멀티미디어 명령어를 강화한 수퍼스칼라 RISC 마이크로프로세서 구조

이용환(李龍煥)*, °문병인(文秉仁)**, 이용석(李容錫)**
*현대반도체, **연세대학교 전자공학과
전화 : (02) 361-2872 / 팩스 : (02) 312-4584

Superscalar RISC Microprocessor Architecture with enhanced Multimedia Instructions

Yong Hwan Lee*, °Byung In Moon**, and Yong Surk Lee**
*Hyundae Microelectonics Corp., **Yonsei University
E-mail : jaeger@hmec.co.kr, blue@yonsei.ac.kr, yslee@yonsei.ac.kr

Abstract

For applications in multimedia to which genuine RISC microprocessors are not suitably applicable, a new generation of fast and flexible microprocessors is required. In this paper, as a technique of integrating DSP functionality in a general RISC processor, a RISC that can execute DSP extension instructions is developed to improve the performance of multimedia application execution. This processor can execute DSP instructions in parallel with the execution of ALU instructions for efficient and fast execution. In addition, the execution ability of integer instructions is improved by enhancing the RISC core itself.

I. 서론

최근 멀티미디어 시대가 도래함에 따라 마이크로프로세서의 멀티미디어 처리능력이 중요시되고 있다. 음성 인식 및 합성, 영상 처리, 비디오 응용, 오디오 신호 처리 등 멀티미디어에 관련된 계산은 기존의 범용 마이크로프로세서 명령어들로는 그 처리에 한계를 보이기 때문에 기존에는 DSP(Digital Signal Processing) 전용 칩을 사용하여 왔다. 그러나 최근에는 범용 마이크로프로세서의 성능이 크게 개선됨에 따라 소프트웨어로 이러한 연산을 수행하고자 하는 움직임과 맞물려 멀티미디어 전용 명령어들을 범용 마이크로프로세서에 추가할 수 있게 됨으로써 그래픽, 비디오, 오디오 및 네트웍 등의 응용을 빠르게 처리하는 것이 가능하게 되었다.

본 논문에서는 RISC 마이크로프로세서에 멀티미디어 DSP 명령어를 결합시킨 32 비트 멀티미디어 RISC(YRD; *Yonsei* RISC DSP) 구조를 연구한다. YRD 구조는 한번에 최대 2 개의 명령어를 실행할 수 있는 수퍼스칼라를 기본으로 한다. 여기에 내장 명령어 캐시 및 데이터 캐시를 적용하여 메모리 참조때문에 발생하는 병목현상을 없애고, 수퍼스칼라의 파이프라인을 개선하여 더 많은 명령어가 처리될 수 있도록 하며, 분기 예측의 개념을 도입하여 분기에 의해 낭비되는 명령어 슬롯을 가능한 한 없애게 된다. 또한 수퍼스칼라 실행을 위한 컴파일러의 개선 방안도 제시한다. 이 YRD 구조는 범용 컴퓨터뿐만 아니라 내장형 시스템*(embedded system)*에도 적용될 수 있도록 단순화된 구조를 유지하며 적절한 성능, 높은 생산성, 낮은 생산비용 및 저전력 소비를 이룰 수 있도록 한다.

II. 프로세서의 구조

YRD 는 수퍼스칼라로서 최소의 하드웨어 구조를 가지면서도 최대의 성능을 갖게 하려는 기본 방향에 의해 그 사양이 결정되었다. 이에 따라 우선 YRD 는 비순차적 이슈*(out-of-order issue)*[1]를 지원하지 않도록 하였다. 그러나 메모리 엑세스 명령어인 로드와 스토어 및 DSP 명령어들과 같이 일반 정수 명령어들보다 수행 사이클이 긴 명령어들에 대한 비순차적 종료를 지원한다[2]. 이들의 비순차적 종료를 지원하지 않으면

이들 명령어가 이슈될 때마다 뒤의 명령어들이 바로 이슈되지 못하고 몇 사이클을 기다려야 하기때문에 유용한 사이클이 많이 낭비될 수밖에 없다. 그림 1 에서는 YRD 구조의 블록 다이아그램을 보여준다.

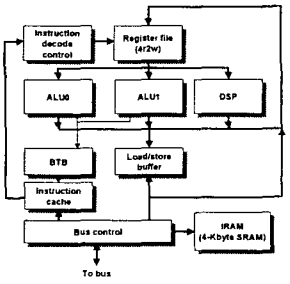

그림 1. 마이크로프로세서 블록다이아그램
Fig. 1. Block diagram of microprocessor

YRD 는 6 단계의 파이프라인(Predict, Fetch, Decode, Execute, Memory, Writeback)을 갖는다. YRD 는 한 사이클에 최대 2 개의 명령어를 디코드 단계에서 이슈하므로 F 단계에서 가능하면 2 개의 명령어를 페치할 수 있도록 해야 한다. 그러나 명령어의 크기가 16, 32, 48 비트로 가변적이므로 2 개의 명령어 크기를 합하면 32 비트에서 96 비트까지이다. YRD 의 명령어 캐시는 모든 경우에 대해 2 개의 명령어를 공급할 수 있도록 하기 위하여 그림 2.와 같이 96 비트의 페치 대역폭을 갖는다. 가변 길이의 명령어를 사용하기 때문에 YRD 의 명령어 공급에는 이것이 매우 유리하게 작용한다. YRD 는 메모리로부터 명령어 캐시에 연결된 시스템 버스가 32 비트이다. 그러나 대부분의 명령어 길이가 16 비트이므로 한번의 버스 동작으로 2 개의 명령어가 유입될 수 있어 시스템 버스를 확장할 필요가 없다[3].

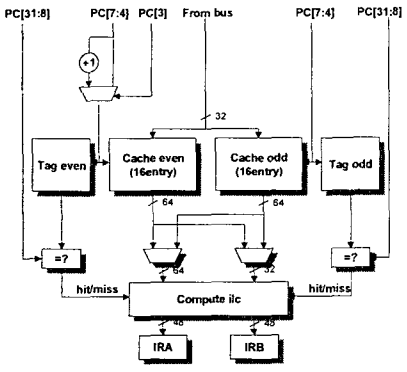

그림 2. 명령어 캐시의 구조
Fig. 2. Instruction cache

그림 3.과 같이 분기 예측을 하는 단계는 P 단계이며 여기서 현재 명령어의 주소를 사용하여 BTB 로 그 다음 명령어를 예측한다. 먼저 F 단계에서는 현재의 명령어를 페치하는데 처음에는 두 개의 명령어 가운데 첫번째 명령어의 시작 주소인 FAPC 만을 알고 있다. FAPC 는 명령어 캐시를 엑세스하기 위해 사용되며 이에 따라 명령어 캐시에서 명령어 라인이 페치된다. 페치된 명령어 라인은 명령어 길이 계산 모듈에서 분석되며 이 결과로 첫번째 명령어의 길이(FAILC), 두번째 명령어의 시작 주소(FBPC)와 그 길이(FBILC), 그리고 두 개의 명령어(FAIR, FBIR)가 나오게 된다. FAPC 와 FBPC 는 P 단계에서 BTB 를 엑세스하는데 사용된다. BTB 의 엑세스 결과 FAPC 또는 FBPC 가 분기 명령어이고 이 분기가 채택될 것이라 예측되면 해당되는 분기 타겟 주소가 다음 페치 주소(next FAPC)가 되며 그렇지 않을 경우는 순차적 주소가 다음 페치 주소가 된다.

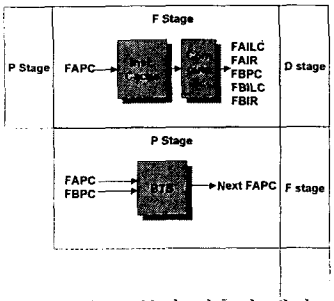

그림 3. 분기 예측의 개념
Fig. 3. Branch prediction concept

그러나 이러한 기본적 개념을 하드웨어로 구현한다면 큰 시간 지연을 필요로 하게 될 수밖에 없다. 이는 F 단계가 거의 끝날 때쯤 해서 발생하는 두번째 명령어의 시작주소 FBPC 를 입력으로 받아야 P 단계가 동작할 수 있으므로 P 단계와 F 단계가 병렬적으로 동시에 수행되지 못하기 때문이다. 또한 이러한 개념을 위해서는 FAPC 와 FBPC 가 동시에 BTB 를 엑세스할 수 있도록 BTB 를 구성해야 하므로 BTB 는 2 개의 포트를 가지고 있어야 하는데 메모리 구조를 가지고 있는 BTB 를 2-포트로 구현하려면 하드웨어적 부담이 패크다.

그림 4.는 첫번째 명령어 주소와 두번째 명령어 주소의 관계를 보여준다. 명령어의 길이는 2, 4, 6 바이트 중 하나이기 때문에 만약 첫번째 명령어의 주소가 n 이라면 이어지는 두번째 명령어의 시작 주소는 n+2, n+4, n+6 중에 하나가 된다. 이러한 특성을 이용하여

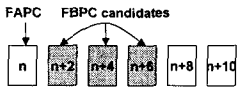

그림 4. 두 명령어의 시작 주소
Fig. 4. Relationship between two instruction addresses

FAPC 와 정확한 FBPC 를 가지고 BTB 를 엑세스하는 대신에 FAPC 와 FBPC 가 될 가능성이 있는 FAPC+2, FAPC+4, FAPC+6 를 가지고 BTB 엑세스를 하게 되는데 이 방법은 당장 FBPC 를 알지 못해도 BTB 엑세스를 할 수 있는 장점을 갖는다. 이러한 방법에 의해 BTB 는 4 개의 주소에 대한 분기 예측을 수행하게 되며 FAPC 에 의해 분기가 채택될 것으로 예측된다면 이 결과를 사용하면 되고 그렇지 않을 경우에는 FAPC+2, FAPC+4, FAPC+6 에 의한 분기 예측 결과 중에서 나중에 F 단계에서 결정되는 FBPC 에 따라 선택하여 사용하면 된다. 이에 따라 P 단계와 F 단계는 거의 병렬적으로 동시에 수행될 수 있으며 분기 예측에 의한 시간 지연은 최소화될 수 있다. 한편 이러한 방법을 사용하면 BTB 에서는 한번에 4 개의 엔트리를 읽어야 되는 부담이 있다. 그러나 읽는 주소가 연속적이므로 BTB 를 4-way interleaved 방식을 사용하여 구현함으로써 이러한 문제를 간단히 해결할 수 있다.

DSP 응용의 특징은 주로 데이터의 크기가 6 비트에서 13 비트 사이의 범위에 있다는 것이다. 예를 들어 의학용으로는 12 비트의 데이터가 사용되며 비디오 응용에서는 주로 8 비트가 사용된다. 따라서 16 비트 연산은 대부분의 응용에서 충분한 데이터 비트로 사용될 수 있기 때문에 흔히 DSP 연산에 관련된 수 표시는 16 비트의 고정 소수점(fixed-point)으로 나타내어 진다. DSP 연산은 정밀도를 크게 요하지 않는 데이터들에 대해 부동 소수점 연산을 하지 않고 데이터 타입을 정수(integer) 타입으로 변환하여 정수 연산을 수행하는 것이다. 이렇게 함으로써 복잡하고 시간이 오래 걸리는 부동 소수점 계산을 피하고 빠르고 간단한 계산을 할 수 있게 된다. YRD 구조의 DSP 유닛은 16x16 곱셈기와 48 비트의 덧셈기를 기본으로 하는 2 개의 MAC(multiply-and-add)으로 이루어져 있다.

한 사이클에 두 개의 명령어를 이슈할 수 있는 YRD 는 기능 유닛(functional units)이 다른 명령어를 처리하고 있어 명령어가 이슈되어도 이를 받아들일 수 없다면 제대로 이슈를 할 수 없게 된다. 이에 따라 명령어들이 가장 많이 사용하는 ALU 를 두 개 두었으며 이 ALU 에서 분기 명령어의 처리도 함께 할 수 있도록 하였다. 이로써 YRD 의 기능 유닛은 2 개의 ALU/branch 유닛, 1 개의 메모리 유닛, 그리고 1 개의 DSP 유닛이 존재한다. YRD 는 이러한 기능 유닛에 따라서 2 개의 ALU 명령어, 2 개의 분기 명령어, 1 개의 메모리 명령어, 1 개의 DSP 명령어 중 최대 2 개의 명령어를 한 사이클에 이슈할 수 있게 된다. YRD 는 2 개의 정수 파이프라인을 가지고 있으며 데이터 포워딩을 제공함으로써 명령어가 의존관계때문에 실행되지 못하는 것을 최대한 막아준다.

III. 시뮬레이터의 작성

마이크로프로세서의 설계에 앞서 우선 구조를 정하고 이의 성능을 예측하는 과정은 필수적이라 할 수 있다. 이러한 과정을 위해 마이크로프로세서의 시뮬레이터가 제작되며 이 시뮬레이터에 의해 각 설계 항목에 대한 trade-off 의 계량화로 최적화된 설계 구조를 도출할 수 있게 된다. 이로써 경험이나 추측에 의한 설계를 하지 않게 되고 정확한 설계 데이터를 얻을 수 있다. 또한 설계 이전이라도 상용화된 프로그램들을 빠른 속도로 시뮬레이션하게 됨으로써 다른 프로세서와의 성능 비교(benchmarking)도 가능하게 된다.

본 논문에서는 프로그램-구동 시뮬레이터를 C 언어를 사용하여 작성한다. 시뮬레이터를 하드웨어 기술언어(hardware description language; HDL)로 기술할 수도 있지만 이럴 경우는 시뮬레이터의 속도가 매우 느리고 하드웨어를 완전히 묘사해야 하기 때문에 부담이 따른다. 그림 5.와 같이 YRD 시뮬레이터는 크게 파이프라인의 흐름과는 역순으로 기술된 시뮬레이터 본체(simulator body)와 시뮬레이터 본체의 실행을 돕고 결과를 분석하기 위한 시뮬레이터 보조(simulator assistant) 루틴으로 나눌 수 있다.

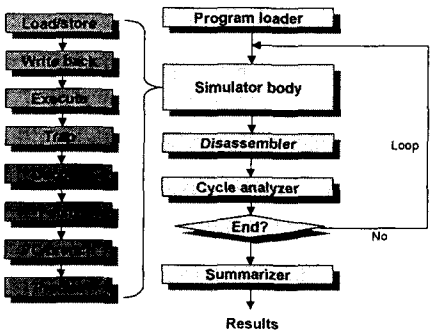

그림 5. 시뮬레이터의 구성
Fig. 5. Flow of simulator

프로그램 로더는 실행할 프로그램을 메모리 영역에 로드해주는 기능을 하며 프로그램 시작 시 단 한번 실

행된다. 디스어셈블러는 시뮬레이터 개발과정에서 시뮬레이터 자체의 디버깅(debugging)을 편하게 해주는 기능과 성능 평가를 위한 기초자료를 제공하는 역할을 한다. 사이클 분석기는 현재 사이클에 어떤 명령어가 진행되고 있는지 또는 그렇지 않을 경우 어떤 원인으로 명령어가 진행되고 있지 못한지를 분석한다. 마지막으로 요약기는 프로그램 실행 후 성능에 관한 결과를 요약해 주므로 이를 사용하여 구조상의 단점 및 개선의 여지를 발견할 수 있게 된다.

IV. 성능평가

그림 6.은 YRD 구조를 시뮬레이션 한 결과이다. 실행 프로그램은 DCT, matrix 연산, IIR 필터, quick sort 이며 YRD-2 는 2 단계 파이프라인 구조를 갖는 프로세서를 뜻하고 이외에 S 는 수퍼스칼라, I 는 2KByte 의 명령어 캐시, D 는 2KByte 의 데이터 캐시, B 는 64 엔트리의 BTB 가 있다는 것을 뜻한다. 그러나 이 구조는 하드웨어적으로 최적화되었는데도 불구하고 CPI 는 1.1 에서 1.2 사이로 별로 낮지 않은 값이다. 2-way 수퍼스칼라 마이크로프로세서가 한번에 2 개의 명령어를 처리할 수 있으므로 가장 이상적인 경우에 CPI 가 0.5 가 될 수 있다는 것에 비하면 이러한 수치는 현저하게 높은 값이라 볼 수 있다. 분석 결과 이와 같은 결과는 소프트웨어적인 것으로 컴파일러로 해결하게 되었다. 앞에서 YRD 구조 시뮬레이션에서 사용된 컴파일러는 수퍼스칼라 구조를 위한 것이 아니다. 따라서 YRD 구조를 위해 사용된 프로그램들은 이러한 수퍼스칼라 구조의 이점을 제대로 살리지 못하여 다소 높은 CPI 가 측정될 수 밖에 없었다고 생각할 수 있다. 이에 따라 기존에 사용되던 프로그램을 수퍼스칼라 YRD 의 구조에 맞도록 수정함으로써 성능을 최적화할 수 있도록 한다.

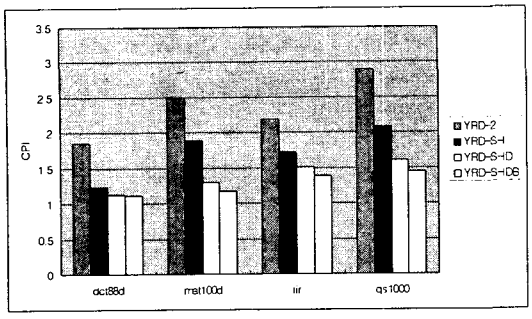

그림 6. 시뮬레이션 결과
Fig. 6. Simulation results

그림 7.은 응용 프로그램을 직접 수정하여 시뮬레이션 한 결과를 보여주고 있다. 프로그램 siv1024a 는 로드/스토어 명령어를 최적화하였고, siv1024b 는 여기에 다중 사이클 명령어를 최적화하였으며 siv1024c 는 분기 명령어와 이중워드 명령어를 최적화하였다. 이 결과 CPI 는 0.58 로 다른 프로그램의 수행결과도 0.6 에서 0.8 사이의 CPI 를 보여준다. 이러한 수치는 2-way 수퍼스칼라 마이크로프로세서의 이상적인 값 0.5 에 매우 근접한 값이라 할 수 있다.

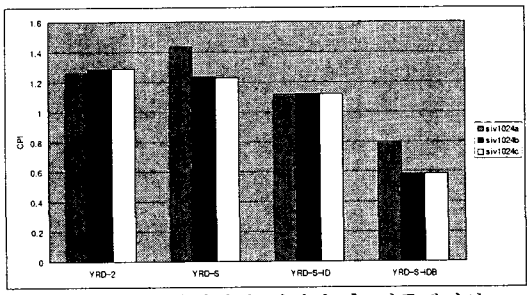

그림 7. 컴파일러의 최적화 후 시뮬레이션
Fig. 7. Simulation after optimizing complier

V. 결론

본 논문에서는 멀티미디어 처리 기능을 강화한 고성능 마이크로프로세서 구조를 연구하고 이 결과로서 최적화된 YRD 구조를 제안하였다. 멀티미디어 연산 처리 능력은 DSP 유닛을 RISC 마이크로프로세서에 추가하고 DSP 명령어를 일반 정수 명령어와 병렬적으로 수행할 수 있도록 함으로써 높이게 되었다. 또한 일반 정수 명령어를 처리할 수 있는 RISC 코어 자체의 성능 향상에 큰 비중을 두어 수퍼스칼라 구조를 적용하였다. 이 결과 YRD 구조는 0.6 에서 0.8 정도의 낮은 CPI 를 보이는 고성능 프로세서임이 입증되었다.

참고문헌

[1] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal*, Vol. 11, Jan. 1967, pp. 25-33

[2] G. F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor," *IBM Journal of Research and Development*, Vol. 34, No. 1, Jan. 1990, pp. 37-58

[3] Kenneth M. Wilson, Kunle Olukotun, Mendel Rosenblum, "Increasing Cache Port Efficiency for Dynamic Superscalar Microprocessors," *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, 1996, pp.147-157