ARM 7 프로세서를 위한 성능평가도구의 개발

심성훈*, 이재범*, 장성태**, 전주식*
* 서울대학교 컴퓨터공학과
** 수원대학교 전자계산학과

Development of Performance Evaluation Tool for ARM 7 Processor

Sung-Hoon Shim*, Jae Bum Lee*, Seong Tae Jhang**, Chu Shik Jhon*
* Dept of Computer Engineering, Seoul National University
** Dept. of Computer Science, Suwon University

요 약

마이크로 프로세서의 개발과정에서 성능평가의 중요성이 점차 증가하고 있다. 이것은 사용자에게 보다 설득력 있는 데이터를 제공하는 의미를 가진다는 것과 함께 개발된 마이크로 프로세서의 성능향상요인을 분석해내고 또한 시스템의 설계에 프로세서의 특성을 보다 효율적으로 반영한다는 점에서 중요하다고 하겠다. 본 논문에서는 ARM 7 프로세서의 성능을 측정하기 위한 도구(시뮬레이터)의 개발에 관해 설명한다.

1. 서 론

오늘날 향상된 성능의 컴퓨터 시스템 개발에 있어서, 사용되는 마이크로 프로세서의 성능이 시스템 전체의 성능에 중요한 영향을 미치고 있다. 또한 마이크로프로세서는 컴퓨터 시스템뿐만이 아니라, 휴대폰, 네트웍 장비, 셋톱 박스등, 여러 시스템의 핵심 부품으로 사용되고 있다 마이크로프로세서 개발에 있어서 개발 기간의 단축과 개발된 마이크로프로세서에 대한 신뢰성등을 높이기 위하여 개발과정에서의 성능평가는 필수적으로 요구된다

시스템의 성능평가방법으로는 해석적 모델링, 모의실험, 벤치마킹과 같은 방법이 보편적으로 사용되고 있는데, 본 논문에서는 모의 실험방법을 이용하는 성능평가도구의 개발에 대해 설명했다, 또한, 정보통신사회에서 점점 중요성이 증대되고 있는 hand-held기기에서 사용되는 마이크로프로세서인 ARM 7을 목표 프로세서로 하여 모의실험을 위한 도구개발에 대해 설명했다

논문의 구성은 다음과 같다 2장에서는 성능평가를 위한 기존의 연구 방향들에 대해서, 3장에서는 실제 본 논문에서 설명하는 성능평가도구의 구현에 대해서, 마지막으로 4장에서는 연구 결과와 함후 과제를 제시한다

2. ARM 7 프로세서

본 논문에서 구현하고자하는 ARM 7 프로세서의 특징 및 명령어들은 다음과 같다

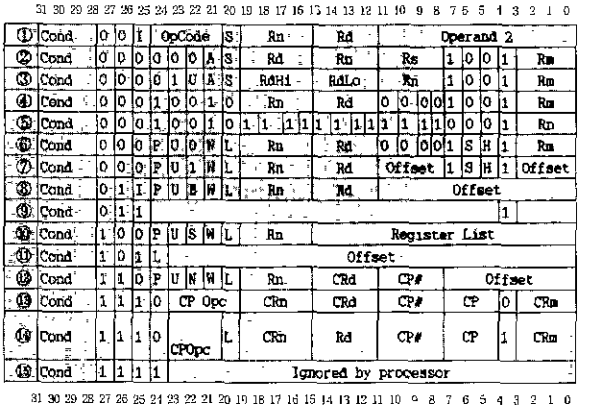
2.1. 프로세서 특징

ARM 7은 32비트의 프로세서로 좋은 성능과 저전력소비, 낮은 가격등의 장점을 가진 범용 RISC 프로세서이다. 범용이기는 하지만 주로 hand-held기기(휴대폰 단말기등)의 핵심 프로세서로 널리 사용되고 있다. 또, RISC 프로세서이기 때문에 해독(decoding)이 상대적으로 간단하며, 이로 인하여 높은 명령어 처리량을 가지며, 실시간 인터럽트응답에 있어서도 좋은 성능을 지니고 있다 또한 전형적인 3단계 파이프라인(명령어 인기, 해독, 실행)을 가지며, 메모리와의 인터페이스도 높은 비용의 메모리시스템을 요구하지 않도록 설계되었다. 이런 이유로 세계 각국의 회사에서 다양한 용도로 사용되고 있다.

ARM 7프로세서는 32비트의 명령어들을 가지며, 상태(시스템 또는 사용자, 슈퍼바이져, FIQ, IRQ, Abort, 비정의)에 따라 각각 17-18개의 레지스터가 있으며, 14번 레지스터는 서브루틴 링크를 위한 복귀주소가 저장되는 레지스터이며, 15번은 현재 PC값을, 16번은 현재 프로세서의 상태를 나타내는 상태레지스터로 설정되어 사용된다.[3]

2.2. 명령어

32비트 명령어인 ARM 명령어의 내부 구조와 형식은 다음 표와 같다[3]



< 표 1. ARM 명령어들의 내부 구조와 형식 >

① 데이터 프로세싱 명령어 및 PSR 전송 명령어
② ③ 곱셈 명령어

④ 단일 데이터 교환 명령어
⑤ 분기와 교환 명령어
⑥ ⑦ 하프워드 데이터(레지스터 옵셋)관련 전송 명령어
⑧ 단일 데이터 전송 명령어
⑨ 비정의 명령어
⑩ 블록데이터 전송 명령어(Pop,Push)
⑪ 분기 명령어
⑫ ⑬ ⑭코프로세서 데이터 관련 처리 명령어
⑮ 소프트웨어 인터럽트 명령어

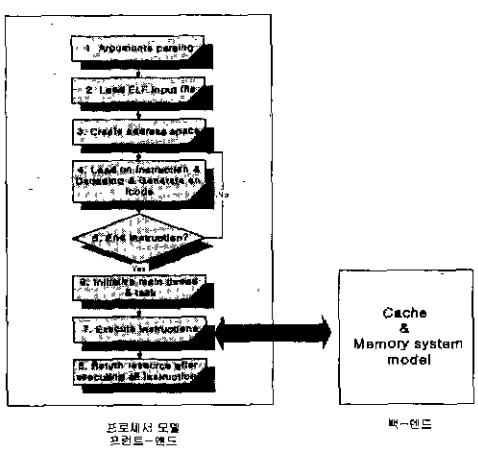
위 표 1에서 P, U, A, B, W, L, S 비트의 의미는 각각 옵셋값의 계산 순서를 나타내는 비트, 기준 주소에 대한 옵셋값의 처리방법(덧셈 또는 뺄셈), 곱셈 명령어에서의 누적(accumulate)여부, 처리 단위(바이트 또는 워드), write-back여부, 읽기(load)인지 쓰기(store)인지, 콘디션 비트의 세팅여부를 나타낸다.

ARM 7 프로세서의 명령어중 특이한 명령어로 블록 데이터 전송 명령어 형태가 있는데, 이 명령어 형태에는 읽기에 관계되는 Ldm 명령어와 쓰기에 관계된 Stm 명령어의 두가지 명령어가 있으며, 이것은 결국에 Pop과 Push와 같은 역할을 한다 즉 각각 하나의 명령어로 여러번의 읽기와 쓰기가 가능토록 명령어 집합을 설계했다.

3. 성능평가 도구(ARMint)의 구조

본 논문에서 설명하는 성능평가도구는 MINT]를 기반으로 하여 구현된 도구로 ARMint(ARM interpreter)라고 부르기로 하겠다. ARMint는 스케쥴링에 관계된 부분은 MINT의 것을 유지하고, 프로세서에 관계된 부분을 ARM 용으로 재작, 구현하였다 [1]

ARMint의 전체적인 흐름도는 다음과 같다.



< 그림 1. ARMint의 전체 구조와 흐름도 >

위의 그림에서 프론트-엔드부분이 구현된 ARMint이며, 백-엔드부분은 실제로 이 성능평가도구를 이용하는 사용자가 구현하는 부분이 된다

ARMint는 입력파일에 대한 처리(읽기와 해독, 모의실험도구를 위한 메모리맵 생성 및 입력구조체 생성)부와 명령문 실행부의 두 부분으로 나뉜다. 위 그림 1에서 1, 2, 3, 4, 5의 부분이 입력파일에 대한 처리부이며, 6, 7, 8부분이 명령문 실행부이다

3.1. 입력파일 처리부

입력파일 처리부내의 각 모듈과 그에 대한 설명은 아래와 같다.

(1) 입력파일 읽기

2.3에서 설명한바와 같이 모의실현도구에서는 다양한 프로그램을 입력으로 받게 된다 ARMint에서는 ELF]라는 특정한 형태의 파일을 입력으로 받는데, ELF(Executable and Linking Format)는 원래 USL (UNIX SYSTEM LABORATORIES)에서 Application Binary Interface(ABI)의 한 부분으로 개발되고 공개된 목적 파일(object file)의 2진 형태(format)로, 기존의 a.out 형태와는 달리 다양한 운영체제에 걸쳐서 사용될 수 있는 이진 인터페이스를 프로그래머에게 제공함으로써, 소프트웨어 개발에 연계성을 주기 위해 만들어졌다.[4]

(2) 주소 공간의 생성

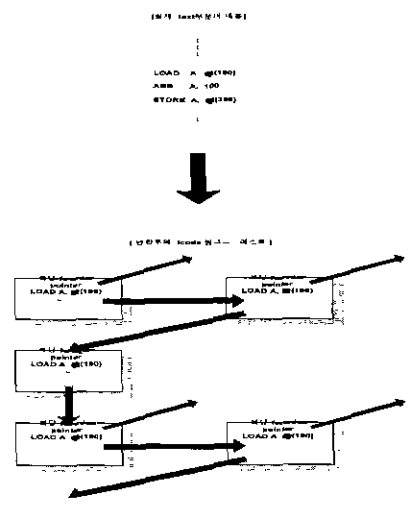
ARMint 자체가 이용하는 메모리 주소 공간의 생성을 의미하며 이는 스택 및 힙, 입력프로그램의 데이터들을 위한 공간등의 생성을 말한다

(3) 명령문 해독

위의 (1)번과정에서 읽은 프로그램의 텍스트부분(결국 명령문(instruction)들을 의미한다.)을 해독(decoding)한다. ARM 7 프로세서의 명령어 형태는 아래와 같다

(4) icode의 생성

명령어 해석과 동시에 ARMint에서 명령어실행에 사용하는 입력구조체를 생성하는 역할을 한다 ARMint에서는 이 입력 구조체를 icode라고 하며, 이 icode는 모의실험을 하기위한 실제 프로그램의 text부분으로, text자체를 메모리에 적재시키는 대신 이 구조체에 저장하여 명령문 실행시 이용한다.

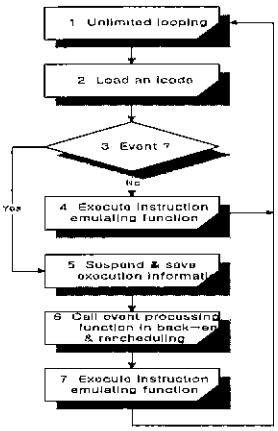


< 그림 2 text의 icode 변환 예 >

3.2. 명령어 실행부

ARMint의 명령어 실행부는 각 명령어에 해당하는 에뮬레이션 함수들의 실행에 관계된 모듈들과 시스템호출 수행에 관계된 모듈들로 나뉜다 흐름도는 아래 그림 3과 같다.

프로그램 구동형 모의실험도구인 ARMint는 입력프로그램의 ARM 프로세서의 명령어를 실제로 소프트웨어적으로 에뮬레이션하여 실행



< 그림 3 명령어 실행부 흐름도 >

한다 따라서 ARM 프로세서의 소프트웨어적 모델링이 필요한데 이는 thread라는 구조체로 구현되며, 구조는 아래 표 2와 같다

멤 버 명		역 할
long	reg[16]	ARM7 프로세서의 레지스터
long	CPSR	현재 프로세서 상태 레지스터(current processor status register)
icode_ptr	picode	thread가 실행할 다음 명령어(다음 PC의 명령어)
event_ptr	pevent	메모리 접근시, 벡-엔드와의 통신을 위한 이벤트 구조를 가리키는 포인터
long	paddr	이벤트 함수에 의해 계산되어진 물리적 주소
mint_time_t	time	thread가 생성되어 종료되기까지의 시간
mint_time_t	cpu_time	해당 thread가 사용한 누적된 cpu 시간
long	mem_map	실제 물리적 주소를 ARMint의 메모리영역의 주소로 변환시키기위한 베이스 주소
int	(*ufunc)	사용자가 정의하거나 이벤트에 관계된 함수의 함수포인터

< 표 2 프로세서 모델 구조체 (thread) >

3.2.1. 일반 명령어 실행부

ARMint에서는 각 프로세서 명령어(예를 들면, ADD, AND등)는 각기 하나의 함수로서 구현된다. 단, 이 함수들은 일반적인 명령어(시스템 호출에 관계된 소프트웨어 인터럽트 명령어(SWI)를 제외한 나머지들로 데이터 프로세싱 명령어와 메모리 접근 명령어, 그리고 코프로세서에 관련된 명령어를 의미한다.)들의 에뮬레이션 수행함수를 말한다 이 에뮬레이션 수행함수들은 위 3.1의 명령어 해석부분에서 설명한 명령어 형태의 명령어들을 구현했다.

3.2.2. 시스템 호출 실행부

시스템 호출은 ARM7의 소프트웨어 인터럽트 명령어를 이용한다. ARM에서의 시스템 호출은 크게 다음과 같은 두가지로 구현될 수 있다

ARMint에서의 시스템 호출은 위 표중에서 첫 번째 방식인 ARM tool kit version 2.11의 Angel 형식을 따른다 이 형식에서 시스템 호출은 역시 소프트웨어 인터럽트 명령어를 사용하며, 그 명령어코드는

구 현 방 식	특 징
Angel	특정 레지스터의 내용을 특정 메모리의 블록의 포인터로 사용하여 시스템 호출을 위한 정보를 전달하는 방식
ARM Debug Monitor(Demon)	실제로 명령어의 특정 비트를 해석하여 시스템 호출의 종류를 분석하고 정보를 전달하는 방식 (자세한 것은 ARM tool kit ver 2.11의 레퍼런스 가이드 참조)

< 표 3 시스템 호출의 구현 방식 >

SWI 0x12345678을 사용한다 레지스터 0번에 호출되는시스템 호출 번호가 저장되어 있다 [5]

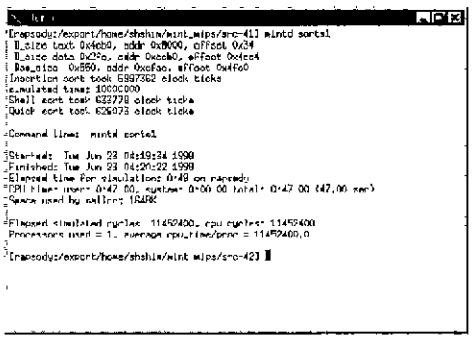
예) SYS_EXIT 시스템 호출

```
MOV R0, 0x18
SWI  0x12345678
```

4. 결과 및 향후 연구과제

ARMint는 현재 코프로세서에 관련된 부분과 64비트 결과를 갖는 곱셈, 그리고 일부 시스템 호출이 구현되지 않았다. 하지만 부동소수점을 사용하는 프로그램은 정수를 이용하여 부동소수점 계산을 수행하는 방법으로 실행이 가능하다.

테스트는 ARM tool kit 2.11 버전의 common 디렉토리 밑의 예제 디렉토리에서 dhrystone 프로그램과 sorts프로그램으로 테스트하였다.



< 그림 4. 벤치마크 프로그램인 sorts를 시뮬레이션 했을 때의 결과 >

향후에 수행할 연구과제로 그림 1의 백엔드부분, 즉 캐시와 메모리 부분에 대한 객체 지향적인 구조로의 설계가 요구된다.[2]

참고 문헌

[1] Jack E Veenstra, Robert J. Fowler "MINT Tutorial User Manual," Technical report 452, June 1993

[2] Mats Brorsson, Fredrik Dahlgren, Hankan Nilsson, and Per Stenstrom "The CacheMire Test Bench- A Flexible and Effective Approach for Simulation of Multiprocessor," proceedings of the 26th Annual Simulation Symposium, March, 1993

[3] Advanced RISC Machine "ARM 7TDMI data sheet," 1995

[4] Mary Lou Nohr "Unix System V Understanding Elf Object Files and Debugging Tools," Prentice-Hall, 1993

[5] Advanced RISC Machine "ARM Software Development Toolkit Version 2.11", 1997