컴퓨터," 패터슨과 디첼의 작품

더글러스 W. 클라크 및 윌리엄 D. 스트렉커 VAX 시스템
아키텍처
디지털 장비 공사 1925 앤도버 스트리트
툭스버리, MA 81876

1980년 9월

패터슨과 디첼의 논문[3]은 축소 명령어 집합 컴퓨터(RISC)가 복합 명령어 집합 컴퓨터(CISC)만큼 비용 효율적일 수 있다고 주장합니다. 이 글에서는 이들의 주장 중 몇 가지가 오해의 소지가 있음을 지적하고, 그 반대편에 대한 몇 가지 증거를 제시하고자 합니다. 저희는 그들과 마찬가지로 VAX-11 아키텍처에 크게 의존하고 있습니다(5] 예시).

해당 CISC에 비해 RISC의 우월성을 입증하기는 매우 어렵습니다. 비용과 성능에 대한 일반적인 평가는 CISC와 RISC의 차이가 극단적으로 크지 않는 한 충분하지 않을 것입니다. 종이 디자인만으로는 충분하지 않습니다. RISC와 CISC를 면밀히 비교하려면 하드웨어와 마이크로코드의 완전한 설계, 프로세서의 구성 또는 시뮬레이션, 컴파일러 및 운영 체제의 작성, 다양한 애플리케이션에 대한 성능 측정이 필요할 것으로 보입니다. 이러한 수준의 노력 없이는 RISC의 비용 효율성 향상에 대한 주장을 뒷받침하기 어렵습니다.

그러나 이러한 포괄적인 실험이 수행되기 전까지는 이 주제에 대해 논의하거나 논쟁할 가치가 없다는 말은 아닙니다. 패터슨-디첼 논문에서 제기된 문제는 흥미롭고 중요하며, 어떤 스타일의 컴퓨터든 설계자는 이에 대한 논의를 통해 이득을 얻을 수 있습니다.

복잡성 대 크기

이 논문은 "감소"와 "복합"을 대조합니다. 이는 잘못된 이분법입니다. 명령어 집합의 복잡성을 단순히 명령어 수로만 측정할 수 있을까요? 명령어 수를

데이터 유형의 수? 완전성(예: 연산자와 데이터 유형의 직교성)이 복잡성을 증가시키거나 감소시키나요? 이 논문에 대한 가장 심각한 비판은 RISC 또는 CISC에 대한 공식적인 정의가 포함되어 있지 않다는 것입니다. 복잡성에 대한 정의가 없는 상황에서 "...우리는 많은 경우 복잡한 명령어 집합이 유용하기보다는 해롭다고 주장할 것이다"라는 진술은 의미가 없습니다.

코드 밀도

패터슨과 디첼은 일반적으로 VAX와 같은 CISC의 장점으로 여겨지는 코드 압축(11)은 RISC에서도 쉽게 달성할 수 있으며, 저렴한 메모리 덕분에 고밀도 코드가 예전만큼 중요하지 않다고 주장합니다. 그러나 RISC에 대한 그들의 주장은 "코드 압축은 원래의 [간단한] 명령어 집합을 정리함으로써 쉽게 달성할 수 있다고 생각합니다."라는 문장으로 요약되는데, 이는 증거가 뒷받침되지 않으면 설득력이 없습니다. 메모리 비용은 시간이 지남에 따라 감소하지만, 소량의 저렴한 메모리가 많은 양의 메모리보다 비용이 적게 드는 것은 사실입니다. 또한, 특정 컴퓨터 모델에 대한 메모리는 시스템당 비용인 반면, 마이크로코드 개발(CISC의 경우)은 일회성 비용입니다.

물론 고밀도 코드는 다른 이점도 제공합니다. 캐시 블록당, 페이지당 더 많은 인스트럭션이 있으면 캐시 성능과 페이징 성능이 향상됩니다.

언어마다 다른 지침 사용

이 논문은 "프로그램 실행의 대부분을 차지하는 연산 코드는 극소수"라는 점을 (Shustek [4] 등을 인용하여) 지적합니다. 충분히 맞는 말입니다. 하지만 다른 언어는 어떨까요?

예를 들어 포트란의 경우 상위 20개 명령어는 COBOL 히트 퍼레이드에 전혀 포함되지 않을 수 있습니다. 예를 들어, 잘 알려진 포트란 벤치마크인 휘트스톤은 VAX-11/780에서 일반적인 종류의 명령어 빈도 분포를 보여줍니다. 상위 10개 명령어가 전체 명령어 실행의 50%를 차지하고, 상위 20개 명령어가 75% 이상, 상위 40개 명령어가 90% 이상을 차지합니다. 그러나 유사한 합성 COBOL 벤치마크에서는 상위 10개 명령어가 전체 명령어 실행의 8%에 불과합니다. 휘트스톤의 상위 20개는 COBOL 벤치마크 실행의 75%가 아닌 21%를 차지합니다. 더 놀라운 사실은 Nhetstone의 상위 2B가 COBOL 벤치마크에서 차지하는 시간이 4%에 불과하다는 점입니다.

따라서 다양한 지침은 다양한 언어를 지원하는 데 도움이 될 수 있습니다. 최근 논문 [2]에서 지적한 바와 같이,

> "대부분의 프로그래밍 환경에서는 시스템이 여러 언어를 효과적으로 지원할 수 있어야 합니다. . . 특정 언어에 맞춰진 단일 인스트럭션 세트는 다른 언어의 구현을 어렵고 비효율적으로 만들 수 있으므로 제한적입니다."

시간이 생명입니다

슈스텍은 논문 [4]에서 명령어의 실행 빈도보다 실행에 소요되는 시간이 성능에 더 중요하다고 주장합니다. 그는 실행 시간이 많이 소요되는 드물게 실행되는 명령어의 예를 제시합니다. 이러한 명령어를 다중 명령어 시퀀스로 대체하면 이러한 문제가 훨씬 더 악화될 수 있으며, 자주 실행되는 명령어만 최적화하는 것은 명백한 위험이 있습니다.

다음은 VAX-11/780의 예입니다. 한 시간 공유 벤치마크에서 MOVC3 명령어(문자 이동 명령어)는 명령어 실행의 0.4% 미만을 차지하지만 13%의 시간을 차지하며, 빈도 순위에서는 60위, 시간 순위에서는 1위를 차지합니다.

컴파일러 작성의 용이성

연산자와 데이터 유형을 직교로 유지하려는 욕구로 인해 대규모 명령어 집합이 정당화될 수 있습니다. 따라서 VAX 아키텍처에는 예를 들어 바이트, 단어, 장어에 대한 2연산자 및 3연산자 버전 등 6개의 서로 다른 배타적 OR 명령어가 포함되어 있습니다. 이 중 일부는 의심할 여지 없이 거의 사용되지 않습니다. 하지만 VAX 컴파일러에서는 이 모든 인스트럭션이 있으므로 코드 생성이 간소화됩니다(이는 VAX 컴파일러 작성자가 입증한 바 있습니다). 또한 일부에 대한 마이크로코드가 있으면 다른 마이크로코드는 매우 저렴하게 구현할 수 있습니다.

Microcode Size

이 논문의 PDP-11/48과 VAX-11/780의 마이크로코드 크기 비교는 추측에 불과합니다. 11/50의 마이크로코드 양은 명령어 세트가 동일함에도 불구하고 11/40의 거의 10배에 달합니다. 마이크로코드의 증가는 (가) 성능 향상, (나) 하드웨어를 마이크로

코드로 대체, (다) 보다 정교한 진단 및 콘솔 기능을 반영합니다. 또한 11/780은 세 가지 명령어 세트(PDP-11, VAX-11, EDIT)를 지원하며 메모리 관리가 훨씬 더 복잡해졌습니다.

디자인 시간 단축

이 백서의 PDP-1 - VAX-11/780 설계 시간 비교도 추측에 불과합니다. 인스트럭션 세트를 완전히 무시하고 VAX-11/780 하드웨어 시스템과 VMS 소프트웨어는 PDP-1 프로세서보다 훨씬 더 복잡합니다. 또한 대량 생산용 제품을 설계하는 대기업에는 소량 생산용 제품을 설계하는 소규모 회사에는 존재하지 않는 수많은 시간이 소요되는 프로세스가 있습니다.

통합 데이터 저장소

다른 모든 것이 동일하다면 소량보다 대량의 마이크로코드에서 더 많은 마이크로코드 오류가 발생한다는 것은 논란의 여지가 없습니다. 하지만 소량(RISC)의 경우 누군가가 복잡한 기능을 어딘가에서 구현해야 합니다. RISC에서는 컴파일러와 런타임 시스템이 이전에는 마이크로코드가 부담하던 부담을 떠안게 되고, 이 소프트웨어에서 구현 오류가 발생할 가능성이 높습니다.

이제 누군가는 소프트웨어가 마이크로코드보다 작성하기 쉽고 변경하기 쉽다고 주장할 수 있습니다. 그리고 마이크로코드 컴파일러와 같은 마이크로프로그래밍 개발 도구가 이러한 상황을 바꿀 것이라고 반박할 수도 있습니다. 하지만 사용자 프로그램에서 요구하는 복잡한 기능을 어떻게든 구현해야 하고 오류가 발생할 수 있습니다.

설계 오류의 탐지는 공식 및 비공식 테스트 프로세스에 크게 좌우됩니다. 이는 프로세서와 마찬가지로 컴파일러에서도 마찬가지입니다. 이러한 프로세스는 프로세서용으로 쉽게 개발할 수 있습니다. (패터슨과 디첼은 VAX-11/780 마이크로코드의 복잡성을 최적화 컴파일러의 복잡성과 어떻게 비교했을까요?)

인터페이스 수준

하위 레벨 인터페이스(RISC)에 비해 상위 레벨 하드웨어-소프트웨어 인터페이스(CISC)의 장점 중 하나는 특수 하드웨어를 사용하여 비용 대비 성능을 향상시킬 수 있는 기회가 더 많다는 점입니다. 예를 들어 곱하기 명령어가 있는 CISC와 명령어가 없는 RISC를 생각해 보겠습니다. 곱하기 함수는 이동, 분기, 이동, 더하기의 시퀀스를 통해 RISC에서 수행됩니다. RISC에서 곱하기 기능의 속도를 높이려면 전체 프로세서의 속도를 높여야 하는 반면, CISC에서 곱하기 명령의 속도를 높이려면 특수 데이터 경로와 제어를 추가하면 됩니다. 일부 기술 및 성능 수준에서는 후자가 전자보다

훨씬 저렴할 수 있습니다.

지표가 없는 상황에서 패터슨-디첼 논문의 이 부분은 설득력이 떨어집니다. 물론 간단한 명령어 세트는 복잡한 명령어 세트보다 더 적은 실리콘으로 구현할 수 있습니다. 그러나 이것이 명령어 세트의 복잡성을 줄임으로써 시스템 비용 효율성이 증가한다는 것을 의미하지는 않습니다.

VAX 인덱스 지침

비합리적인 구현에 대한 일화적인 설명은 확실히 흥미롭습니다. 하지만 복합 명령어가 동등한 단순 명령어 시퀀스보다 더 느리게 실행되는 것이 일반적일까요? 이 논문에서는 여러 개의 간단한 명령어 시퀀스가 780에서 45%의 속도 향상으로 VAX INDEX 명령어를 대체할 수 있다고 보고합니다. 이것은 아키텍처가 아닌 구현의 문제입니다. 근본적으로 결국 , 두 경우 모두 동일한 하드웨어를 가정 할 때 둘 이상의 명령어로 INDEX 함수를 구현하는 것은 단일 명령어 버전보다 시간이 덜 걸릴 수 없습니다. 이 이상 현상에 대한 설명은 788의 부동 소수점 가속기가 다중 명령어 구현에서 곱하기의 속도를 높이지만 INDEX를 전혀 보지 못하기 때문입니다.

최종 VAX 사실

패터슨과 디첼은 마케팅 전략이 명령어 집합의 크기나 복잡성을 증가시킬 수 있다고 제안합니다. 직접 경험한 바에 따르면 VAX 아키텍처에서는 그렇지 않다고 말할 수 있습니다.

참조

[1] Dietz, W.B. 및 Szewerenko, L. 컴퓨터 제품군 아키텍처 선택, 4단계 최종 보고서: 후보 컴퓨터 아키텍처의 비교 평가. Tech. 보고서, 컴퓨터 과학부 , 카네기 멜론 대학교, 1979년 11월.

{2] Ditzel, D,R. 및 Patterson, D.A. 고수준 언어 컴퓨터 아키텍처에 대한 회고. 제 7회 컴퓨터 아키텍처 국제 심포지엄, 프랑스 라 바울, 5월 198B.

(3] 패터슨, D.A., 및 디첼, D.R. 명령어 감소 컴퓨터의 사례. ACM SIGARCH 컴퓨

터 아키텍처 뉴스, 이번 호.

[4] Shustek, L.J. 컴퓨터 명령어 세트의 분석 및 성능." 박사 학위 논문, 스탠포드 선형 가속기 보고서 205, 스탠포드 대학교, 1978년 5월.

{5) Strecker, W.D. VAX-11/780: DEC PDP-11 제품군에 대한 가상주소 확장 . Proc. NCC, 6월 1978, 967-980쪽.