한국시뮬레이션학회 논문지
Vol. 15, No. 4, pp. 51-58 (2006. 12)

명령어 집합 시뮬레이터를 이용한 임베디드 소프트웨어 디버거

정 훈[1] · 신동하[2†] · 손성훈[3]

An Embedded Software Debugger Using an Instruction Set Simulator

Jung Hun · Shin Dongha · Son Sunghoon

ABSTRACT

Debugging embedded softwares is very different from debugging general softwares. For examples, debugging embedded software requires more information, such as information on power consumption, information on the distribution of executed instructions, information on the distribution of used registers, and information on the amount of clocks consumed during the execution of a program, that is not needed in debugging general softwares. In this paper, we propose more effective method for debugging embedded softwares using an instruction set simulator for the microprocessor that is executing embedded softwares. In this research, we develop a debugger based on an instruction set simulator for a domestic embedded microprocessor called SE1608 and we shows an effective debugging method using a MiBench program which is widely used to benchmark embedded softwares. The debugging method proposed in this paper is relatively easy to implement and shows many advantages compared with existing debugging methods.

Key words : Debugger, Embedded software, Instruction set simulator, Microprocessor

요 약

임베디드 소프트웨어의 디버깅은 일반 소프트웨어의 디버깅과는 많이 다르다. 예를 들어 임베디드 소프트웨어 디버깅에는 일반 소프트웨어의 디버깅에는 필요하지 않는 전력 소비량에 대한 정보, 실행된 명령어 분포에 대한 정보, 사용된 레지스터 분포에 대한 정보, 프로그램 수행 시 소요된 클럭 수에 대한 정보 등이 추가적으로 더 필요하다. 본 논문은 임베디드 소프트웨어가 수행되는 마이크로프로세서의 명령어 집합 시뮬레이터를 이용하여 임베디드 소프트웨어를 효과적으로 디버깅하는 새로운 방법을 제안한다. 본 연구에서는 국산 임베디드 마이크로프로세서인 SE1608의 명령어 집합 시뮬레이터를 기반으로 디버거를 개발하고 이를 사용하여 임베디드 소프트웨어 벤치마크 프로그램으로 많이 이용되는 MiBench 프로그램을 사용하여 임베디드 소프트웨어를 효과적으로 디버깅하는 보기를 제시한다. 본 연구에서 제시한 디버깅 방법은 기존의 디버깅 방법에 비하여 비교적 구현하기도 쉬우면서 많은 장점이 있는 것으로 판단된다.

주요어 : 디버거, 임베디드 소프트웨어, 명령어 집합 시뮬레이터, 마이크로프로세서

1. 서 론

임베디드 소프트웨어는 일반 소프트웨어와는 다르게 여러 종류의 효율성(efficiency)을 요구한다(Wolf 2001). 임베디드 소프트웨어가 요구하는 효율성은 소비 에너지(energy)에 대한 효율성, 코드 크기(code size)에 대한 효율성, 수행 시간(run time)에 대한 효율성 및 가격(cost)에 대한 효율성 등을 포함하고 있다. 이러한 이유 때문에 임베디드 소프트웨어의 디버깅(debugging)에는 일반 소프트웨어의 디버깅에서 요구하지 않는 특별한 요구 사항이 존재한다. 예를 들어 임베디드 소프트웨어를 효과적으로(effectively) 디버깅하기 위해서는 주어진 소프트웨어의 수행에 따른 소비 에너지의 량, 사용 메모리(memory)의 량, 실행한 클럭(clock) 수 및 실행한 명령어(instruction)

2005년 10월 24일 접수, 2006년 11월 26일 채택
[1] 상명대학교 일반대학원 컴퓨터과학과 석사 과정
[1] 상명대학교 소프트웨어학부 부교수
[1] 상명대학교 소프트웨어학부 조교수
주 저 자 : 정 훈
교신저자 : 신동하
E-mail; dshin@smu.ac.kr

에 대한 분포 등의 정보가 필요하다. 하지만 이러한 정보를 제공하는 임베디드 소프트웨어 디버거(debugger)가 현재 없기 때문에 대부분의 임베디드 소프트웨어 개발자들은 일반 소프트웨어를 디버깅하기 위하여 개발된 GDB (Stallman 2006) 같은 디버거를 타겟(target) 하드웨어에 이식하여(port) 사용한다.

본 연구와 유사한 연구로 시뮬레이션 기술을 사용해 전력 소비량을 예측하는 연구(Contreras 2004)와 실행 클럭을 정확히 예측하여 전력 소비량을 예측하기 위한 연구(Šimunić 1999)가 있었다. 그러나 이러한 연구에서는 소프트웨어의 정확성 검증을 위한 디버깅 기능은 전혀 고려하지 않았다.

본 논문은 효율성 조건을 만족하는 임베디드 소프트웨어 개발에 적합한 새로운 디버거를 제안한다. 제안하는 디버거는 임베디드 소프트웨어를 수행하는 마이크로프로세서의 명령어 집합 시뮬레이터(instruction set simulator) (Cmelik 1994; Zhu 1999; Reshadi 2003)에 기반을 두고 있으며 기존의 소프트웨어 개발에 사용된 디버거가 제공하는 기능뿐만 아니라 임베디드 소프트웨어 디버깅이 요구하는 추가적인 기능을 제공한다. 일반 디버거가 소프트웨어 수행의 정확성(correctness) 검증에 중점을 두고 있다면 본 논문에서 제안하는 디버거는 소프트웨어 수행의 효율성(efficiency) 검증에 중점을 두고 있다. 이는 임베디드 소프트웨어 개발자에게는 매우 필요한 기능이었다.

명령어 집합 시뮬레이터는 마이크로프로세서의 명령어 수행을 컴퓨터 프로그램으로 구현한 소프트웨어로서 타겟 하드웨어가 구현되기 전에 개발된 소프트웨어를 수행시켜보기 위하여 사용되어 왔다. 본 연구에서 개발할 디버거가 명령어 집합 시뮬레이터에 기반을 두고 있는 이유는 명령어 집합 시뮬레이터가 마이크로프로세서의 명령어를 시뮬레이션하기 때문에 임베디드 소프트웨어를 구성하는 명령어를 수행하면서 생성되는 많은 정보들을 자연스럽게 접근할 수 있기 때문이다. 또한 임베디드 마이크로프로세서(Atmel 2006; Microchip 2006)는 일반 데스크 탑 혹은 서버 컴퓨터에서 사용되는 마이크로프로세서보다 매우 작고 단순하여 시뮬레이터의 구현이 비교적 쉽기 때문이다. 이는 임베디드 소프트웨어 개발자에게는 매우 중요한 요소이다. 왜냐하면 대부분의 임베디드 소프트웨어 개발자는 자기가 사용할 개발 도구를 본인 스스로가 개발하여 사용하여야 하는데 디버거의 구현이 용이하다는 것은 이들에게 큰 장점이 될 수 있기 때문이다.

본 연구에서는 에이디칩스 사에서 개발한 16비트 임베디드 마이크로프로세서인 SE1608(Advanced Digital Chips Inc. 2001, 2003)에 대한 시뮬레이션 기반 디버거를 개발하였다. 이 마이크로프로세서는 50Mhz로 동작되며 7개의 일반 목적 레지스터(R0 ~ R6), 프로그램 카운터(PC), 확장 레지스터(ER), 스택 포인터(SP) 및 상태 레지스터(SR)를 가지고 있으며 약 50개의 명령어를 제공한다. 개발한 시뮬레이터는 높은 수행 속도를 얻기 위하여 ELF(TIS Committee 1995) 형식으로 표현되는 SE1608 마이크로프로세서의 수행 파일을 C 언어 프로그램으로 작성된 시뮬레이터 프로그램으로 변환하는 컴파일(compile) 방식으로 구현되었다.

본 연구에서는 MiBench(Guthaus 2001)라는 임베디드 벤치마크 프로그램 중 산업 제어용으로 분류된 bitcount 프로그램을 본 연구에서 개발한 시뮬레이터 기반의 디버거를 사용하여 효과적인 프로그램 개발에 활용될 수 있음을 보인다. 특히 bitcount의 7가지 알고리즘을 임베디드 소프트웨어의 효율성 관점에서 디버깅하여 비교한 결과를 보인다.

본 논문의 구성은 다음과 같다. 1절 서론에서는 본 연구의 배경이 되는 연구의 필요성 및 본 연구의 특징에 대하여 기술하였고, 2절에서는 본 연구에서 개발하려는 디버거의 기능을 디버거 명령어를 나열하여 기술한다. 3절에서는 2절에서 제시된 기능을 가지는 시뮬레이터 기반 디버거의 전반적인 설계에 관하여 설명하고, 4절에서는 설계된 디버거의 구현에 관하여 기술한다. 5절에서는 임베디드 벤치마크 프로그램인 MiBench를 사용하여 개발된 디버거를 활용하는 예를 보인다. 마지막 6절 결론에서는 본 연구에서 개발한 디버거의 장점 및 개선점 등을 설명한다.

2. 시뮬레이터 기반 디버거의 기능

본 연구에서 제안하는 시뮬레이터 기반 디버거의 기능은 효율적인 임베디드 소프트웨어 개발에 중점을 두고 정의되었다. 다음의 표 1은 개발할 디버거의 기능을 명령어 기준으로 요약하여 나타낸 표이다. 아래 표의 디버거 명령어 중 제어 및 데이터 명령으로 분류된 명령어는 일반 디버거에서도 많이 제공되는 명령어인데 이들은 소프트웨어의 정확한(correct) 수행을 점검하기 위한 명령어들이다. 통계 명령으로 분류된 다양한 stat 명령어는 본 연구에서 제안하는 새로운 명령어로 효율적인(efficient)

표 1. 디버거 명령어 분류 표

분 류	명령어	설 명
제어 명령	break pc	break 지점 설정
	watch memory	memory 조건 break 설정
	watch register	register 조건 break 설정
	step	한 명령어 실행
	continue	프로그램 계속 실행
데이터 명령	print memory	memory 값 출력
	print register	register 값 출력
	edit memory	memory 값 수정
	edit register	register 값 수정
통계 명령	stat power	CPU가 소비한 에너지 양 출력
	stat instruction	실행한 명령어 수 출력
	stat cycle	실행한 클럭 수 출력
	stat memory	사용한 memory 양 출력
	stat register	사용한 register 양 출력

수행을 위한 임베디드 소프트웨어 개발에 사용되는 명령어이다.

3. 시뮬레이터 기반 디버거의 설계

본 절에서는 2절에서 설명한 기능을 가지는 디버거를 임베디드 마이크로프로세서의 명령어 집합 시뮬레이터 기반으로 설계하는 방법을 기술한다. 디버거를 명령어 집합 시뮬레이터 기반으로 구현하는 이유는 2절에서 기술한 통계 명령의 구현이 마이크로프로세서의 명령어 집합을 시뮬레이션하면서 쉽게 구현할 수 있기 때문이다. 일반적으로 명령어 집합 시뮬레이터의 구현은 인터프리터(interpreter) 방식과 컴파일(compile) 방식이 있다(Reshadi 2003). 인터프리터 방식은 실제 마이크로프로세서의 동작과 유사하게 가상 기계(virtual machine)의 메모리에 프로그램을 적재한(load) 후 프로그램 카운터(program counter)가 가리키는 명령어를 읽어 와서(fetch), 디코드(decode)한 후 수행하는(execute) 방식이다. 인터프리터 방식은 구현이 간단하고 자신의 코드 수정(self modifying code)이 가능하다는 장점이 있지만 수행 속도가 느리다는 단점이 있다. 컴파일 방식은 시뮬레이션할 프로그램을 주어진 언어(본 연구에서는 C 언어를 사용)로 표현되는 동등한(equivalent) 프로그램으로 변환한 다음 변환된 프로그램을 주어진 언어의 컴파일러(본 연구에서는 GNU C 컴파일러를 사용)를 사용하여 기계어로 변환하여

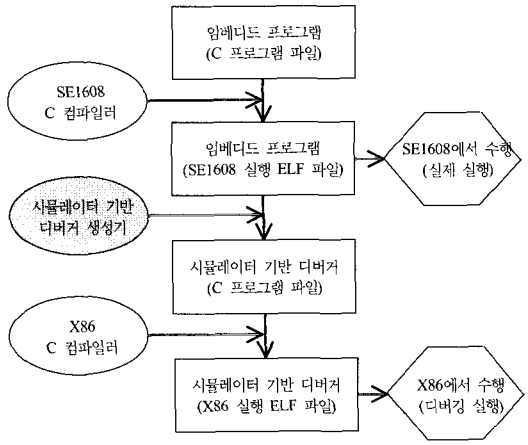

그림 1. 시뮬레이터 기반 디버거의 수행 절차도

수행하는 방식이다. 컴파일 방식은 자신의 코드를 수정할 수 없지만 수행 속도가 빠르다는 장점이 있다. 본 연구에서는 수행 시 코드 수정이 필요하지 않지만 빠른 수행이 필요하므로 컴파일 방식으로 시뮬레이터를 구현한다.

그림 1은 본 연구에서 개발하는 시뮬레이터 기반 디버거의 수행 절차도이다. 먼저 임베디드 프로그램 개발자가 작성한 임베디드 C 프로그램은 SE1608용 C 컴파일러에 의하여 SE1608에서 수행되는 실행 ELF(Executable and Linking Format)(TIS Committee 1995) 파일로 변환된다. 이 실행 파일은 본 연구에서 개발하는 시뮬레이터 기반 디버거 생성기에 의하여 C 프로그램 파일로 변환된다. 이 파일이 시뮬레이터 기반 디버거의 C 소스 파일이다. 이 파일은 최종적으로 X86용 C 컴파일러에 의하여 컴파일되어 X86에서 수행되면서 임베디드 프로그램 개발자가 작성한 프로그램을 디버깅한다.

시뮬레이터 기반 디버거 생성기는 SE1608 실행 ELF 파일을 입력으로 읽어서 C 프로그램을 생성하는 프로그램이다. 생성된 C 프로그램은 SE1608 실행 ELF 파일을 시뮬레이션하면서 2절에서 설명한 명령어를 입력받아서 디버깅 작업을 수행하는 프로그램이다. 그림 2는 시뮬레이터 기반 디버거 생성기 프로그램의 구조이다. ELF 파일 분석기는 입력 ELF 파일을 TEXT(명령어 집합), DATA(초기화된 전역 데이터) 및 BSS(초기화되지 않은 전역 데이터) 섹션으로 분리하는 프로그램이고, TEXT 섹션 변환기는 입력 TEXT 섹션의 명령어들을 읽어서 가상 기계에서 동등한 기능을 수행하는 함수로 변환하는 프로그램이며, DATA 섹션 변환기는 입력 DATA 섹션의 데이터를 읽어서 가상 기계를 구성하는 메모리의 DATA 섹

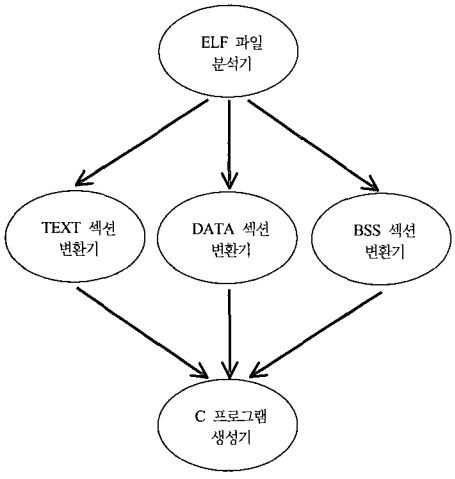

그림 2. 시뮬레이터 기반 디버거 생성기의 구조

션에 해당하는 자료 구조를 생성하는 프로그램이며, BSS 섹션 변환기는 입력 BSS 섹션의 위치 및 크기에 해당하는 가상 기계의 메모리를 생성하는 프로그램이다. 마지막으로 C 프로그램 생성기는 각 섹션별로 변환한 프로그램을 연결하고 가상 기계의 레지스터 등의 자료 구조를 생성하는 프로그램이다.

그림 3은 디버거 생성기에 의하여 생성된 디버거의 모습을 나타낸 그림이다. 생성된 디버거는 크게 가상 기계의 자료 구조 부분과 시뮬레이션할 명령어를 변환한 C 함수 부분으로 구성되어 있다. 생성된 디버거는 C프로그램이기 때문에 다시 X86 C 컴파일러에 의하여 컴파일되어 X86에서 수행된다. 이 프로그램은 최초 주어진 SE1608 실행 ELF 파일을 시뮬레이션하면서 2절에서 주어진 명령어에 대하여 응답함으로서 디버깅 작업을 수행한다.

시뮬레이터 기반 디버거 생성기에서 가장 중요한 부분은 TEXT 섹션 변환기이다. 이 변환기는 하나의 SE1608 명령어를 하나의 C 함수로 변환한다. 명령어 수행에 해당되는 C 함수는 다음과 같은 순서로 생성된다.

1. 명령어 주소로 구분되는 함수 선언
2. OPDATA에 따라 내부 변수 선언
3. 미리 정의된 명령어의 동작 구현 코드를 복사
4. 통계 데이터 계산 코드 복사

그림 5는 TEXT 섹션의 주소 0x43e0에 있는 SE1608 명령 "add %R0, 0x01, %R2"를 C 함수로 변환한 보기이다. 모든 명령은 이와 유사한 모습으로 변환된다.

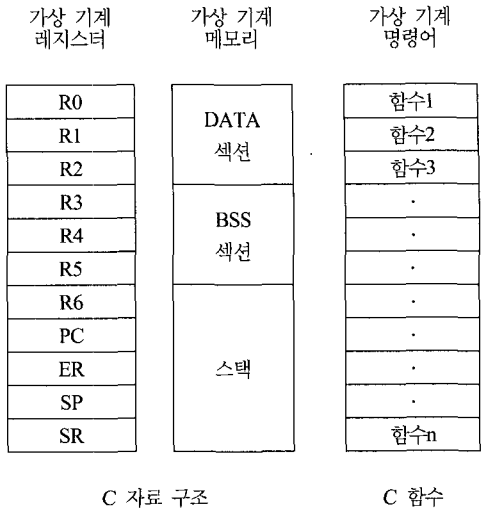

그림 3. 생성된 시뮬레이터 기반 디버거의 모습

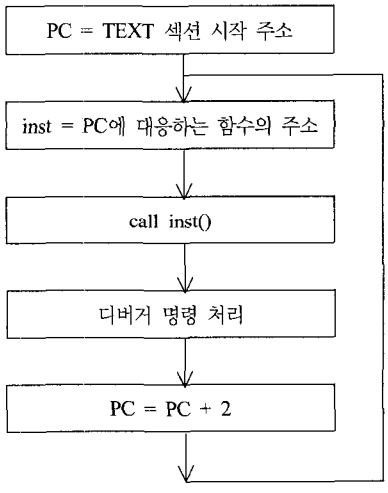

그림 4. 디버거의 명령어 수행 주 루프

TEXT 섹션의 모든 명령어가 C 함수로 변환되면 디버거가 수행되면서 명령어 주소에 따라 C 함수를 찾을 수 있도록 주소를 인덱스로 하는 함수의 주소 테이블을 생성한다. 변환된 C 함수는 그림 4와 같은 디버거의 명령어 수행 주 루프(main loop)의 내부에서 불려서 시뮬레이터 기반 디버거의 기능을 수행한다.

```
/* TEXT 섹션 주소: 0x43e0 */
/* 명령: add %R0, 0x01, %R2 */
void inst_43e0(void)
{
  unsigned char SrcRn = 0x00;
  unsigned short Immd = 0x0001;
  unsigned char DstRn = 0x02;

  if(SR의 E 값이 0이 아니면)
    Immd = (ER << 4) + Immd;
  Rn[DstRn] = Rn[SrcRn] + Immd;

  계산 결과에 따라 SR 값 수정;

  통계 데이터 계산;
}
```

그림 5. 명령 "add %R0, 0x01, %R2"의 수행 함수

4. 시뮬레이터 기반 디버거의 구현

3절에서 설계한 시뮬레이터 기반 디버거 생성기는 리눅스 커널 버전 2.6.15을 탑재한 Fedora Core 5 운영체제에서 C 언어로 구현되었다. 구현 시 ELF 파일 분석기 부분의 구현을 용이하게 하기 위하여 ELF 라이브러리(libelf-0.119.so)를 사용하였다.

디버거 생성기에 의하여 생성된 디버거는 ISO C를 만족하는 C 프로그램이기 때문에 어떤 하드웨어와 운영체제에서도 실행 가능하다.

5. 디버거 활용 보기

본 절에서는 MiBench라는 임베디드 벤치마크 프로그램을 사용하여 본 연구에서 개발한 디버거의 기능 중 통계 기능 위주로 활용하는 예를 보인다. MiBench는 2001년 미국의 미시간 대학에서 개발하여 공개 소스로 제공하는 임베디드 전용 벤치마크 프로그램으로서 총 6개 임베디드 분야에 대하여 벤치마크할 수 있게 구성되어 있다. 본 연구에서는 SE1608 마이크로프로세서에 가장 적합한 산업 제어용 벤치마크 프로그램인 bitcount 7개 알고리즘을 대상 프로그램으로 정하였다.

디버거 활용의 첫 번째 보기로 bitcount 7개 알고리즘을 각각 수행하면서 마이크로프로세서가 소비하는 에너지의 양을 출력하였다. 프로그램이 수행하면서 마이크로프로세서가 소비하는 에너지의 양은 프로그램 수행 시 실행되는 각 명령이 소비하는 에너지의 양을 합한 값이다

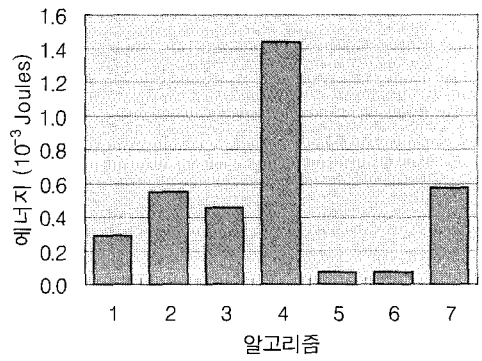

그림 6. 마이크로프로세서의 소비 에너지양 비교

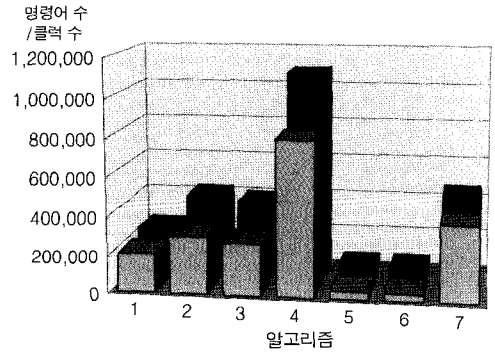

그림 7. 실행 명령어 수 및 실행 클럭 수의 비교

(Tiwari 1994, 1996). 본 연구에서는 이전 연구(신동하 2006)에서 측정한 SE1608의 각 명령어에 대한 에너지 소비 값을 활용하였다. 그림 6은 bitcount 7개 알고리즘 수행 시 마이크로프로세서가 소비하는 에너지양을 비교한 그래프이다. 이 그래프에서 알고리즘 5 및 6의 에너지 소비량이 가장 적음을 알 수 있다. 만약 개발하려는 임베디드 시스템에서 에너지 소비가 중요한 요소라면 알고리즘 5 혹은 6의 사용을 고려하면 좋을 것이다.

디버거 활용의 두 번째 보기로 bitcount 7개 알고리즘의 실행 명령어 수 및 실행 클럭 수를 출력하였다. 일반적으로 실행 클럭 수는 수행 시간에 비례한다. 그림 7은 bitcount 7개 알고리즘의 수행 명령어 수 및 실행 클럭 수를 비교한 그래프이다. 이 그래프에서 알고리즘 5 및 6이 가장 작은 실행 명령어 수 및 실행 클럭 수를 보이고 있다. 그림 6과 그림 7을 비교하면 소비 에너지와 실행 클럭 수는 유사한 모양의 그래프를 나타냄을 알 수 있다.

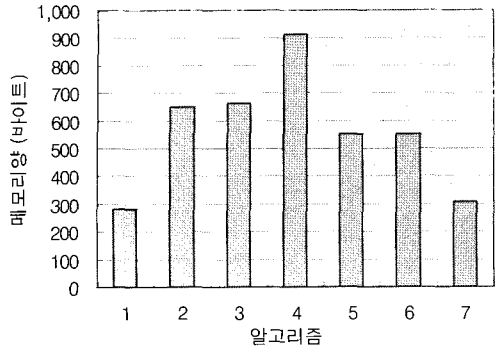

그림 8. 사용한 메모리양의 비교

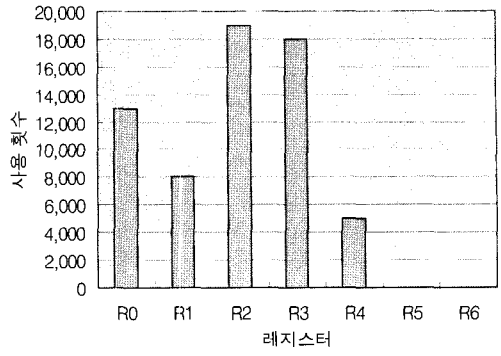

그림 9. 각 레지스터의 사용 횟수

디버거 활용의 세 번째 보기로 bitcount 7개 알고리즘의 실행 시 사용된 메모리의 양을 출력하였다. 일반적으로 어떤 프로그램이 수행되면서 필요한 메모리의 양은 컴파일 시간에 정해지는 TEXT 섹션, DATA 섹션, BSS 섹션의 크기의 합에 수행 시간에 함수 부름(function call) 시 사용한 STACK의 크기를 더한 값이다. 그림 8은 bitcount 7개 알고리즘의 수행 시 사용한 메모리의 양을 비교한 그래프이다. 이 그래프에서 알고리즘 4가 가장 많은 양의 메모리를 사용한다는 것을 알 수 있다. 만약 개발하려는 임베디드 시스템에서 메모리의 크기가 중요한 요소라면 알고리즘 4의 사용은 피해야할 것이다.

디버거 활용의 네 번째 보기로 bitcount 7개 알고리즘의 실행 시 각 레지스터의 사용 횟수를 출력해 볼 수 있다. 대부분의 마이크로프로세서는 명령어 수행 시 사용되는 레지스터에 따라 전력 소비가 다른 것으로 알려져 있다. 이전 연구에 의하면 SE1608 마이크로프로세서의 경우 레지스터 R3, R5, R6을 사용하는 경우가 레지스터 R0, R1, R2, R4을 사용하는 경우보다 약 1.5% 정도의 전력 소비양이 큰 것으로 측정되었다. 각 레지스터의 사용 횟수를 바탕으로 소비 에너지의 양을 줄일 수 있는 최적화가 가능하다. 그림 9는 bitcount 7개 알고리즘 중 알고리즘 5의 수행 시 각 레지스터의 사용 횟수를 나타낸 그래프이다. 레지스터는 명령어에서 소스(source) 레지스터 혹은 목적(destination) 레지스터로 사용될 수 있다.

앞에서 보인 보기 외에도 본 연구에서 개발한 디버거를 사용하면 마이크로프로세서가 수행한 각 명령 별로 에너지 사용량, 수행 횟수 등을 출력해 볼 수 있다. 그림 10은 bitcount 7개 알고리즘 중 알고리즘 5의 수행 시 각 명령 종류별로 수행한 명령의 횟수를 그래프로 나타낸 그림이다.

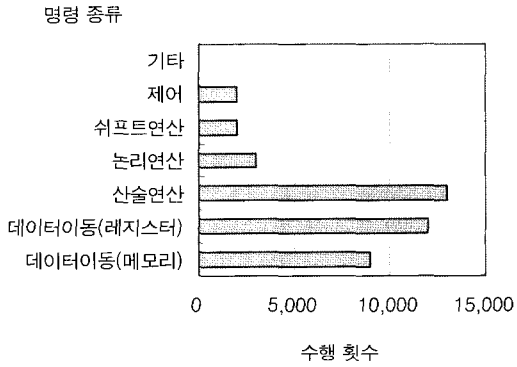

그림 10. 사용한 명령 종류의 수행 횟수

6. 결 론

임베디드 소프트웨어는 일반 소프트웨어에서는 요구하지 않는 여러 가지 효율성을 요구하고 있다. 예를 들어 소비 에너지(energy)에 대한 효율성, 코드 크기(code size)에 대한 효율성, 수행 시간(run time)에 대한 효율성 및 가격(cost)에 대한 효율성 등이 그것이다. 본 논문은 일반 소프트웨어의 디버깅에 필요한 기능뿐만 아니라 이러한 효율성도 디버깅이 가능한 임베디드 소프트웨어 디버깅 방법에 대하여 연구하였다. 본 연구에서는 명령어 집합 시뮬레이션 기술을 적용하여 디버거를 개발하였는데 그 이유는 마이크로프로세서의 명령어 집합을 시뮬레이션함으로서 이러한 효율성 정보를 쉽게 얻을 수 있기 때문이다. 본 논문에서는 임베디드 마이크로프로세서인 SE1608의 시뮬레이션을 바탕으로 디버거 생성기를 설계하고 구현하였으며 임베디드 벤치마크 프로그램인 MiBench를 사용하여 실제 활용하는 예도 보였다.

본 연구에서 제안한 디버거는 다음과 같은 장점을 가지고 있다. 첫째, 정확성 위주의 디버깅을 강조하는 기존의 디버거(Stallman 2006)와 달리 프로그램의 정확성뿐만 아니라 효율성까지도 디버깅이 가능하여 임베디드 소프트웨어 디버깅에 매우 유용하다. 둘째, 대부분의 임베디드 소프트웨어 개발자들은 사용하는 마이크로프로세서의 명령어 집합에 대한 사전 지식을 가지고 있기 때문에 명령어 집합 시뮬레이션 기반의 디버거 개발에 추가적인 기술적 교육이 필요하지 않다. 이러한 이유 때문에 임베디드 소프트웨어 개발자는 일반 디버거를 개발하여 사용하는 것 보다 본 연구에서 제안하는 디버거를 개발하는데 노력이 덜 드는 장점이 있다. 셋째, 본 연구에서 제안하는 디버거 개발 시 부산물로 발생하는 명령어 집합 시뮬레이터는 임베디드 하드웨어의 개발이 완료되지 않은 상황에서 개발한 소프트웨어를 수행해 볼 수 있는 장점이 있어서 임베디드 소프트웨어 개발 기간을 단축 할 수도 있다. 넷째, 최근 고 사양의 임베디드 마이크로프로세서(Intel Corporation 2004)는 성능 카운터(performance counter)를 통해 본 디버거가 제공하는 유사한 기능을 하드웨어 자체에서 제공하기도 한다. 하지만 아직도 임베디드 시장에서 대부분을 차지하는 8비트 및 16비트 마이크로프로세서는 이러한 기능이 제공되지 않는다. 본 디버거는 이러한 저사양의 임베디드 마이크로프로세서를 사용하는 개발자에게는 더욱 장점이 될 수 있다. 본 연구에서 개발한 디버거는 아직 시제품 수준이다. 앞으로 개발자가 실제 개발에 활용하여도 손색이 없을 정도로 개발하기 위해서는 많은 개선이 필요할 것으로 보인다.

참 고 문 헌

1. Advanced Digital Chips Inc. (2001), *SE1608 Core Manual Extendable Instruction Set Computer*, Ver. 2.0.
2. Advanced Digital Chips Inc. (2003), *ADC16S310 USB 1.1 Core Embedded Microprocessor*, Ver. 0.1.
3. ATMEL Corporation (2006), *8-bit AVR Microcontroller with 8K Bytes In-System Programmable Flash, ATmega48, ATmega88, ATmega168 Automotive.*
4. Cmelik, B. and Keppel, D. (1994), "Shade: A Fast Instruction-Set Simulator for Execution Profiling", *ACM SIGMETRICS Performance Evaluation Review*, Vol.22 No.1, pp. 128-137.
5. Contreras, G., Martonosi, M., Peng, J., Ju, R. and Lueh, G. (2004), "XTREM: a power simulator for the Intel XScale® core", *Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pp. 115-125.
6. Guthaus, M. R., Ringenberg, J. S., Emst, D., Austin, T. M., Mudge, T. and Brown, R. B. (2001) "MiBench: A Free, Commercially Representative Embedded Benchmark Suite", *IEEE 4th Annual Workshop on Workload Characterization*. pp. 3-14.
7. Intel Corporation, (2004), *Intel XScale® Core Developer's Manual.*
8. Microchip Technology Inc. (2006), *PIC24FJ128GA Family Data Sheet General purpose, 16-Bit Flash Microcontrollers.*
9. Reshadi, M., Mishra, P. and Dutt, N. (2003), "Instruction Set Compiled Simulation: A Technique for Fast and Flexible Instruction Set Simulation", *Proceedings of The 40th Conference on Design Automation*, pp. 758-763.
10. Šimunić, T., Benini, L. and Micheli, G. (1999), "Cycle-accurate simulation of energy consumption in embedded systems", *Proceedings of the 36th ACM/IEEE conference on Design automation table of contents*, pp. 367-872.
11. Stallman, R., Pesch, R., Shebs, S., et al (2006), *Debugging with GDB The GNU Source-Level Debugger*, Ninth Edition.
12. TIS Committee (1995), *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification*, Ver. 1.2.
13. Tiwari, V., Malik, S. and Wolfe, A. (1994), "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization", *IEEE Transactions on VLSI Systems*, Vol.2, No.4, pp. 437-445.
14. Tiwari, V., Malik, S., Wolfe, A. and Lee, M. T. (1996), "Instruction Level Power Analysis and Optimization of Software", *Journal of VLSI Signal Processing Systems*, Vol.13, pp. 223-238.
15. Wolf, W. (2001), *Computers as Components Principles of Embedded Computing System Design*, Morgan Kaufmann Publishers.
16. Zhu, J. and Gajski, D. D. (1999), "A Retargetable, Ultra-Fast Instruction Set Simulator", *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 298-302.
17. 신동하, 강경희 (2006), "임베디드 마이크로프로세서에서 산술 및 논리 명령어에 대한 전력 예측 모델", *한국해양정보통신학회 논문지*, Vol.10, No.8, pp. 1422-1427.

정 훈 (powerloki@smu.ac.kr)

2005년 상명대학교 소프트웨어학부 학사
2005년~현재 상명대학교 일반대학원 컴퓨터과학과 석사 과정

관심분야 : 명령어 집합 시뮬레이션, 임베디드 소프트웨어 디버깅, 저 전력 임베디드 소프트웨어 등

신 동 하 (dshin@smu.ac.kr)

1980년 경북대학교 전자공학과 학사
1982년 서울대학교 전자계산기공학과 석사
1994년 University of South Carolina 컴퓨터과학과 박사
1982년 한국전자통신연구원 책임연구원
1997년~현재 상명대학교 소프트웨어학부 부교수

관심분야 : 저 전력 임베디드 소프트웨어, 실시간 임베디드 운영체제, 임베디드 리눅스 운영체제 등

손 성 훈 (shson@smu.ac.kr)

1991년 서울대학교 계산통계학과 학사
1983년 서울대학교 전산과학과 석사
1999년 서울대학교 전산과학과 박사
1999년 한국전자통신연구원 선임연구원
2004년~현재 상명대학교 소프트웨어학부 조교수

관심분야 : 저 전력 임베디드 소프트웨어, 임베디드 리눅스 운영체제, 멀티미디어 시스템 등