조되지 않고(dereferenced) 실행된다. 이러한 상황은 백엔드(33)가 특정 IR 노드들을 특정 코드 발생 펑크션들을 맵핑하는 기본 노드 및 복합 노드의 경우들과는 다르다. 폴리모픽 노드들로, 폴리모픽 펑크션이 상기 노드 자체로 직접 인코딩되어, 백엔드(33)는 더욱 적은 계산을 수행한다. 도 5에 있어서, 이러한 차이는 폴리모픽 플랜트 블록(216)이 폴리모픽 IR 블록(212) 및 백엔드(33) 모두와 인접한다는 사실로 도시되어 있다(즉, 자명하지 않은 계산들을 지정하는 화살표들은 폴리모픽 IR 블록(212)과 폴리모픽 플랜트 블록(216) 사이에 도시되어 있지 않다).

예 1: 폴리모픽 IR 예

IR 에서 폴리모픽 노드들을 이용하기 위해 번역기(30)를 최적화하는 과정을 설명하기 위해, 다음과 같은 예는 제 1 베이스 노드들 및 이후 폴리모픽 노드들을 이용하여 PPC P4(PowerPC to Pentium4) 번역기에서 필요로 되는 PPC(PowerPC "SHL64" 명령 (좌 시프트, 64 bit)의 번역을 기술한다.

폴리모픽 노드들의 구현을 위해 번역기를 최적화하지 않고, PPC SHL64 명령의 번역은 기본 노드들만을 이용할 수 있다:

PPC SHL64 => 베이스 IR 다중 노드들 => P4 다중 명령들

최적화되지 않은 번역기의 프론트엔드 디코더(200)는 현재 블록을 디코딩하고 PPC SHL64 명령과 만난다. 다음에, 프론트엔드 리얼라이즈 블록(202)은 커널(32)에 다중 베이스 노드들로 구성되는 IR를 구성하도록 명령한다. 이후 커널(32)은 IR 포레스트(명령들의 현재 블록으로부터 발생된)를 최적화하고 베이스 IR 블록(204)에서의 코드 발생 순서를 결정하기 위해 오더링 트래버설(ordering traversal)을 수행한다. 다음에, 커널(32)은 각 IR 노드에 대한 코드 발생을 순서대로 수행하여, 백엔드(33)에 적절한 RISC 타입 명령들을 플랜팅하도록 명령한다. 끝으로, 백엔드(33)는 플랜트 블록(206)에 코드를 플랜팅하고 인코드 블록(208)에서 하나 이상의 타깃 아키텍쳐 명령들로 각각의 RISC 타입 명령을 인코딩한다.

성능 결정 명령들에 대한 프론트엔드(31) 및 백엔드(33)의 특수화에 의해 특정 타깃 아키텍쳐에 대해 최적화될 경우:

PPC SHL64 > 폴리 IR 단일 노드 > P4 단일/수개의 명령들

최적화된 번역기(30)의 프론트 디코더(200)는 현재 블록을 디코딩하고 PPC 명령과 만난다. 다음에, 프론트엔드 리얼라이즈 블록(202)은 커널(32)에 단일 폴리모픽 IR 노드로 구성된 IR을 구성하도록 명령한다. 단일 폴리모픽 노드가 생성되면, 백엔드(33)는 SHL64의 시프트 오퍼랜드가 특정 레지스터(P4 상의 %ecx)에 있어야 한다는 것을 알고 있다. 이러한 요구조건은 폴리모픽 노드로 인코딩된다. 이후 커널(32)은 현재 블록에 대한 IR 포레스트를 최적화하고 오더링 트래버설을 수행하여 폴리모픽 IR 블록(212)에 코드 발생 순서를 고정시킨다. 다음에, 커널(32)은 각 노드에 대한 코드 발생을 수행하고,

백에네드(33)에 적절한 RISC 타입 명령들을 플랜팅하도록 명령한다. 그러한 코드 발생 중, 폴리모픽 노드들은 베이스 노드들과는 상이하게 처리된다. 각 폴리모픽 노드는 백엔드(33)에 상주하는 전용 코드 발생기 펑크션의 호출을 야기한다. 백엔드(33) 전용 코드 발생기 펑크션은 코드를 플랜트 블록(216)에 플랜팅하고 인코드 블록(208)에서 하나 이상의 타깃 아키텍처 명령들로 각각의 서브젝트 아키텍처 명령을 인코딩한다. 발생 단계에서의 레지스터 할당 도중, 특정 레지스터 정보가 정확한 레지스터를 할당하기 위해 사용된다. 이것은 적합하지 않은 레지스터들이 할당되면 필요로 될 수도 있는 백엔드(33)에 의해 수행되는 계산을 감소시킨다.

예 2: 어려운 명령들

다음의 예는 본 발명의 번역기(30)에 의해 수행될 수도 있는 PPC MFFS 명령의 번역 및 최적화(32 비트 FPU 제어 레지스터의 64비트 범용 FPU 레지스터로의 이동)를 설명한다. 이러한 서브젝트 명령은 너무 복잡해서 베이스 노드들에 의해서는 표현될 수 없다.

최적화되지 않은 경우에 있어서, 이러한 명령은 치환 펑크션을 이용하여 번역될 수 있다. 치환 펑크션들은 표준 번역 방식을 이용해서는 번역하기 특히 어려운 서브젝트 명령들을 위한 명시된 번역들(explicit translation)이다. 치환 펑크션 번역들은 서브젝트 명령의 시맨틱을 수행하는 타깃 코드 펑크션들로서 구현된다. 이들은 표준 IR 명령 기반 번역 방식보다 훨씬 높은 실행 비용을 초래한다. 따라서, 이러한 명령을 위한 최적화되지 않은 번역 방식은 다음과 같이 된다:

PPC MFFS 명령 = > 베이스 IR 치환 펑크션 = > P4 치환 펑크션

폴리모픽 IR을 이용하는 번역기(30)에 있어서, 이와 같은 특별한 경우의 명령들은 폴리모픽 노드를 이용하여 번역된다. 폴리모픽노드'의 펑크션 포인터는 어려운 서브젝트 명령의 맞춤 번역을 공급하기 위해 백엔드(33)에 대해 더 효율적인 메카니즘을 제공한다. 따라서 동일한 명령에 대해 최적화된 번역 방식은 다음과 같이 된다:

PPC MFFS 명령 = > 단일 폴리모픽 IR 노드 = > P4 SSE2 명령들

아키텍처 특정 노드들

본 발명의 번역기(30)의 다른 최선의 실시예에 있어서, 번역기(30)는 도 5에 도시된 것과 같은, 아키텍처 특정 노드들(ASNs)을 이용하며, 이것은 특정 아키텍처들(즉, 특정 소스 아키텍처 타깃 아키텍처 조합)에 특정이다. 각각의 아키텍처 특정 노드(ASN)는 특히 특정 명령에 맞추어져 있고, 따라서 특정 아키텍처들에 특정인 ASN들을 제공한다. ASN 메카니즘을 이용할 경우, ASN 시맨틱을 포함하는 아키텍처 특정 최적화들은 구현될 수 있으므로 ASN들에 대해 동작할 수 있다.

IR 노드들은 3개의 성분들, 즉 데이터 성분, 구현 성분, 및 변환 성분까지 포함할 수 있다. 데이터 성분은 노드 자체에 고유하지 않은 어떤 시맨틱 정보(즉, 상수 즉치 명령 필드의 값)를 갖는다. 구현 성분은 코드 발생을 수행하고, 그러므로 특히 특정 아키텍처와 관련되어 있다. 변환 성분은 상기 노드를 상이한 타입의 IR 노드들, 즉 ASN 노드들 또는 베이스 노드들로 변환한다. 번역기에서의 본 발명의 주어진 구현에 있어서, 발생된 IR에서의 각각의 베이스 노드 및 ASN은 변환 성분 또는 구현 성분을 포함하지만, 둘다를 포함하지는 않는다.

각 베이스 노드는 타깃 아키텍처에 특정인 구현 성분을 가진다. 베이스 노드들이 IR 노드 계층에서 시맨틱 정보의 최소 가능 양을 인코딩하기 때문에 베이스 노드들은 변환 성분들을 가지지 않고, 따라서 베이스 노드들의 IR 노드들의 다른 타입들로의 변환은 어떠한 이점도 제공하지 않을 수 있기 때문이다. 다른 타입들의 IR 노드들로의 베이스 노드들의 임의의 이와 같은 변환은 이디엄 인식을 통해 시맨틱 정보의 재수집을 필요로 할 수 있다.

ASN의 구현 성분은 노드 아키텍처에 특정이므로, 그것은 상기 ASK에 대응하는 아키텍처 특정 명령을 발생한다. 예를 들면, MIPSLoad ASN의 구현 성분은 MIPS"ld' (로드) 명령을 발생한다. 동일한 서브젝트 및 타깃 아키텍처들(즉, 가속기로서)을 갖는 본 발명의 번역기를 이용할 때, 서브젝트 ASN들은 구현 성분들을 처리할 것이다. 상이한 서브젝트 및 타깃 아키텍처들을 갖는 번역기를 이용할 때, 서브젝트 ASN들은 변환 성분들을 가질 것이다.

예를 들면, 도 7은 MIPS-MIPS 가속기에서 본 발명의 실시예를 이용할 때 MIPS 명령에 대한 ASN을 도시한다. 프론트엔드(31)는 MIPS"addi"(add immediate) 명령(701)을 디코딩하고 대응하는 ASN, MIPS ADDI(703)을 포함하도록 IR를 발생한다. 서브젝트 및 타깃 아키텍처들은 가속기에 대해 동일하고, 따라서 변환 성분 "CVT"(707)은 정의되지 않는다. 구현 성분 "IMPL"(705)은 코드 발생 패스에서 레지스터 할당 차이들의 영향을 받는, 동일한 MIPS "addi" 명령(709)을 발생하도록 정의된다.

도 8은 MIPS X86 번역기에서 본 발명의 실시예를 사용할 때 동일한 MIPS 명령에 대한 IR에서의 ASN들을 도시한다. 프론트엔드(31)는 MIPS "addi"서브젝트 명령을 디코딩하고 대응하는 서브젝트 ASN, MIPS ADDI(801)를 발생한다. 소스 및 타깃 아키텍쳐들은 이러한 번역기에 대해 서브젝트 ASN(801)의 구현 성분(803)은 따라서 정의되지 않는다. MIPS ADDI의 변환 성분(805)은 전용 변환 성분이고, 이 전용 변환 성분은, 서브젝트 ASN(801)을 타깃 ASN(807)으로 변환한다. 비교에 의해, 일반 변환 성분은 서브젝트 ASN(801)를 베이스 노드 표현으로 변환할 수 있다. MIPS ADDI 노드(801)의 타깃 ASN 표현은 단일 X86 ADDI 노드(807)이다. 타깃 ASN(807)의 변환 성분(811)은 정의되지 않는다. 타깃 ASN(807)의 구현 성분(809)은 X86 명령 "ADD $EAX,#10"의 경우에, 타깃 명령(813)을 발생한다.

번역기(30)가 ASN들을 이용하고 있을 경우, 모든 서브젝트 명령들은 서브젝트 특정 ASN들로서 실현된다. 도 5에 있어서, 프론트엔드가 블록(200)을 디코딩하고, ASN이 블록(218)을 실현하고, 서브젝트 ASN 블록(220)이 서로 인접한다는 사실은 ASN들이 프론트엔드(31)에 의해 정의되고 상기 실현이 사소한 것이라는 사실을 나타내는 데, 그 이유는 서브젝트 명령 타입들 및 서브젝트 ASN들 타입들 간에 일대일 관계가 있기 때문이다. 프론트엔드(31)는 서브젝트 ASN들의 시맨틱을 이해하고 서브젝트 ASN들에 대해 동작할 수 있는 서브젝트 특정 최적화들을 포함한다. 즉, 서브젝트 코드는 초기에 서브젝트 ASN들로 전부 구성된 IR 포레스트로서 실현되고, 그것에는 이후 서브젝트 최적화가 적용된다.

디폴트에 의해, 서브젝트 ASN는 베이스 노드들의 IR 트리를 발생하는 일반 변환 성분을 가진다. 이것은 새로운 서브젝트 아키텍쳐에 대한 지원이 일반 IR 노드들을 이용하여 신속하게 구현되게 허용한다. 서브젝트 ASN들은 도 5의 ASN 베이스 IR 블록(222) 및 플랜트 블록(206)을 통해 연장하는 경로를 통해 베이스 노드들로서 실현되고, 이것은 위에 상세하게 기술된 것과 같이 다른 베이스 노드들과 유사한 방식으로 타깃 코드로 번역된다.

성능에 중대한 서브젝트 명령들에 있어서, 대응하는 서브젝트 ASN 노드들은 전용 변환 성분들을 제공하고, 이 전용 변환 성분들은 타깃 ASN 노드들의 IR 트리들을 발생한다. 전용 변환 성분을 구현하는 지의 여부에서 고려되는 인자들은 (1) 타깃 아키텍쳐 특징들이 베이스 노드 번역에서 상실될 수도 있는 특히 효율적인 번역을 준비하는지의 여부, 및 (2) 서브젝트 명령이 성능에 중요한 영향을 줄 수 있는 그와 같은 주파수로 생기는지의 여부를 포함한다. 이들 전용 변환 성분들은 서브젝트 타깃 아키텍쳐 쌍에 특정이다. 타깃 ASN들(정의에 의해 타깃과 동일한 아키텍쳐을 가지는)은 구현 성분들을 포함한다.

전용 변환 성분들을 구현할 때, 대응하는 서브젝트 ASN 노드들은 서브젝트 ASN들을 타깃 ASN들로 타깃 ASN 블록(224)을 통해 변환하는 타깃 변환 성분들을 제공한다. 타깃 ASN 구현 성분은 이후 타깃 ASN 플랜트 블록(226)에서 코드 발생을 수행하도록 불어내어 진다. 각 타깃 ASN는 하나의 특정 타깃 명령에 대응하므로, 타깃 ASN으로부터 발생된 코드는 단지 ASN이 인코딩하는 대응하는 타깃 명령이다. 그러한 것으로서, 타깃 ASN들을 이용하는 코드 발생은 계산적으로 최소이다(백엔드(33) 내의 타깃 ASN 블록(224) 및 인코드 블록(208) 모두와 인접하고 있는 타깃 ASN 플랜트 블록(226)의 설명에 의해 도 5에 반영되어 표현된, 중대한 계산들을 지정하는 화살표들은 이들 성분들 사이에 도시되어 있지 않다). 더욱이, IR 횡단, 변환, 및 코드 발생 처리들은 모두 커널(32)에 의해 제어된다.

도 9는 ASN 메카니즘을 이용하는 본 발명의 번역기의 최선의 실시예에 따라 수행되는 번역 처리를 도시한다. 프론트엔드(31)에 있어서, 번역기는 단계 903에서 서브젝트 코드(901)를 서브젝트 ASN들(904)로 디코딩한다. 번역기는 서브젝트 ASN들로 만들어진 IR 트리에 대해 단계 905에서 서브젝트 특정 최적화들을 수행한다. 각 서브젝트 ASN(904)은 이후 서브젝트 ASN 변환 성분을 불어옴으로써 단계 907에서 타깃 호환 IR 노드들(타깃 ASN들 911)로 변환된다. 디폴트에 의한 일반 변환 성분들을 가진 서브젝트 ASN 노드들은 베이스 노드들(909)로 변환된다. 백엔드(925)에 의해 제공된, 전용 변환 성분들을 가진 서브젝트 ASN 노드들은 타깃 ASN들(911)로 변환된다. 따라서 변환은 베이스 노드들(909) 및 타깃 ASN들(911) 모두를 포함하는 혼합된 IR 포레스트(913)을 생성한다. 커널(32)에 있어서, 번역기는 혼합된 IR 포레스트(913) 내의 베이스 노드들에 대해 단계 915에서 일반 최적화들을 수행한다. 번역기는 이후 혼합된 IR 포레스트(913) 내의 타깃 ASN들에 대해 단계 916에서 타깃 특정 최적화들을 수행한다. 끝으로, 코드 발생은 단계 917에서 혼합된 트리(베이스 노드들 및 타깃 ASN 노드들 모두는 구현 성분들을 가짐)에서 각 노드의 구현 성분을 불러오고, 이 단계 917은 이후 타깃 코드(919)를 발생한다.

코드 가속기의 특수한 경우에 있어서, 서브젝트 및 타깃 아키텍쳐들은 모두 동일하다. 이러한 시나리오에서, 서브젝트 ASN들은 번역 내내 존속한다. 프론트엔드(31)에 있어서, 디코딩은 서브젝트 명령들로부터 서브젝트 ASN들을 발생한다. 커널(32)에 있어서, 서브젝트 ASN들은 아키텍쳐 특정 최적화들을 통과한다. 코드 발생은 서브젝트 ASN 구현 성분들을 불러와 대응하는 명령들을 발생한다. 그러한 것으로서, 코드 가속기에 있어서, ASN들의 이용은 최적화들에 의해 증가될 수 있는, 1:1의 최소 서브젝트 대 타깃 명령 변환비를 보장함으로써, 코드 파열(code explosion)을 방지한다.

본 발명의 번역기의 여러 실시예들이 특정 번역기 애플리케이션들(즉, 특정 서브젝트 아키텍쳐 타깃 아키텍쳐 쌍들)을 위해 구성될 수 있다. 그러한 것으로서, 본 발명의 번역기는 임의의 서브젝트 아키텍쳐 상에서 실행하도록 설계된 서브젝트 코드를 임의의 타깃 아키텍쳐 상에 실행 가능한 타깃 코드로 변환하도록 구성가능하다. 다중 번역기 애플리케이션에 걸쳐, 각 베이스 노드는 각각 지원되는 타깃 아키텍쳐에 대해 하나씩 다중 구현 성분들을 가진다. 의도된 특정 구성(즉, 조건부 컴필레이션)은 IR 노드들 중 어느 것을 그리고 이들 노드들 중 어느 성분들을 특정 번역기 애플리케이션에 포함하는가를 결정한다.

본 발명의 최선의 실시예에서의 ASN들의 사용은 복수의 유리한 이점들을 제공한다. 먼저, 스크래치(scratch)로 만들어진 번역기 제품은 서브젝트 명령들의 일반 IR 구현들을 이용하여 신속하게 개발될 수 있다. 둘째, 기존 번역기 제품들은 성능(미리 알려지거나 경험적으로 결정된)에 중요한 서브젝트 명령들에 대한 타깃 특정 변환 성분들을 구현함으로써, 점진적으로 증가될 수 있다. 셋째, 더 많은 번역기 제품들이 개발되기 때문에, ASN 노드들(및 구현된 기능)의 라이브러리(library)는 시간이 지남에 따라 성장하고, 그래서 미래의 번역기 제품들은 신속하게 구현되거나 최적화될 수 있다.

본 발명의 이러한 실시예는 어느 서브젝트 명령들이 가치있는 최적화를 하고 있는가를 픽크 및 선택하기 위해 백엔드 구현한다(타깃 전용 변환 성분들을 정의함으로써). 일반 변환 성분은 ASN-기반 번역기가 신속하게 개발되게 허용하고, 한편 전용 변환 성분들은 성능 중요 명령들(performance critical instructions)이 선택적으로 및 점진적으로 최적화되게 허용한다.

예 3: ASN을 이용한 어려운 명령들

상기 예 2의 PowerPC SHL64 명령으로 돌아가서, ASN들을 이용하는 번역기(30)는 다음과 같은 단계들을 수행한다. 프론트엔드 디코더(200)는 현재 블록을 디코딩하고 PowerPC SHL64 명령과 만난다. 프론트엔드(31)는 이후 명령, 상기 SHL64 PPC P4에 대해 단일 ASN을 실현하다. 커널(32)은 이후 명령들의 현재 블록에 대한 IR을 최적화하고, 코드 발생을 위한 준비시 IR의 순서화 횡단(ordering traversal)를 수행한다. 커널(32)은 이후 구현 성분의 요소인, 각각의 특정 ASN 노드 코드 발생기 펑크션을 불러옴으로서 ASN 노드들에 대한 코드 발생을 수행한다. 백엔드(33)는 이후 서브젝트 아키텍쳐(PPC) 명령들을 하나 이상의 타깃 아키텍쳐 (P4) 명령들로 인코딩한다.

MIPS 예들

도 10, 11, 12를 참조하여, 베이스 IR 노드들, MIPS-MIPS ASN IR 노드들, 및 MIPS-X86 ASN IR 노드들을 이용하여 동일 MIPS 명령 시퀀스로부터 발생된 상이한 IR 트리들을 도시하는 예들이 각각 도시되어 있다. 예 MIPS 서브젝트 명령 시퀀스(상 즉치, 이후 비트와이즈- 또는 즉치 로드; load upper immediate, then bitwise-or immediate)의 시맨틱은 32 비트 상수값 OxI2345678을 서브젝트 레지스터 "al"로 로딩하는 것이다.

도 10에 있어서, 이진 디코더(300)는 서브는 서브젝트 코드를 개개의 서브젝트 명령들로 디코딩(파싱)하는 번역기(30)의 프론트엔드(31) 성분이다. 서브젝트 명령들 디코딩된후, 이들은 베이스 노드들(302)로서 실현되고 명령들의 현재 블록을 위해 작동하는 IR 포레스트에 부가된다. IR 관리자(304)는 IR 발생 중 작동하는 IR 포레스트를 유지하는 번역기(30)의 부분이다. IR 관리자(304)는 요약 레지스터들 및 이들의 연관 IR 트리들로 구성된다(IR 포레스트의 루트들은 요약 레지스터들이다). 예를 들면, 도 10에 있어서, 요약 레지스터 "a1"(306)은 5개의 노드들의 IR 트리(308)의 루트이며, 이것은 현재의 블록 작동 IR 포레스트의 부분이다. C++로 구현되는 번역기(30)에 있어서, IR 관리자(304)는 요약 레지스터 오브젝트의 세트(또는 IR 노드 오브젝트들에 대한 참조들)를 포함하는 C++ 오브젝트로서 구현될 수 있다.

도 10은 베이스 노드들만을 이용하여 MIPS-X86 번역기에 의해 발생된 IR 트리(308)를 도시한다. MIPS-LUI 명령(310)은 이 경우에 모두 상수들인, 2개의 오퍼랜드 노드들(316, 318)을 갖는 "SHL" (shift left) 베이스 노드(314)를 실현한다. MIPS LUI 명령(310)의 시맨틱은 상수 비트들에 의해 상수값(Ox1234)을 좌측으로 이동시키는 것이다. MIPS ORI 명령(312)은 2개의 오퍼랜드 노드들(314, 322), SHL 노드(314)의 결과 및 상수값을 갖는 "ORI" (비트와이즈 또는 즉치) 베이스 노드(320)를 실현한다. MIPS ORI 명령(312)의 시맨틱은 상수값(Ox5678)으로 기존 레지스터 콘텐츠의 비트와이즈-오어(bitwise-or)를 수행하는 것이다.

최적화되지 않은 코드 발생기에 있어서, 베이스 노드들은 로드 즉치이외의 즉치형 연산자들을 포함하지 않고, 그래서 각 상수 노드는 로드 즉치 명령의 발생을 일으킨다. 그러므로 최적화되지 않은 베이스 노드 번역기는 이러한 서브젝트 명령들 시퀀스를 위한 5개의 RISC 타입 동작들(로드, 로드, 시프트, 로드, 오어)을 필요로 한다. 백엔드(33) 이디엄 인식은 즉치형

타깃 명령들(즉, 시프트 즉치 및 오어 즉치)을 발생하기 위해, 이들의 페어런트 노드들을 갖는 상수 노드들을 합체함으로써, 이러한 수를 5에서 2로 감소시킨다. 이것은 타깃 명령들의 수를 2로 감소시키지만, 증가된 번역 비용때문에, 코드 발생기에서 이디엄 인식을 수행한다.

IR에서 복합 노드들을 이용하면 즉치형 IR 노드들을 실현할 수 있는 데, 이것은 백엔드(33)에서 이디엄 인식을 수행할 필요성을 제거하고 코드 발생기의 번역 비용을 감소시킨다. 복합 노드들은 더 많은 원 서브젝트 명령들의 시맨틱을 보존하며, 약간의 IR 노드들이 실현되어, 노드 발생의 번역 코스트도 복합 노드들을 이용할 때 감소된다.

도 11은 ASN들을 이용하는 MIPS X86 (MIPS to X86) 번역기에 의해 발생된 IR 트리를 도시한다. 서브젝트 명령들이 이진 디코더(300)에 의해 디코딩된 후, 이들은 MIPS_X86 ASN 노드들(330)로서 실현되며, 이것은 이후 현재 블록을 위한 동작 IR 포레스트에 부가된다. 먼저, MIPS_X86_LUI ASN 노드는 ASN 변환 성분에 의해 X86 32-비트 상수 노드(332)로 변환된다. 둘째, MIPS_X86_ORI ASN 노드는 이전 X86 상수 노드(상수 폴딩; constant folding)에 의해 즉시 폴딩되는 X86 ORI 노드를 생성하여, 단일 X86 32-비트 상수 노드(334)로 된다. 이러한 노드(334)는 단일 X86 로드 상수 명령, "mov %eax, $0x12345678"로 인코딩된다. 알 수 있는 것과 같이, ASN 노드들은 베이스 노드 예보다 더 적은 수의 노드들로 되고, 따라서 번역 비용을 감소시키고 더 양호한 타깃 코드를 제공한다.

도 12는 ASN들을 이용하는 MIPS-MIPS 번역기(즉, MIPS 가속기)에 의해 발생된 IR 트리를 도시한다. 서브젝트 명령들(310, 312)이 2진 디코더(300)에 의해 디코딩된 후, 이들은 MIPS_MIPS ASN 노드들(340)로서 실현되며, 이것은 이후 현재 블록을 위한 동작 IR 포레스트에 부가된다. 소스 및 타깃 아키텍쳐들은 MIPS-MIPS 번역기에 대해 동일하기 때문에, MIPS_MIPS_LUI 및 MIPS_MIPS_ORI ASN 노드들(340)은 널(미정의) 변환 성분들을 가진다. 그러한 것으로서, 서브젝트 명령들과 코드를 발생하기 위해 사용되는 최종 IR 노드들 사이에 직접 대응(direct correspondence)이 있다. 이것은 최적화들이 적용되기 전에도, 1: 1 서브젝트 대 타깃 명령 번역비를 보장한다. 즉, ASN 노드들은 동일-동일 번역기들(same-same translators)(가속기들)에 대한 코드 파열을 제거한다. ASN 노드들은 또한 16 비트 상수 노드들이 공유되게 허용하고, 이것은 MIPS 플랫폼 상에서 연속하는 메모리 액세스들의 효율적인 번역을 위해 유용하다.

명령들의 기본 블록들은 동시에 하나의 서브젝트 명령으로 번역된다. 각각의 서브젝트 명령은 IR 트리를 형성되게 한다(실현한다). 주어진 명령에 대한 IR 트리가 생성된 후, 그것은 이후 현재 블록을 위한 동작 IR 포레스트에 통합된다. 동작 IR 포레스트의 루트들은 요약 레지스터들이며, 이것은 서브젝트 레지스터들 및 서브젝트 아키텍쳐의 다른 특징들에 대응한다. 최후 서브젝트 명령이 디코딩되고, 실현되고, 그 IR 트리가 동작 IR 포레스트와 통합되면, 상기 블록에 대한 IR 포레스트는 완전하다.

도 12에 있어서, 제 1 서브젝트 명령(310)은 "lui al, OxI234"이다. 이러한 명령(310)의 시맨틱은 상수값 OxI234을 서브젝트 레지스터 "al"(342)의 상측 16 비트로 로딩하는 것이다. 이러한 명령(310)은 OxI234의 즉치 필드 상수값을 갖는 MIPS_MIPS_LUI 노드(344)를 실현한다. 번역기는 MIPS_MIPS_LUI 노드(344)로 포인팅하기 위해 요약 레지스터 "al"(342)를 설정함으로써 동작 IR 포레스트에 이러한 노드를 부가한다.

도 12의 동일한 예에 있어서, 제 2 서브젝트 명령(312)는 44ori al, al, Ox5678"이다. 이러한 명령(312)의 시맨틱은 서브젝트 레지스터 4'al"(342)의 현재 콘텐츠로 상수값 Ox5678의 비트와이즈-오어를 수행하고 그 결과를 서브젝트 레지스터 4'al"(346)에 저장하는 것이다. 이러한 명령(312)은 Ox5678의 즉치 필드 상수값과 함께 MIPS MIPS ORI 노드(348)를 실현한다. 요약 레지스터 "al"(342)(서브젝트 명령의 소스 레지스터)에 의해 지금 포인팅되는 IR 트리로 포인팅하기 위해 ORI 노드를 먼저 설정하고 나서, ORI 노드(348)로 포인팅하기 위해 요약 레지스터 "al"(346)(서브젝트 명령의 목적지 레지스터)를 설정함으로써, 번역기는 이러한 노드를 동작 IR 포레스트에 부가한다. 즉, 요약 레지스터(342)(즉, 기존 LUI 노드)에 뿌리를 둔 기존 "al" 트리는 ORI 노드(348)의 서브트리(350)로 되고, 이후 ORI 노드(348)는 새로운 al 트리가 된다. 오래된 "al" 트리(LUI 후 ORI 전)는 요약 레지스터(342)에 뿌리를 두고 선(345)에 의해 연결된 것처럼 도시되어 있고, 한편 현재의 "al"트리 (ORI 후)는 요약 레지스터(346)에 뿌리를 두고 있다.

상기한 바로부터 알 수 있는 것과 같이, 본 발명에 따라 만들어진 개선된 프로그램 코드 변환 장치는 최적 수준의 성능 및 번역속도를 번역된 타깃의 효율과 균형을 맞추면서 임의의 서브젝트 및 타깃 프로세서 아키텍쳐 쌍을 구성 가능하다. 더욱이, 변환과 관련된 서브젝트 및 타깃 컴퓨팅 환경들의 아키텍쳐들에 의존하여, 본 발명의 프로그램 코드 변환 장치는 그 중간 표현에 베이스 노드들, 복합 노드들, 폴리모픽 노드들, 및 아키텍쳐 특정 노드들의 조합을 이용함으로써 일반 및 특정 변환 특징들의 혼합 설계로 설계될 수 있다.

본 발명의 개선된 프로그램 코드 변환 장치의 상이한 구조들이 상기 실시예들의 각각에서 개별적으로 설명된다. 그러나, 본원에 기술된 각 실시예의 개별 양태들은 본원에 기술된 다른 실시예들과 조합될 수 있다는 것은 본 발명의 발명자들의

전체 의도이다. 예를 들어, 본 발명에 따라 만들어진 번역기는 여러 IR 타입들의 혼합 최적화들(hybrid optimizations)을 포함할 수 있다. 이 기술 분야에서 숙련된 사람들은 앞에 기재된 최선의 실시예의 여러 개조물 및 변형예들이 본 발명의 범위 및 사상을 벗어나지 않고 구성될 수 있다는 것을 이해할 수 있을 것이다. 그러므로, 첨부된 청구항들의 범위 내에서, 본 발명은 본원에 특별히 기술된 것과는 다르게 실시될 수 있다는 것이 이해될 것이다.

몇몇 최선의 실시예들이 도시되고 기술되었지만, 이 기술 분야에서 숙련된 사람들은 여러 변경예들 및 변형예들이 첨부 청구항들에 정의된 것과 같이, 본 발명의 범위를 벗어나지 않고 만들어 질 수 있다는 것을 이해할 수 있을 것이다.

주의해야 할 점은 이 출원과 관련하여 이 명세서와 동시에 또는 이전에 제출되고 이 출원과 관련하여 이 명세서와 함께 공개되고 이 명세서와 함께 공개 열람에 오픈된 모든 서류들 및 문서들에 있고, 모든 이와 같은 서류들 및 문서들의 내용들은 참조로 본원에 포함된다.

이 명세서(첨부 청구항들, 요약 및 도면들을 포함)에 개시된 모든 특징들 및/또는 그렇게 개시된 임의의 방법 또는 공정들의 모든 단계들은 이와 같은 특징들 및/또는 단계들의 적어도 일부가 상호 배타적인 조합들을 제외하고, 어떠한 조합으로도 조합될 수 있다.

이 명세서(첨부 청구항들, 요약 및 도면들을 포함)에 개시된 각각의 특징은 달리 명확하게 설명되지 않는다면, 동일, 등가 또는 유사 목적을 달성하는 다른 특징들에 의해 대체될 수 있다. 따라서, 달리 명확하게 설명되지 않는다면, 개시된 각 특징은 등가 또는 유사 특징들만의 일반 시리즈의 일 예이다.

본 발명은 상기 실시예(들)의 상세들로 제한되지 않는다. 본 발명은 이 명세서(첨부 청구항들, 요약 및 도면들을 포함)에 개시된 특징들의 어느 신규한 하나 또는 임의의 신규의 조합, 또는 그렇게 개시된 임의의 방법 또는 공정의 어느 신규의 하나 또는 어느 신규의 조합으로 확대된다.

(57) 청구의 범위

청구항 1.

프로그램 코드의 중간 표현(intermediate representation; IR)을 발생하는 방법에 있어서,

상기 프로그램 코드 내의 명령들을 디코딩하는 단계;

복수의 가능한 타입들의 IR 노드들 중에서 적어도 하나의 타입의 IR 노드들을 포함하도록 상기 디코딩된 프로그램 코드의 중간 표현(IR)을 발생하는 단계;

상기 디코딩된 프로그램 코드 내의 각각의 개별 명령에 대해 상기 중간 표현으로 어느 타입의 IR 노드들을 발생할 것인가를 결정하는 단계를 포함하고,

상기 중간 표현(IR)의 상기 IR 노드들은 상기 프로그램 코드에 의해 수행되는 표현식들(expressions), 계산들, 및 동작들의 요약 표현들인, 프로그램 코드의 중간 표현 발생 방법.

청구항 2.

제 1 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 베이스 노드들 및 복합 노드들을 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 3.

제 2 항에 있어서,

베이스 노드들은 상기 프로그램 코드를 실행하는 임의의 서브젝트 아키텍쳐의 최상의 기본 시맨틱(semantics)을 표현하므로, 상기 베이스 노드들의 시맨틱은 더 간단한 시맨틱을 표현하는 다른 노드들로 분해될 수 없는, 프로그램 코드의 중간 표현 발생 방법.

청구항 4.

제 3 항에 있어서,

베이스 노드들은 복수의 가능한 서브젝트 아키텍쳐들에 걸쳐 일반적인, 프로그램 코드의 중간 표현 발생 방법.

청구항 5.

제 3 항 또는 제 4 항에 있어서,

복합 노드들은 베이스 노드 표현들의 것보다 상기 프로그램 코드의 복합 명령들의 상기 시맨틱의 더 컴팩트한 표현을 제공하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 6.

제 5 항에 있어서,

상기 복합 노드들은 상수 오퍼랜드 값이 중간 필드에서 중간 타입 명령 자체로 인코딩되는 상기 중간 타입 명령들을 표현하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 7.

제 5 항 또는 제 6 항에 있어서,

복합 노드는 상기 디코딩된 프로그램 코드로 명령의 동일한 시맨틱을 표현하기 위해 복수의 베이스 노드들로 분해될 수 있는, 프로그램 코드의 중간 표현 발생 방법.

청구항 8.

제 5 항 내지 제 7 항 중 어느 한 항에 있어서,

상기 프로그램 코드는 서브젝트 아키텍쳐에 의해 실행되도록 설계되고, 상기 방법은 상기 서브젝트 아키텍쳐에 대해 대응적으로 구성 가능한 이들 특징들에 대해서만 복합 노드들을 발생하는 단계를 더 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 9.

제 2 항 또는 제 2 항에 종속하는 청구항 중 어느 한 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 폴리모픽 노드들(polymorphic nodes)을 더 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 10.

제 9 항에 있어서,

상기 프로그램 코드는 서브젝트 아키텍쳐에 대해 실행하도록 설계된 서브젝트 코드이고 타깃 아키텍쳐에 대해 실행하기 위한 타깃 코드로 동적으로 번역되고, 상기 방법은:

폴리모픽 노드들을 포함하도록 상기 중간 표현을 발생하는 단계를 더 포함하고, 폴리모픽 노드들은 상기 서브젝트 코드 내의 특정 명령에 대해 특정인 상기 타깃 아키텍쳐의 펑크션에 대한 펑크션 포인터(function pointer)를 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 11.

제 10 항에 있어서,

상기 방법은 상기 타깃 아키텍쳐의 상기 특징들이 특정 서브젝트 명령의 상기 시맨틱으로 하여금 베이스 노드들로서 실현되면 상실되게 할 수 있을 경우 폴리모픽 노드들을 발생하는 단계를 더 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 12.

제 10 항 또는 제 11 항에 있어서,

각 폴리모픽 노드는 상기 서브젝트 코드 내의 특정 명령 및 상기 타깃 아키텍쳐의 펑크션의 조합에 특정인, 프로그램 코드의 중간 표현 발생 방법.

청구항 13.

제 10 항 내지 제 12 항 중 어느 한 항에 있어서,

상기 IR 노드들의 타입을 결정하는 단계는 폴리모픽 노드들로서 실현될 폴리모픽 명령들의 리스트 상의 명령에 대응하는 서브젝트 코드 내의 명령을 식별하는 단계를 더 포함하고;

서브젝트 명령이 상기 폴리모픽 명령들의 리스트 상의 명령들에 대응할 경우, 상기 IR를 발생하는 단계는 상기 폴리모픽 명령들의 리스트 상의 서브젝트 명령들에 대응하는 이들 서브젝트 명령들에 대해서만 폴리모픽 노드들을 발생하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 14.

제 1 항 내지 제 13 항 중 어느 한 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 베이스 노드들 및 아키텍쳐 특정 노드들을 더 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 15.

제 14 항에 있어서,

상기 프로그램 코드는 서브젝트 아키텍쳐에 대해 실행하도록 설계된 서브젝트 코드이고 타깃 아키텍쳐에 대해 실행하기 위한 타깃 코드로 동적으로 번역되고, 상기 방법은:

서브젝트 아키텍쳐 및 타깃 아키텍쳐의 특정 조합에 특정인 아키텍쳐 특정 노드들을 포함하도록 상기 중간 표현을 발생하는 단계를 더 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 16.

제 15 항에 있어서,

상기 중간 표현 발생 단계는:

처음에는 상기 서브젝트 코드 내의 상기 명령들 모두를 서브젝트 아키텍쳐 특정 노드들로서 표현하는 단계로서, 각각의 서브젝트 아키텍쳐 특정 노드는 상기 서브젝트 코드 내의 개개의 명령에 대응하는, 상기 표현하는 단계;

상기 서브젝트 코드 내의 명령이 타깃 아키텍쳐 전용 변환 펑크션을 제공하기 위한 것 중 하나인지의 여부를 결정하고, 서브젝트 아키텍쳐 특정 노드들을 타깃 아키텍쳐 전용 변환 펑크션을 제공하도록 결정된 이들 명령들을 위한 타깃 아키텍쳐 특정 노드들로 변환하는 단계; 및

타깃 아키텍쳐 전용 코드 발생 펑크션을 제공하는 것으로서 식별되지 않은 나머지 서브젝트 아키텍쳐 특정 노드들로부터 베이스 노드들을 발생하는 단계를 더 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 17.

제 16 항에 있어서,

상기 타깃 아키텍쳐에 전용되는 상기 타깃 아키텍쳐 특정 노드들로부터 대응하는 타깃 코드를 발생하는 단계를 더 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 18.

제 15 내지 제 17 항 중 어느 한 항에 있어서,

상기 타깃 아키텍쳐에 전용되지 않은 상기 베이스 노드들로부터 대응하는 타깃 코드를 발생하는 단계를 더 포함하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 19.

제 1 항 내지 제 18 항 중 어느 한 항의 방법을 수행하는 프로그램 코드를 포함하는 컴퓨터 판독 가능 기록 매체.

청구항 20.

타깃 프로그램 코드로의 서브젝트 프로그램 코드의 번역 중 다음의 단계들을 수행하도록 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드의 형태로 그 위에 상주하는 번역기 소프트웨어를 가진 컴퓨터 판독 가능 저장 매체에 있어서,

상기 서브젝트 프로그램 코드 내의 명령들을 디코딩하는 단계;

복수의 가능한 타입들의 IR 노드들 중에서 적어도 하나의 형태의 IR 노드들을 포함하도록 상기 디코딩된 서브젝트 프로그램 코드의 중간 표현(IR)을 발생하는 단계;

상기 디코딩된 서브젝트 프로그램 코드 내의 각각의 개별 명령에 대해 상기 중간 표현으로 어느 타입의 IR 노드들을 발생할 것인가를 결정하는 단계로서, 상기 중간 표현(IR)의 상기 IR 노드들은 상기 프로그램 코드에 의해 수행되는 표현식들, 계산들, 및 동작들의 요약 표현들인, 상기 결정하는 단계; 및

상기 중간 표현(IR)을 이용하여 타깃 프로그램 코드를 발생하는 단계를, 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 21.

제 20 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 베이스 노드들 및 복합 노드들을 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 22.

제 21 항에 있어서,

베이스 노드들은 상기 프로그램 코드를 실행하는 임의의 서브젝트 아키텍처의 최상의 기본 시맨틱을 표현하므로, 상기 베이스 노드들의 시맨틱은 더 간단한 시맨틱을 표현하는 다른 노드들로 분해될 수 없는, 컴퓨터 판독 가능 저장 매체.

청구항 23.

제 22 항에 있어서,

베이스 노드들은 복수의 가능한 서브젝트 아키텍처들에 걸쳐 일반적인, 컴퓨터 판독 가능 저장 매체.

청구항 24.

제 22 항에 있어서,

복합 노드들은 베이스 노드 표현들의 것보다 상기 프로그램 코드의 복합 명령들의 상기 시맨틱의 더 컴팩트한 표현을 제공하는, 컴퓨터 판독 가능 저장 매체.

청구항 25.

제 24 항에 있어서,

상기 복합 노드들은 상수 오퍼랜드 값이 중간 필드에서 중간 타입 명령 자체로 인코딩되는 상기 중간 타입 명령들을 표현하는, 컴퓨터 판독 가능 저장 매체.

청구항 26.

제 24 항에 있어서,

복합 노드는 상기 디코딩된 프로그램 코드로 명령의 동일한 시맨틱을 표현하기 위해 복수의 베이스 노드들로 분해될 수 있는, 컴퓨터 판독 가능 저장 매체.

청구항 27.

제 24 항에 있어서,

상기 서브젝트 프로그램 코드는 서브젝트 아키텍처에 의해 실행되도록 설계되고, 상기 방법은 상기 서브젝트 아키텍처에 대해 대응적으로 구성 가능한 이들 특징들에 대해서만 복합 노드들을 발생하는 단계를 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 28.

제 21 항 내지 제 27 항 중 어느 한 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 폴리모픽 노드들을 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 29.

제 28 항에 있어서,

상기 서브젝트 프로그램 코드는 서브젝트 아키텍처에 대해 실행하도록 설계된 서브젝트 코드이고 타깃 아키텍처에 대해 실행하기 위한 타깃 코드로 동적으로 번역되고,

상기 번역기 소프트웨어는:

폴리모픽 노드들을 포함하도록 상기 중간 표현을 발생하는 단계로서, 폴리모픽 노드들은 상기 서브젝트 코드 내의 특정 명령에 대해 특정인 상기 타깃 아키텍처의 펑크션에 대한 펑크션 포인터를 포함하는, 상기 발생하는 단계를 수행하도록 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 30.

제 29 항에 있어서,

상기 번역기 소프트웨어는 상기 타깃 아키텍처의 상기 특징들이 특정 서브젝트 명령의 상기 시맨틱로 하여금 베이스 노드들로서 실현되면 상실되게 할 수 있을 경우 폴리모픽 노드들을 발생하도록 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 31.

제 29 항에 있어서,

각 폴리모픽 노드는 상기 서브젝트 코드 내의 특정 명령 및 상기 타깃 아키텍처의 펑크션의 조합에 특정인, 컴퓨터 판독 가능 저장 매체.

청구항 32.

제 29 항에 있어서,

상기 IR 노드들의 타입을 결정하기 위해 컴퓨터에 의해 실행 가능한 상기 컴퓨터 판독 가능 코드는:

폴리모픽 노드들로서 실현될 폴리모픽 명령들의 리스트 상의 명령에 대응하는 서브젝트 코드 내의 명령을 식별하고;

서브젝트 명령이 상기 폴리모픽 명령들의 리스트 상의 명령들에 대응할 경우, 상기 폴리모픽 명령들의 리스트 상의 서브젝트 명령들에 대응하는 이들 서브젝트 명령들에 대해서만 폴리모픽 노드들을 발생하는 것을 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 33.

제 20 항 내지 제 32 항 중 어느 한 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 베이스 노드들 및 아키텍쳐 특정 노드들을 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 34.

제 33 항에 있어서,

상기 서브젝트 프로그램 코드는 서브젝트 아키텍쳐에 대해 실행하도록 설계된 서브젝트 코드이고 타깃 아키텍쳐에 대해 실행하기 위한 타깃 코드로 동적으로 번역되고, 상기 번역기 소프트웨어는:

서브젝트 아키텍쳐 및 타깃 아키텍쳐의 특정 조합에 특정인 아키텍쳐 특정 노드들을 포함하도록 상기 중간 표현을 발생하는 단계를 수행하도록 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 35.

제 34 항에 있어서,

상기 번역기 소프트웨어는:

처음에는 상기 서브젝트 코드 내의 상기 명령들 모두를 서브젝트 아키텍쳐 특정 노드들로서 표현하는 단계로서, 각각의 서브젝트 아키텍쳐 특정 노드는 상기 서브젝트 코드 내의 개개의 명령에 대응하는, 상기 표현하는 단계;

상기 서브젝트 코드 내의 명령이 타깃 아키텍쳐 전용 변환 펑크션을 제공하기 위한 것 중 하나인지의 여부를 결정하고, 서브젝트 아키텍쳐 특정 노드들을 타깃 아키텍쳐 전용 변환 펑크션을 제공하도록 결정된 이들 명령들을 위한 타깃 아키텍쳐 특정 노드들로 변환하는 단계; 및

타깃 아키텍쳐 전용 코드 발생 펑크션을 제공하는 것으로서 식별되지 않은 나머지 서브젝트 아키텍쳐 특정 노드들로부터 베이스 노드들을 발생하는 단계를 수행하도록 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 36.

제 35 항에 있어서,

상기 번역기 소프트웨어는 상기 타깃 아키텍쳐에 전용되는 상기 타깃 아키텍쳐 특정 노드들로부터 대응하는 타깃 코드를 발생하기 위해 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 37.

제 34 항에 있어서,

상기 번역기 소프트웨어는 상기 타깃 아키텍쳐에 전용되지 않은 상기 베이스 노드들로부터 대응하는 타깃 코드를 발생하기 위해 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 38.

서브젝트 컴퓨팅 환경에 적합한 서브젝트 프로그램 코드를 번역해서 타깃 컴퓨팅 환경에 적합한 타깃 프로그램 코드를 생성하는, 프로세서 및 상기 프로세서에 연결된 메모리를 가진 타깃 컴퓨팅 환경에 사용하기 위한 번역기 장치에 있어서,

상기 서브젝트 프로그램 코드 내의 명령들을 디코드하도록 구성된 디코딩 메카니즘;

복수의 가능한 타입들의 IR 노드들 중에서 적어도 하나의 타입의 IR 노드들을 포함하도록 상기 디코딩된 프로그램 코드의 중간 표현(IR)을 발생하도록 구성된 중간 표현 발생 메카니즘;

상기 디코딩된 프로그램 코드 내의 각각의 개별 명령에 대해 상기 중간 표현으로 어느 타입의 IR 노드들을 발생할 것인가를 결정하도록 구성된 중간 표현(IR) 타입 결정 메카니즘을 포함하고,

상기 중간 표현(IR)의 상기 IR 노드들은 상기 프로그램 코드에 의해 수행되는 표현식들, 계산들, 및 동작들의 요약 표현들인, 번역기 장치.

청구항 39.

제 38 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 베이스 노드들 및 복합 노드들을 포함하는, 번역기 장치.

청구항 40.

제 39 항에 있어서,

베이스 노드들은 상기 프로그램 코드를 실행하는 임의의 서브젝트 아키텍쳐의 최상의 기본 시맨틱을 표현하므로, 상기 베이스 노드들의 시맨틱은 더 간단한 시맨틱을 표현하는 다른 노드들로 분해될 수 없는, 번역기 장치.

청구항 41.

제 40 항에 있어서,

베이스 노드들은 복수의 가능한 서브젝트 아키텍쳐들에 걸쳐 일반적인, 번역기 장치.

청구항 42.

제 40 항에 있어서,

복합 노드들은 베이스 노드 표현들의 것보다 상기 프로그램 코드의 복합 명령들의 상기 시맨틱의 더 컴팩트한 표현을 제공하는, 번역기 장치.

청구항 43.

제 42 항에 있어서,

상기 복합 노드들은 상수 오퍼랜드 값이 중간 필드에서 중간 타입 명령 자체로 인코딩되는 상기 중간 타입 명령들을 표현하는, 번역기 장치.

청구항 44.

제 42 항에 있어서,

복합 노드는 상기 디코딩된 프로그램 코드로 명령의 동일한 시맨틱을 표현하기 위해 복수의 베이스 노드들로 분해될 수 있는, 번역기 장치.

청구항 45.

제 42 항에 있어서,

상기 프로그램 코드는 서브젝트 아키텍쳐에 의해 실행되도록 설계되고, 상기 중간 표현 발생 메카니즘은 상기 서브젝트 아키텍쳐에 대해 대응적으로 구성 가능한 이들 특징들에 대해서만 복합 노드들을 발생하는 복합 노드 발생 메카니즘을 더 포함하는, 번역기 장치.

청구항 46.

제 39 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 폴리모픽 노드들을 더 포함하는, 번역기 장치.

청구항 47.

제 46 항에 있어서,

상기 프로그램 코드는 서브젝트 아키텍쳐에 대해 실행하도록 설계된 서브젝트 코드이고 타깃 아키텍쳐에 대해 실행하기 위한 타깃 코드로 동적으로 번역되고, 상기 중간 표현 발생 메카니즘은:

폴리모픽 노드들을 포함하도록 상기 중간 표현을 발생하기 위한 폴리모픽 노드 발생 메카니즘을 더 포함하고, 폴리모픽 노드들은 상기 서브젝트 코드 내의 특정 명령에 대해 특정인 상기 타깃 아키텍처의 펑크션에 대한 펑크션 포인터를 포함하는, 번역기 장치.

청구항 48.

제 47 항에 있어서,

상기 폴리모픽 노드 발생 메카니즘은 상기 타깃 아키텍처의 상기 특징들이 특정 서브젝트 명령의 상기 시맨틱로 하여금 베이스 노드들로서 실현되면 상실되게 할 수 있을 경우 폴리모픽 노드들을 발생하는, 번역기 장치.

청구항 49.

제 47 항에 있어서,

각 폴리모픽 노드는 상기 서브젝트 코드 내의 특정 명령 및 상기 타깃 아키텍처의 펑크션의 조합에 특정인, 번역기 장치.

청구항 50.

제 47 항에 있어서,

상기 중간 표현(IR) 타입 결정 메카니즘은 폴리모픽 노드들로서 실현될 폴리모픽 명령들의 리스트 상의 명령에 대응하는 서브젝트 코드 내의 명령을 식별하는 폴리모픽 식별 메카니즘을 더 포함하고;

서브젝트 명령이 상기 폴리모픽 명령들의 리스트 상의 명령들에 대응할 경우, 상기 중간 표현 발생 메카니즘은 상기 폴리모픽 명령들의 리스트 상의 서브젝트 명령들에 대응하는 이들 서브젝트 명령들에 대해서만 폴리모픽 노드들을 발생하는, 번역기 장치.

청구항 51.

제 38 항에 있어서,

상기 복수의 가능한 타입들의 IR 노드들은 베이스 노드들 및 아키텍처 특정 노드들을 더 포함하는, 번역기 장치.

청구항 52.

제 51 항에 있어서,

상기 프로그램 코드는 서브젝트 아키텍처에 대해 실행하도록 설계된 서브젝트 코드이고 타깃 아키텍처에 대해 실행하기 위한 타깃 코드로 동적으로 번역되고, 상기 중간 표현 발생 메카니즘은:

서브젝트 아키텍처 및 타깃 아키텍처의 특정 조합에 특정인 아키텍처 특정 노드들을 포함하도록 상기 중간 표현을 발생하는 아키텍처 특정 노드 발생 메카니즘을 더 포함하는, 번역기 장치.

청구항 53.

제 52 항에 있어서,

상기 중간 표현 발생 메카니즘은:

처음에는 상기 서브젝트 코드 내의 상기 명령들 모두를, 상기 서브젝트 코드 내의 개개의 명령에 대응하는 각각의 서브젝트 아키텍처 특정 노드들로서 표현하고;

상기 서브젝트 코드 내의 명령이 타깃 아키텍처 전용 변환 펑크션을 제공하기 위한 것 중 하나인지의 여부를 결정하고, 서브젝트 아키텍처 특정 노드들을 타깃 아키텍처 전용 변환 펑크션을 제공하도록 결정된 이들 명령들을 위한 타깃 아키텍처 특정 노드들로 변환하고;

타깃 아키텍처 전용 코드 발생 펑크션을 제공하는 것으로서 식별되지 않은 나머지 서브젝트 아키텍처 특정 노드들로부터 베이스 노드들을 발생하도록 구성되어 있는, 번역기 장치.

청구항 54.

제 53 항에 있어서,

상기 타깃 아키텍처에 전용되는 상기 타깃 아키텍처 특정 노드들로부터 대응하는 타깃 코드를 발생하는 전용 타깃 코드 발생 메카니즘을 더 포함하는, 번역기 장치.

청구항 55.

제 52 항에 있어서,

상기 타깃 아키텍처에 전용되지 않은 상기 베이스 노드들로부터 대응하는 타깃 코드를 발생하는 비전용 타깃 코드 발생 메카니즘을 더 포함하는, 번역기 장치.

청구항 56.

제 47 항에 있어서,

상기 발생된 폴리모픽 노드들은 타깃 코드 발생 중 할당될 상기 레지스터들을 특정하는, 번역기 장치.

청구항 57.

제 47 항에 있어서,

상기 발생된 폴리모픽 노드들은 다른 상황에서는 상기 폴리모픽 노드로부터 결정할 수 없는 상기 폴리모픽 노드 내의 상기 펑크션 포인터로부터 정보를 추정함으로써 일반 커널 최적화(generic kernel optimization)에 이용되는, 번역기 장치.

청구항 58.

제 50 항에 있어서,

서브젝트 명령이 폴리모픽 명령들의 리스트상의 명령에 대응할 경우, 상기 중간 표현 발생 메카니즘은 폴리모픽 노드들 또는 폴리모픽 명령들의 상기 리스트 상의 것에 대응하는 이들 서브젝트 명령들을 위한 베이스 노드들을 발생하는, 번역기 장치.

청구항 59.

제 10 항에 있어서,

상기 발생된 폴리모픽 노드들은 타깃 코드 발생 중 할당될 상기 레지스터들을 특정하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 60.

제 10 항에 있어서,

상기 발생된 폴리모픽 노드들은 다른 상황에서는 상기 폴리모픽 노드로부터 결정할 수 없는 상기 폴리모픽 노드 내의 상기 펑크션 포인터로부터 정보를 추정함으로써 일반 커널 최적화에 이용되는, 프로그램 코드의 중간 표현 발생 방법.

청구항 61.

제 13 항에 있어서,

서브젝트 명령이 폴리모픽 명령들의 리스트상의 명령에 대응할 경우, 상기 중간 표현 발생 단계는 폴리모픽 노드들 또는 폴리모픽 명령들의 상기 리스트 상의 것에 대응하는 이들 서브젝트 명령들을 위한 베이스 노드들을 발생하는, 프로그램 코드의 중간 표현 발생 방법.

청구항 62.

제 29 항에 있어서,

상기 발생된 폴리모픽 노드들은 타깃 코드 발생 중 할당될 상기 레지스터들을 특정하는, 컴퓨터 판독 가능 저장 매체.

청구항 63.

제 29 항에 있어서,

상기 발생된 폴리모픽 노드들은 다른 상황에서는 상기 폴리모픽 노드로부터 결정할 수 없는 상기 폴리모픽 노드 내의 상기 펑크션 포인터로부터 정보를 추정함으로써 일반 커널 최적화에 이용되는, 컴퓨터 판독 가능 저장 매체.

청구항 64.

제 32 항에 있어서,

IR 노드들의 상기 타입을 결정하기 위해 컴퓨터에 의해 실행 가능한 상기 컴퓨터 판독 가능 코드는 또한, 서브젝트 명령이 폴리모픽 명령들의 상기 리스트상의 명령에 대응할 경우, 폴리모픽 노드들 또는 폴리모픽 명령들의 상기 리스트 상의 것에 대응하는 이들 서브젝트 명령들을 위한 베이스 노드들을 발생하는, 컴퓨터 판독 가능 저장 매체.

청구항 65.

복수의 가능한 서브젝트/타깃 처리 아키텍쳐 페어링들(pairings) 사이에서 구성 가능한 번역기를 이용하여, 서브젝트 프로세서 아키텍쳐 상에서 실행될 수 있는 서브젝트 프로그램 코드를 타깃 처리 아키텍쳐 상에서 실행될 수 있는 타깃 코드로 번역하는 방법에 있어서,

상기 서브젝트 프로그램 코드가 복수의 가능한 서브젝트 프로세서 아키텍쳐들로부터 실행되도록 설계되는 서브젝트 프로세서 아키텍쳐를 선택하는 단계;

상기 타깃 프로그램 코드가 복수의 가능한 타깃 프로세서 아키텍쳐들로부터 실행되는 타깃 프로세서 아키텍쳐를 선택하는 단계; 및

상기 선택된 서브젝트 프로세서 아키텍쳐와 상기 선택된 타깃 프로세서 아키텍쳐의 페어링을 이용하여 상기 서브젝트 프로그램 코드를 타깃 프로그램 코드로 번역하도록 번역기를 구성하는 단계를 포함하는, 방법.

청구항 66.

제 65 항에 있어서,

상기 타깃 프로그램 코드가 상기 타깃 처리 아키텍쳐 상에서 실행되고 있는 동안 런-타임(run-time)에서 상기 서브젝트 프로그램 코드를 타깃 프로그램 코드로 동적으로 번역하는 단계를 더 포함하는, 방법.

청구항 67.

제 65 항에 있어서,

상기 서브젝트 프로그램 코드 내의 명령들을 디코딩하는 단계;

상기 선택된 서브젝트 프로세서 아키텍쳐와 상기 선택된 타깃 프로세서 아키텍쳐의 상기 페어링에 기초하여 의도된 상기 특정 번역기 구성에 기초한 상기 디코딩된 프로그램 코드 내의 각 개개의 명령에 대한 상기 디코딩된 프로그램 코드의 중간 표현에 복수의 가능한 타입들의 IR 노드들 중 어느 타입의 중간 표현(IR) 노드들이 이용되는 가를 결정하는 단계; 및

복수의 가능한 타입들의 IR 노드들 중에서 적어도 하나의 타입의 IR 노드들을 포함하도록 상기 디코딩된 프로그램 코드의 중간 표현(IR)을 발생하는 단계를 더 포함하고;

상기 중간 표현(IR)에서의 상기 IR 노드들은 상기 프로그램 코드에 의해 수행되는 상기 표현식들, 계산들, 및 동작들의 요약 표현들인, 방법.

청구항 68.

제 67 항에 있어서,

일반 변환 특징들 및 특정 변환 특징들의 조합을 포함하도록 상기 중간 표현(IR)을 발생하는 단계를 더 포함하고, 일반 변환 특징들은 복수의 가능한 프로세서 아키텍쳐들에 대해 구현될 수 있고, 한편 특정 변환 특징들은 특정 프로세서 아키텍쳐에 의해 구현될 수 있는, 방법.

청구항 69.

제 68 항에 있어서,

의도된 상기 특정 번역기 구성은 이용되는 일반 변환 특징들과 특정 변환 특징들의 각각의 조합을 결정하는, 방법.

청구항 70.

복수의 가능한 서브젝트/타깃 처리 아키텍쳐 페어링들 사이에서 구성 가능한 번역기를 이용하여, 서브젝트 프로세서 아키텍쳐 상에서 실행될 수 있는 서브젝트 프로그램 코드를 타깃 처리 아키텍쳐 상에서 실행될 수 있는 타깃 코드로 번역하는 방법을 수행하기 위해 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드의 형태로 그 위에 상주하는 번역기 소프트웨어를 가진 컴퓨터 판독 가능 저장 매체에 있어서,

상기 방법은:

상기 서브젝트 프로그램 코드가 복수의 가능한 서브젝트 프로세서 아키텍쳐들로부터 실행되도록 설계되는 서브젝트 프로세서 아키텍쳐를 선택하는 단계;

상기 타깃 프로그램 코드가 복수의 가능한 타깃 프로세서 아키텍쳐들로부터 실행되는 타깃 프로세서 아키텍쳐를 선택하는 단계; 및

상기 선택된 서브젝트 프로세서 아키텍쳐 및 상기 선택된 타깃 프로세서 아키텍쳐의 페어링을 이용하여 상기 서브젝트 프로그램 코드를 타깃 프로그램 코드로 번역하도록 번역기를 구성하는 단계를 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 71.

제 70 항에 있어서,

상기 번역기 소프트웨어는 상기 타깃 프로그램 코드가 상기 타깃 처리 아키텍쳐 상에서 실행되고 있는 동안 런-타임에서 상기 서브젝트 프로그램 코드를 타깃 프로그램 코드로 동적으로 번역하기 위해 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하는, 컴퓨터 판독 가능 저장 매체.

청구항 72.

제 70 항에 있어서,

상기 번역기 소프트웨어는:

상기 서브젝트 프로그램 코드 내의 명령들을 디코딩하는 단계;

상기 선택된 서브젝트 프로세서 아키텍쳐와 상기 선택된 타깃 프로세서 아키텍쳐의 상기 페어링에 기초하여 의도된 상기 특정 번역기 구성에 기초한 상기 디코딩된 프로그램 코드 내의 각 개개의 명령에 대한 상기 디코딩된 프로그램 코드의 중간 표현에 복수의 가능한 타입들의 IR 노드들 중 어느 타입의 중간 표현(IR) 노드들이 이용되는 가를 결정하는 단계; 및

복수의 가능한 타입들의 IR 노드들 중에서 적어도 하나의 타입의 IR 노드들을 포함하도록 상기 디코딩된 프로그램 코드의 중간 표현(IR)을 발생하는 단계를 수행하도록 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하고,

상기 중간 표현(IR)에서의 상기 IR 노드들은 상기 프로그램 코드에 의해 수행되는 상기 표현식들, 계산들, 및 동작들의 요약 표현들인, 컴퓨터 판독 가능 저장 매체.

청구항 73.

제 72 항에 있어서,

상기 번역기 소프트웨어는 일반 변환 특징들 및 특정 변환 특징들의 조합을 포함하도록 상기 중간 표현(IR)을 발생하기 위해 컴퓨터에 의해 실행 가능한 컴퓨터 판독 가능 코드를 더 포함하고, 일반 변환 특징들은 복수의 가능한 프로세서 아키텍처들에 대해 구현될 수 있고, 한편 특정 변환 특징들은 특정 프로세서 아키텍처에 의해 구현될 수 있는, 컴퓨터 판독 가능 저장 매체.

청구항 74.

제 73 항에 있어서,

의도된 상기 특정 번역기 구성은 이용되는 일반 변환 특징들과 특정 변환 특징들의 각각의 조합을 결정하는, 컴퓨터 판독 가능 저장 매체.

청구항 75.

복수의 가능한 서브젝트/타깃 처리 아키텍처 페어링들 사이에 구성 가능한 번역기를 이용하여, 서브젝트 프로세서 아키텍처 상에서 실행될 수 있는 서브젝트 프로그램 코드를 타깃 컴퓨팅 환경의 타깃 처리 아키텍처 상에서 실행될 수 있는 타깃 코드로 번역하기 위한, 프로세서 및 상기 프로세서에 결합된 메모리를 가진 타깃 컴퓨팅 환경에서 사용하기 위한 번역기 장치에 있어서,

상기 서브젝트 프로그램 코드가 복수의 가능한 서브젝트 프로세서 아키텍처들로부터 실행되도록 설계되는 서브젝트 프로세서 아키텍처를 선택하도록 구성되는 서브젝트 프로세서 선택 메카니즘;

상기 타깃 프로그램 코드가 복수의 가능한 타깃 프로세서 아키텍처들로부터 실행되는 타깃 프로세서 아키텍처를 선택하도록 구성되는 타깃 프로세서 선택 메카니즘; 및

상기 선택된 서브젝트 프로세서 아키텍처 및 상기 선택된 타깃 프로세서 아키텍처의 페어링을 이용하여 상기 서브젝트 프로그램 코드를 타깃 프로그램 코드로 번역하도록 번역기를 구성하도록 구성되는 구성 메카니즘을 포함하는, 번역기 장치.

청구항 76.

제 75 항에 있어서,

상기 타깃 프로그램 코드가 상기 타깃 처리 아키텍처 상에서 실행되고 있는 동안 런-타임에서 상기 서브젝트 프로그램 코드를 타깃 프로그램 코드로 동적으로 번역하도록 구성된 번역 메카니즘을 더 포함하는, 번역기 장치.

청구항 77.

제 75 항에 있어서,

상기 서브젝트 프로그램 코드 내의 명령들을 디코딩하도록 구성되는 디코딩 메카니즘;

상기 선택된 서브젝트 프로세서 아키텍쳐와 상기 선택된 타깃 프로세서 아키텍쳐의 상기 페어링에 기초하여 의도된 상기 특정 번역기 구성에 기초한 상기 디코딩된 프로그램 코드 내의 각 개개의 명령에 대한 상기 디코딩된 프로그램 코드의 중간 표현에 복수의 가능한 타입들의 IR 노드들 중 어느 타입의 중간 표현(IR) 노드들이 이용되는 가를 결정하도록 구성되는 중간 표현(IR) 타입 결정 메카니즘; 및

복수의 가능한 타입들의 IR 노드들 중에서 적어도 하나의 타입의 IR 노드들을 포함하도록 상기 디코딩된 프로그램 코드의 중간 표현(IR)을 발생하도록 구성되는 중간 표현(IR) 발생 메카니즘을 더 포함하고,

상기 중간 표현(IR)에서의 상기 IR 노드들은 상기 프로그램 코드에 의해 수행되는 상기 표현식들, 계산들, 및 동작들의 요약 표현들인, 번역기 장치.

청구항 78.

제 77 항에 있어서,

상기 중간 표현(IR) 발생 메카니즘은 또한 일반 변환 특징들 및 특정 변환 특징들의 조합을 포함하기 위해 상기 중간 표현(IR)을 발생하도록 구성되고, 일반 변환 특징들은 복수의 가능한 프로세서 아키텍쳐들에 대해 구현될 수 있고, 한편 특정 변환 특징들은 특정 프로세서 아키텍쳐에 의해 구현될 수 있는, 번역기 장치.

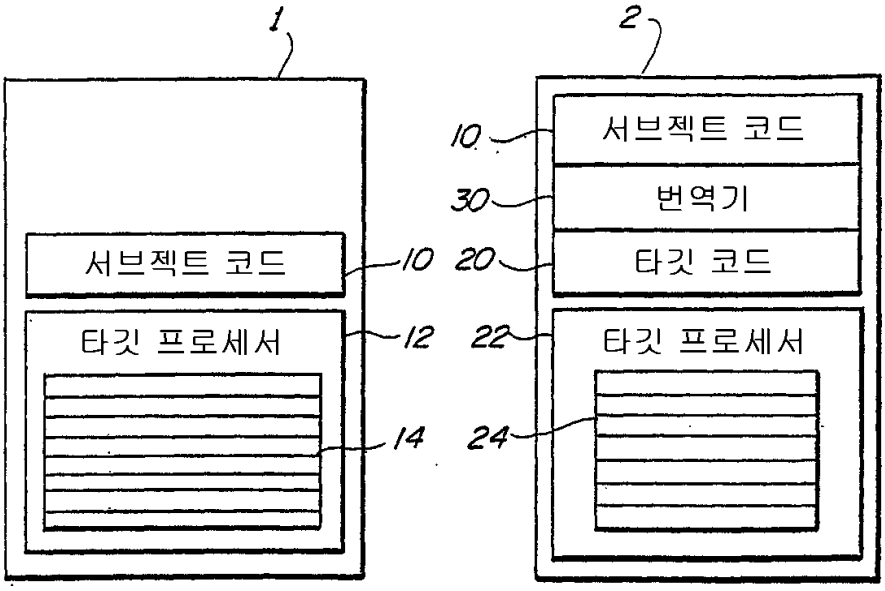
청구항 79.

제 78 항에 있어서,

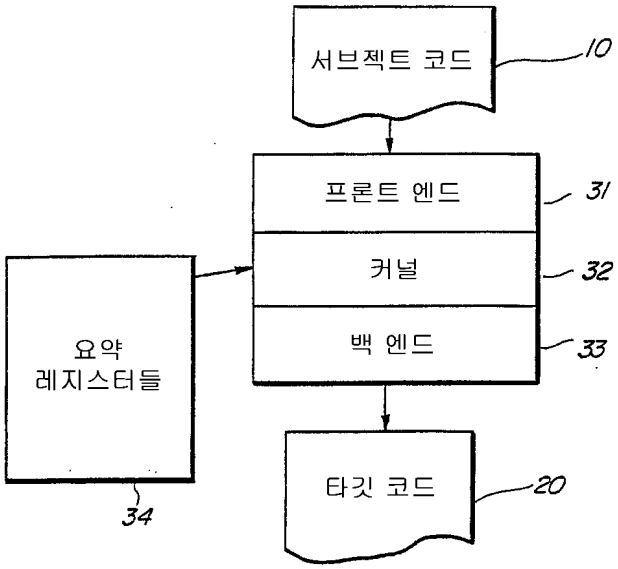
의도된 상기 특정 번역기 구성은 이용되는 일반 변환 특징들과 특정 변환 특징들의 각각의 조합을 결정하는, 번역기 장치.

도면

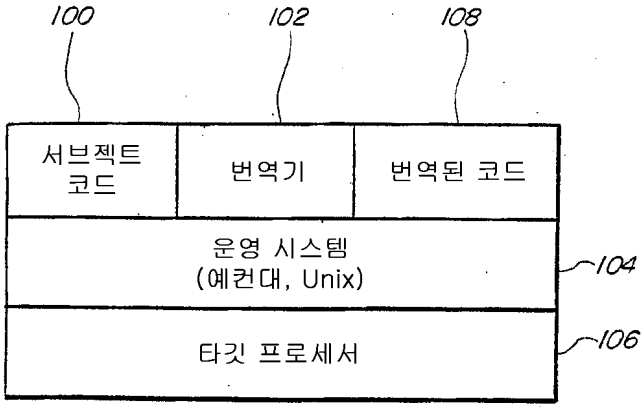
도면1



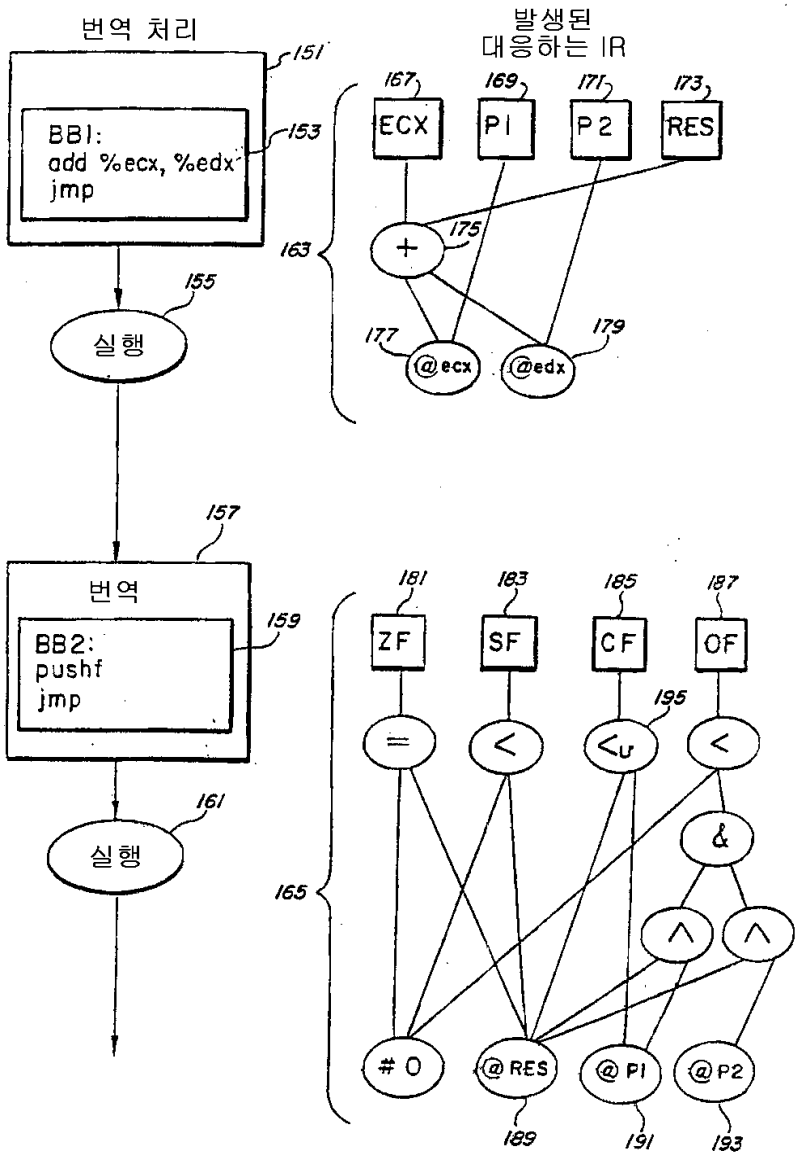
도면2


서브젝트 코드
10
프론트 엔드
31
커널
32
백 엔드
33
요약
레지스터들
34
타깃 코드
20

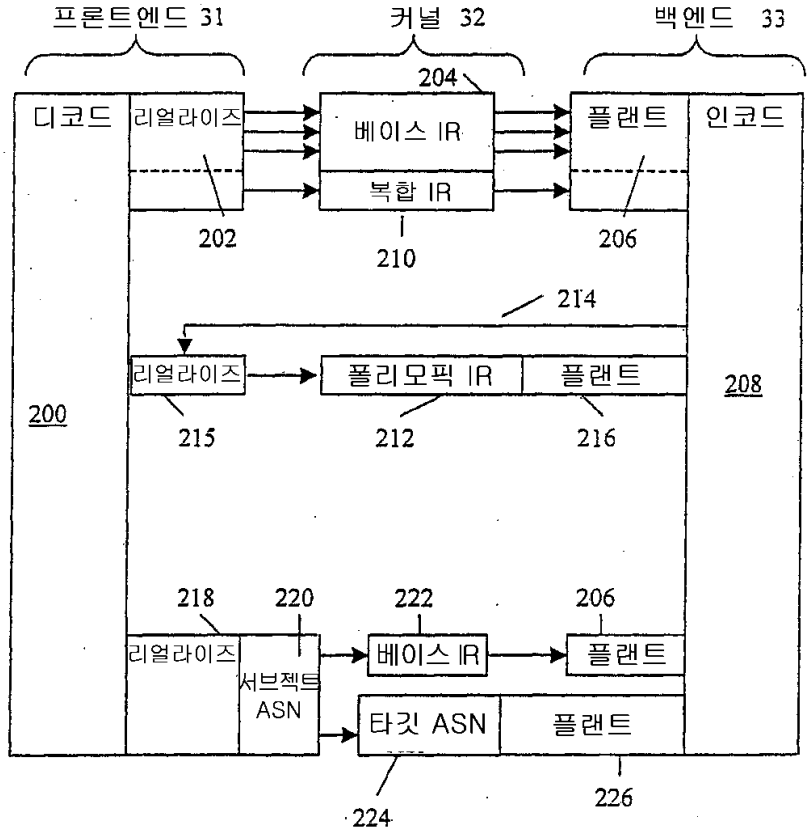
도면3


100
102
108
서브젝트
코드
번역기
번역된 코드
운영 시스템
(예컨대, Unix)
104
타깃 프로세서
106

도면4


번역 처리
151
BB1:
add %ecx, %edx
jmp
153
실행
155
157
번역
BB2:
pushf
jmp
159
실행
161
163
165
발생된
대응하는 IR
167
ECX
169
P1
171
P2
173
RES
175
+
177
@ecx
179
@edx
181
ZF
183
SF
185
CF
187
OF
195
=
<
<u
<
&
^
^
#0
@RES
@P1
@P2
189
191
193

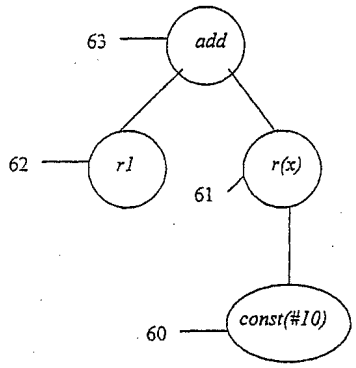
도면5



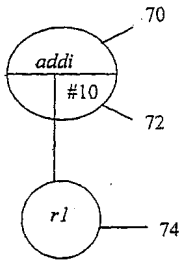
도면6

명령: addi r1, #10

베이스 노드 표현

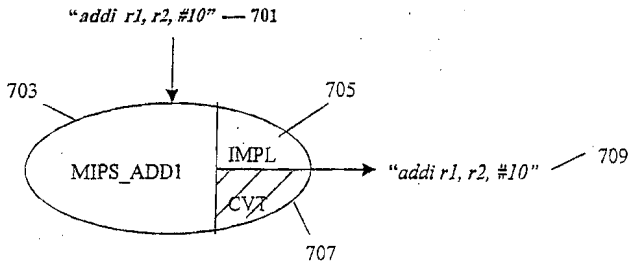


복합 노드 표현



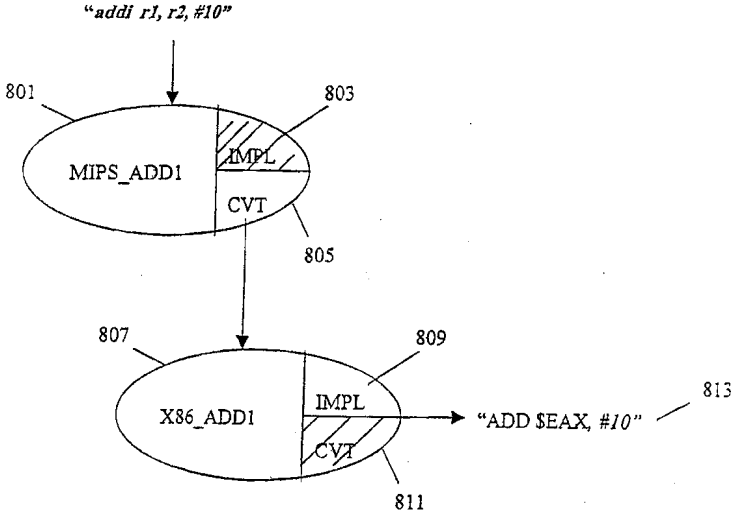
도면7

MIPS-MIPS 가속기

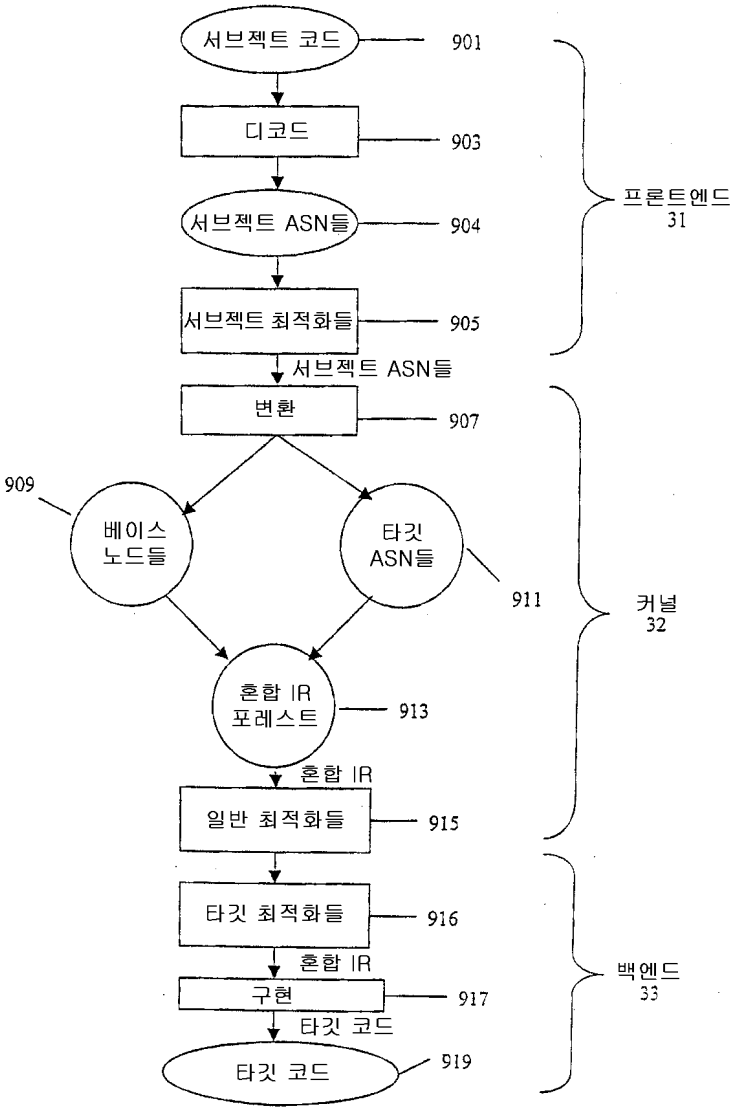


도면8

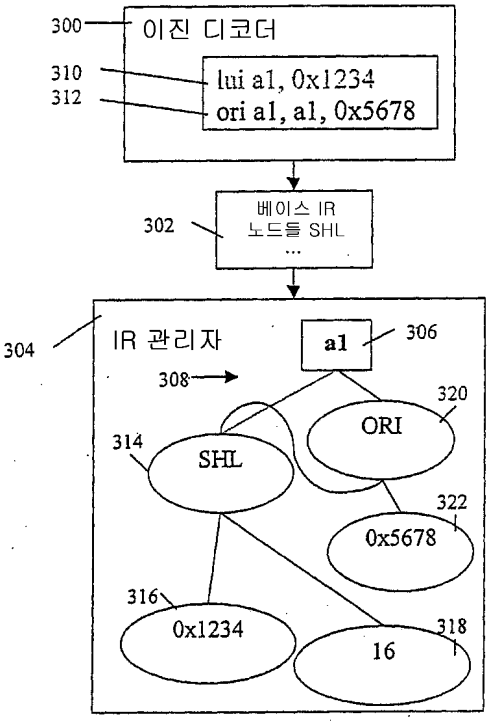
MIPS-x86 번역기



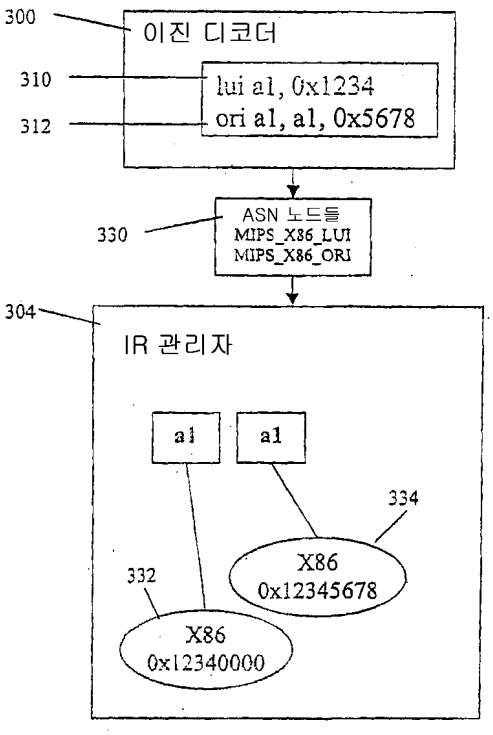
도면9


서브젝트 코드
901
디코드
903
서브젝트 ASN들
904
서브젝트 최적화들
905
프론트엔드
31
서브젝트 ASN들
변환
907
909
베이스
노드들
타깃
ASN들
911
커널
32
혼합 IR
포레스트
913
혼합 IR
일반 최적화들
915
타깃 최적화들
916
혼합 IR
구현
917
백엔드
33
타깃 코드
타깃 코드
919

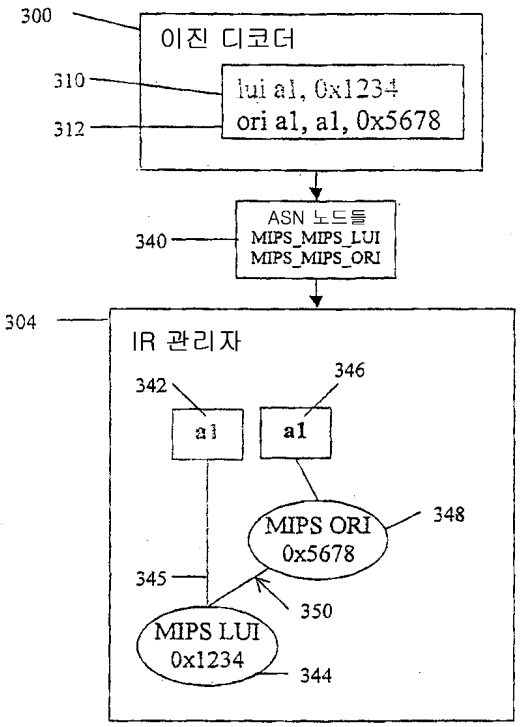
도면10


300
이진 디코더
310
312
lui a1, 0x1234
ori a1, a1, 0x5678
302
베이스 IR
노드들 SHL
...
304
IR 관리자
a1
306
308
320
ORI
314
SHL
322
0x5678
316
0x1234
318
16

도면11


300
이진 디코더
310
312
lui a1, 0x1234
ori a1, a1, 0x5678
330
ASN 노드들
MIPS_X86_LUI
MIPS_X86_ORI
304
IR 관리자
a1
a1
334
X86
0x12345678
332
X86
0x12340000

도면12


300
이진 디코더
310
lui a1, 0x1234
312
ori a1, a1, 0x5678
340
ASN 노드들
MIPS_MIPS_LUI
MIPS_MIPS_ORI
304
IR 관리자
342
346
a1
a1
MIPS ORI
0x5678
348
345
350
MIPS LUI
0x1234
344