조건부 실행 명령어의 비순차 실행을 위한 프로세서 구조

정하영[O], 문제길, 이용석, 정진우[1]
연세대학교 전기전자공학과 프로세서 연구실
삼성종합기술원 Comm. & Network Lab[1]
{hyjeong[O], jgmoon}@dubiki.yonsei.ac.kr, yonglee@yonsei.ac.kr, jjoung@samsung.com[1]

A Processor Architecture for Supporting Out-of-Order Conditional Execution

Hayoung Jeong[O], Jegil Moon, Yongsurk Lee, Jinoo Joung[1]
Processor Lab., Dept. of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea
Comm. & Network Lab., Samsung Advanced Institute of Technology, Kiheung, Korea[1]

요 약

조건부 실행 명령어는 분기명령어의 사용을 줄여 분기 명령어 예측 실패로 인한 프로세서의 성능 저하를 막을 수 있다. 하지만 조건부 실행 명령어는 순차적 프로세서를 위하여 설계되었기 때문에, 고성능 비순차적 프로세서에서는 적용할 수 없었다.

본 논문에서는 기존의 슈퍼스칼라 프로세서 구조를 최소한의 변경을 통하여 조건부 실행 명령어의 비순차 실행을 지원하는 구조를 제안한다. 또한 제안된 구조를 시뮬레이션 할 수 있는 시뮬레이터를 작성 성능을 검증하였다. 그 결과 제안된 구조를 통하여 프로세서의 성능을 27% 이상 향상시킬 수 있다.

1. 서 론

고성능 프로세서에서 분기 명령어는 높은 ILP (Instruction Level Parallelism)를 얻어내는데 가장 큰 장애물이다. 프로세서는 분기 예측을 위한 하드웨어를 가지고 있어야 하며, 매 cycle 분기 예측을 수행하여야 한다. 또한 분기 예측이 잘못 되었을 경우 분기명령어 이후에 실행한 명령어의 결과를 모두 지워서 프로세서의 상태를 복구하여야 한다. 특히 동시에 여러 명령어를 실행하는 superscalar 프로세서나 VLIW(Very Long Instruction Word) 프로세서의 경우 분기 명령어에 의한 성능 감소 현상이 매우 두드러지게 나타난다.

분기 명령어의 단점을 보완하기 위해서 조건부 실행 명령어를 도입하는 프로세서가 나타났다. 조건부 실행이란 명령어가 특정한 조건에서만 수행되는 것을 말한다. 조건부 실행 명령어는 특정한 필드에 명령어가 수행되는 조건을 기록하여, 해당 명령어가 수행될 때의 프로세서의 상태와 기록된 조건을 비교하여 자신의 수행여부를 판단한다[1].

조건부 실행을 지원하는 프로세서에는 Alpha 21264, IA-64, ARM 프로세서 등이 있다[2][3][4]. Alpha에서는 조건부 실행 명령어로 CMOV(Conditional MOVe) 명령어를 사용한다. CMOV 명령어는 1개의 조건 오퍼랜드와 소스 오퍼랜드, 결과 오퍼랜드를 사용한다. 이때 조건 오퍼랜드가 0일 때 만 소스 오퍼랜드를 결과 오퍼랜드로 이동시키는 명령을 수행한다[2]. IA-64의 경우 대부분의 명령어가 조건부 실행을 지원하며 이를 위하여 predicate register를 사용한다. 비교 명령어로 predicate register를 설정한 후, 명령어에 오퍼랜드로 predicate register를 사용함으로 명령어를 조건부 수행을 할 수 있다[3][5]. 위의 두 프로세서가 고성능 superscalar 프로세서인 반면, ARM은 임베디드 프로세서로 많이 쓰이는 프로세서이다. ARM의 경우 모든 명령어가 조건 코드를 가지고 있으며, 수행 시 조건을 만족할 경우에만 실행 결과를 레지스터에 저장한다[4].

조건부 실행 명령어는 분기명령어의 수를 줄이는 효과가 있고, 분기 명령어 예측의 실패로 인한 성능 저하를 막을 수 있다[6]. 분기 명령어의 수를 줄인다는 것은 기본 블록 사이즈를 크게 만들 수 있다는 것을 의미하므로 superscalar 프로세서나 VLIW 프로세서의 경우 많은 명령어를 동시에 이슈 할 수 있다는 것을 의미한다. 하지만 조건부 실행 명령어는 이전 명령어의 실행 결과로 생성된 프로세서의 상태를 통해서 자신의 실행 여부를 판단하므로 비순차 실행이 어려운 단점이 있다.

이러한 조건부 실행 명령어는 if-else 구문을 위하여 많이 사용된다. 조건부 수행 명령어를 지원하지 않는 프로세서에서는 하나의 if-else 구문 위하여 2개 이상의 분기 명령어가 필요하지만, 조건부 수행 명령어를 지원하는 프로세서에서는 하나의 조건 생산 명령어와 하나 이상의 조건부 실행 명령어에 의하여 수행하는 것이다. 따라서 분기 명령어의 사용 빈도를 줄일 수 있게 된다. If-else 구문을 분석하여 보면 다음 3가지로 구분할 수 있다. 첫 번째는 조건 생산자이다. 조건 생산자는 비교 연산을 통하여 프로세서의 상태 레지스터를 설정하며, 설정된 상태 레지스터 값은 이후의 조건부 실행 명령어의 수행 조건이 된다. 두 번째는 여러 개의 조건부 실행 명령어로 구성된 조건문이다. 각 조건부 실행 명령어는 프로세서의 상태 레지스터에 설정된 값이 자신의 실행 조건을 만족하는지 판단하고, 그 결과에 따라 실행을 하거나 아무 작업도 하지 않는다. 마지막 세 번째는 이러한 조건부 실행 명령어의 실행 결과를 사용하는 명령어이다.

조건부 실행 명령어의 비순차 실행을 위해서는 수행 조건의 판단을 연기하는 방법이 있다. 이 방법은 조건

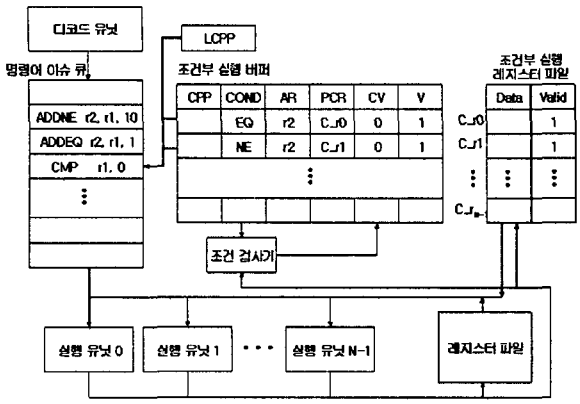

그림 1 조건부 실행 명령어의 비순차 실행을 위한 프로세서 구조

생산자와 조건부 실행 명령어를 병렬적으로 수행하고 조건 생산자에 의하여 생성된 프로세서 상태에 알맞은 조건부 실행 명령어의 결과만을 레지스터에 업데이트 하는 방식이다. 기존의 연구에서는 조건부 실행 명령어의 조건 판단 연기를 위하여 특수한 마이크로 오퍼레이션을 사용하거나 결과를 선택하는 명령어를 삽입하였다[6]. 그러나 마이크로 오퍼레이션을 사용하는 경우 지연 cycle이 늘어나고, 특수한 명령어를 사용하는 경우 프로그램을 다시 작성해야 되는 단점이 있다. 따라서 본 논문에서는 하드웨어를 통하여 조건부 실행 명령어의 비순차 실행을 지원하는 프로세서 구조를 제안한다.

2. 조건부 실행 버퍼

조건부 실행 명령어의 비순차 실행을 지원하기 위해서 그림 ?과 같이 최근 조건 생산자 포인터(LCPP: Latest Condition Producer Pointer), 조건부 실행 버퍼(CEB: Conditional Execution Buffer), 조건부 실행 레지스터 파일(CRF: Conditional Register File), 조건 검사기(CV: Condition Validator)가 추가로 필요하다. LCPP는 최근의 조건 생산자를 추적하는 포인터로 가장 최근의 조건 생성자의 명령어 큐를 지정하고 있다. 조건부 실행 레지스터 파일은 조건부 실행 명령어의 결과를 저장하는 레지스터 파일이다. 각 조건부 실행 엔트리에 할당된 조건부 실행 명령어의 결과 값이 저장되므로 조건부 실행 엔트리 수와 조건부 실행 레지스터의 수는 같다고 할 수 있다. 조건 검사기는 연산 결과로 나오는 상태 값과 조건부 실행 버퍼에 저장된 실행 조건을 비교하여 이 명령어가 실행 가능한 것인지 검사하는 기능을 한다.

CPP	COND	AR	PCR	CV	V

그림 2 조건부 실행 버퍼의 엔트리 구조

조건부 실행 버퍼는 그림 ?와 같은 구조의 엔트리를 가지고 있으며, 조건부 수행 명령어가 이슈될 때 마다 할당된다. 엔트리의 구조를 살펴보면, 조건 생산자를 지정하고 있는 포인터인 CPP(Condition Producer Pointer)가 있다. 이것은 조건부 실행 명령어의 실행 조건을 생성하는 조건 생성자의 명령어 큐를 지정하고 있다. COND(CONDition)는 조건부 실행 명령어의 실행 조건을 저장 하고 있다. OR은 조건부 실행 명령어의 리네이밍 되기 이전 결과 아키텍쳐럴 레지스터 번호이다. 이 AR(Architectural Register)은 해당 조건부 실행 명령어 이후에 이슈 되는 명령어간의 dependency를 검출하기 위하여 사용된다. PCR(Physical Conditional Register)은 조건부 실행 명령어를 위해 할당된 리네이밍된 레지스터 번호를 의미한다. 이 레지스터 번호는 조건부 실행 레지스터 파일이라는 별도의 레지스터 파일의 하나의 레지스터를 지정하는데 사용된다. CV는 조건 생산자가 실행을 완료하여 조건을 생성하면 그것을 조건과 비교를 하여 조건이 맞는지 틀리는지를 저장한다. 마지막으로 V는 이 엔트리가 유효한 것인가를 의미한다.

조건부 실행 버퍼의 동작은 다음과 같다. 조건 생산자가 디코드 되어 이슈 큐에 들어가면 LCPP는 조건 생산자의 이슈 큐를 지정하게 된다. 이 후 조건부 실행 명령어가 디코드 되면 조건 실행 버퍼는 하나의 엔트리를 할당한다. 이때 LCPP를 참조하여 CPP를 설정하고, 조건부 실행을 위한 리네이밍 레지스터를 하나 할당하여 조건부 실행 버퍼에 AR 필드와 PCR 필드에 리네이밍 정보를 기록한다.

조건 생산자가 실행을 마치면 조건부 수행 버퍼의 엔트리 중 CPP 필드가 자신을 지정하는 엔트리가 있는지 검색하게 된다. 만약 자신의 조건을 사용하는 엔트리가 있다면 생성된 조건을 포워딩하게 된다. 이때 조건 검사기는 조건부 실행 버퍼의 COND 필드에 저장된 조건과 비교 연산하여 CV 필드를 셋 하거나 리셋 한다. 만약 실행조건이 맞지 않다면 해당 엔트리를 무효화 처리한다. 매 cycle 조건부 수행 버퍼는 CV 필드와 V 필드가 모두 세팅된 엔트리가 있는지 검사하며, CV 필드와 V 필드가 모두 세팅된 엔트리는 아키텍쳐럴 레지스터 파일에 수행 결과를 업데이트하고 해당 엔트리를 삭제한다.

일반 명령어 중 소스 오퍼랜드로 조건부 실행 명령어의 결과를 사용하는 명령어를 검출하기 위하여 매 cycle 조건부 실행 버퍼의 AR 필드를 검색한다. 이 때 조건부 실행 명령어의 결과를 사용하는 명령어가 디코드 된다면, 이 명령어는 조건 생산자가 완료되어 조건부 실행 명령어 중 유효한 명령어만을 선택할 때 까지 이슈 할 수 없다. 하지만 조건 생산자가 먼저 실행을 완료한다면 아무런 지연 없이 명령어를 실행할 수 있다.

3. 시뮬레이션

조건부 실행 버퍼의 성능을 예측하기 위해서 ARM ISA를 사용하는 시뮬레이터를 작성하였다. ARM ISA는 모든 명령어에 조건 코드가 있어서 조건부 실행이 매우 광범위하게 사용되고 있다[4].

Fetch	Decode	Issue	Exec	Writeback	Commit

그림 3 제안된 프로세서의 파이프라인 구조

본 논문에서 사용한 시뮬레이터는 그림 ?과 같은 파이프라인 구조를 갖는다. Fetch 단에서는 명령어를 캐시에서 읽어와 명령어 큐로 보낸다. Decode 단에서는 명령어를 디코드 하여 명령어 이슈 큐로 보내고, Issue 단에서는 명령어의 의존성을 검사하여 실행 가능한 상태

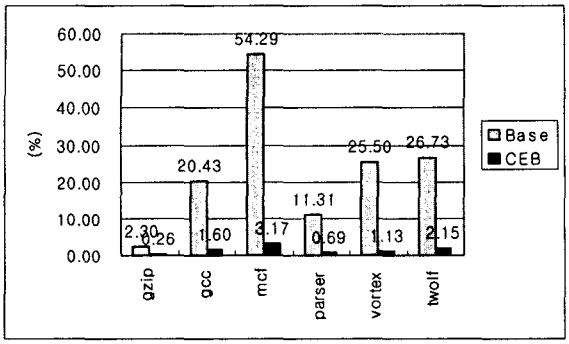

그림 4 지연 Cycle 비교

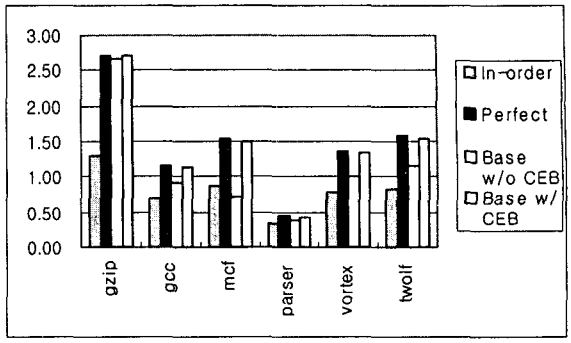

그림 5 IPC 비교

가 되면 각 실행 유닛을 할당해 준다. Exec 단에서는 각 실행 유닛 별로 명령어를 실행하고, writeback 단에서 그 결과를 리오더 버퍼에 저장한다. Commit 단에서는 명령어의 원래 순서대로 명령어가 완료할 수 있도록, 이전 명령어가 모두 수행 완료될 때 까지 기다린 후, 이전 명령어들이 모두 완료된다면 명령어를 명령어 이슈 큐에서 삭제하여 수행을 완료한다.

기준 프로세서 모델은 한 cycle에 4개의 명령어씩 실행이 가능한 슈퍼스칼라 프로세서이다. 실행 유닛으로는 4개의 ALU, 2개의 MUL/DIV, 1개의 LSU을 사용하였다.

비교를 위하여 명령어를 순차적으로 실행하는 In-order 모델, 비순차적 조건부 수행 명령어를 디코드 단에서 완벽하게 예측하여 실행하는 Perfect 모델, 조건 생성자의 수행이 완료될 때까지 조건부 명령어를 이슈하지 않는 Base 모델, 조건부 실행 버퍼를 사용하여 조건부 실행 명령어의 비순차적 실행을 지원하는 CEB 모델을 비교하였다. Workload로는 SPEC 2000의 integer 벤치마크 프로그램 중 gzip, gcc, mcf, parser, vortex, twolf를 사용하였다.

4. 시뮬레이션 결과

Base 모델과 CEB 모델의 조건부 실행 명령어 처리를 위해 지연되는 cycle을 비교를 해보면 그림 4와 같다. Base 모델에 비하여 조건부 실행 버퍼를 사용한 CEB 모델의 지연 cycle 최대 51.1%, 평균 21.9% 차이가 남을 볼 수 있다. 이것은 조건 생산자와 조건부 실행 명령어는 연속하여 오는 경우가 많으므로, Base 모델의 경우 조건 생산자가 실행을 완료할 때 까지 조건부 실행 명령어를 이슈하지 않기 때문에 많은 지연 cycle이 생기게 된다.

그림 5는 IPC를 비교한 것이다. 명령어의 순차적 실행을 하는 In-order 모델의 성능이 가장 낮음을 볼 수 있다. Base 모델의 경우 조건부 실행 명령어를 제외한 명령어에 대하여 비순차적 처리를 하므로 In-order 모델보다 IPC가 평균 42.5% 높아진다. Perfect 모델의 경우 Base 모델에 비하여 평균 28.9% 높은 IPC를 보여준다. 본 논문에서 제안한 CEB 모델은 Base 모델에 비하여 평균 27.1% 높은 IPC를 보여주며 Perfect 모델에 근접한 성능을 보여주고 있다. 이것으로 보아 제안된 프로세서 구조는 조건부 실행 명령어의 비순차 실행을 완벽하게 지원하며, 그로인해 평균 27% 이상의 성능이 향상되었다.

5. 결론

조건부 실행 명령어는 분기명령어의 수를 줄이는 효과가 있고, 분기 명령어 예측의 실패로 인한 성능 저하를 막을 수 있다. 하지만 조건부 실행 명령어는 비순차적으로 수행할 수 없기 때문에 고성능 슈퍼스칼라 프로세서에서 적용하기 어려웠다.

본 논문에서는 조건부 실행 명령어의 비순차 실행을 지원하는 프로세서 구조를 제안하였으며, SPEC2000 integer 벤치마크를 사용하여 시뮬레이션 한 결과 평균 27%의 성능향상을 보였다.

6. 참고문헌

[1] J. R. Allen, K. Kennedy, C. Porterfield, and J. Warren, "Conversion of control dependence to data dependence", Proceedings of the 10th ACM Symposium on Principles of Programming Languages, January 1983, pp. 177~189W.

[2] R. E. Kessler, "The Alpha 21264 Microprocessor", IEEE Micro, pp.24~36, March 1999

[3] IA-64 Application Developer's Architecture Guide, Intel Corporation, May 1999, pp2-4

[4] ARM Architecture Reference Manual, ARM Limited, June 2000, pp. A3-5~A3-8

[5] D. I. August, D. A. Connors, S. A. Mahlke, J. W. Sias, K. M. Crozier Ben-Chung Cheng, P. R. Eaton, Q. B. Olaniran Wen-mei W. Hwu, "Intergrated Predicated and Speculative Execution in the IMPACT EPIC Architecture", Proceedings of the 25th annual international symposium on Computer architecture, June 1998, pp. 227~237

[6] S. A. Mahlke, R. E. Hank, R. A. Bringmann, J. C. Gyllenhaal, D. M. Gallagher, and W. W. Hwu, "Characterizing the Impact of Predicated Execution on Branch Prediction", Proceedings of the 27th International Symposium on Microarchitecture, December 1994, pp. 217-227