x86 프로세서 이해

김도연*, 안현수*, 전재욱*
*성균관대학교 전자전기컴퓨터공학과
rainbox91@skku.edu, ahs128@skku.ac.kr, jwjeon@yurim.skku.ac.kr

Understanding of x86 processor architecture

Do Yeon Kim*, Hyun Su An*, Jae Wook Jeon*
* Department of Electrical and Computer Engineering

요 약

본 논문은 공학계열 학생들이 어셈블리 프로그래밍을 통해 x86 프로세서를 학습하는 교육 과정을 소개한다. 이 교육 과정은 어셈블리 프로그래밍을 통해 가상머신에서 프로그램을 실행시켜 학생들이 전공 교과 과정에서 학습한 마이크로프로세서 이론의 이해를 향상시키도록 도와준다. 작성된 어셈블리 파일은 NASM 을 이용하여 컴파일 되고, VMware 의 Workstation Player 가 컴파일러에 의해 생성된 바이너리 파일을 실행시키기 위해 사용되었다. 교육 과정은 마이크로프로세서 이론 수업에 맞추어 과제가 학생에게 주어지고, 학생들은 이론 수업의 이해를 바탕으로 결과물을 완성하고 이를 직접 시연하여 평가받았다.

1. 서론

마이크로프로세서는 중앙 처리 장치와 동일한 전자 로직과 DIP 로 패키지 된 단일 칩을 의미한다. [1] 마이크로프로세서는 특히 임베디드 시스템(Embedded System)에 주로 사용된다. 임베디드 시스템은 목표하고자 하는 기능을 위해 마이크로프로세서에 적재된 프로그램이 여러 하드웨어 자원을 제어한다. 또한 임베디드 시스템은 특정 기능의 집합을 원하고자 하는 제약을 충족시키면서 구현된다. 그 제약에는 성능, 비용, 배출량, 전력 소비 및 무게와 같은 것들이 있다. [2] 이런 임베디드 시스템은 보통 코어, ROM, RAM, 주변 장치들로 구성된다. [3]

마이크로프로세서는 크게 두가지의 ISA(Instruction Set Architecture)로 나뉘어진다. 하나는 CISC 인 x86 ISA 이고, 다른 하나는 RISC 인 ARM ISA 이다. 스마트폰과 태블릿은 ARM ISA 를 실행하며, 데스크톱 및 노트북은 x86 ISA 를 실행한다. [4] 어떠한 프로그램을 만들 때 특정 ISA 에서 실행되도록 프로그램의 소스코드는 컴파일 및 기계어 형태로 해석되어야 한다. [5] 따라서 마이크로프로세서를 활용하여 임베디드 프로그래밍을 할 시에, 자신이 사용하는 프로세서의 ISA 가 무엇인지를 파악해야 한다. 특히 마이크로프로세서를 이용한 프로그래밍은 기계어에 더 가까운 어셈블리 언어로 주로 이루어지므로, ISA 의 깊은 이해가 필수적이다.

주변의 사물들, 즉 여러 임베디드 시스템이 네트워크를 통해 인터넷에 접속되어 임베디드 시스템 간의 커뮤니케이션이 가능하며, 그로 인해 사용자에게 새로운 서비스가 가능하다는 개념의 IoT(Internet of Things) 기술이 여러 산업에서 각광받고 있다. [6] 데스크톱, 노트북, 스마트폰, 태블릿과 같은 기본적인 전자기기에서 웨어러블 디바이스, 스마트홈 기기까지 다양한 IoT 제품들이 쏟아져 나오고 있다. 따라서 미래 엔지니어가 될 공학계열 학생들에게 마이크로프로세서에 대한 교육을 양질의 제공하는 것이 필요하다. 특히 x86 ISA 가 모바일 저전력 장치 시장에 진입하고 있는 만큼, CISC 의 특성과 구조를 이해하고 활용할 수 있는 미래 인력 양성이 필요하다.

x86 프로세서 아키텍처는 ARM 프로세서 아키텍처에 비해 명령어가 복잡하다. 따라서 학생들이 처음 x86 프로세서 아키텍처를 접했을 때 어려움이 많다. 본 논문은 전공 교과 과정에서 학습한 x86 마이크로프로세서 이론에 대한 이해를 더 깊고 넓게 하도록, 어셈블리 프로그래밍을 통해 학생들의 이해력 향상을 도와줄 수 있는 교육 과정을 소개한다.

학생들이 작성한 어셈블리 코드 파일은 NASM 를 이용하여 컴파일되고, VMware 의 Workstation Player 가 컴파일러에 의해 생성된 실행 파일을 재생한다. 학생들은 Workstation Player 를 이용하여 특정 레지스터의 값이나, 메모리의 값들을 출력하여 확인할 수 있다.

본 논문의 구성은 다음과 같다. 2 장에서는 학생들의 개발환경에 대해 소개한다. 3 장에서는 교육 과정

에 대해 상세히 서술한다. 4 장에서는 평가 방법에 대해 설명하며, 마지막으로 5 장에서 논문을 끝맺는다.

2. 개발 환경

2.1. Notpad++

x86 프로세서의 이해를 위해 어셈블리 프로그래밍을 할 때 IDE(Integrated Development Environment)는 사용하지 않는다. 따라서 어셈블리 코드를 작성할 에디터 프로그램이 필요하다. 본 논문이 제안하는 교육 과정에서는 오픈 소스 프로그램인 Notepad++가 이용된다.

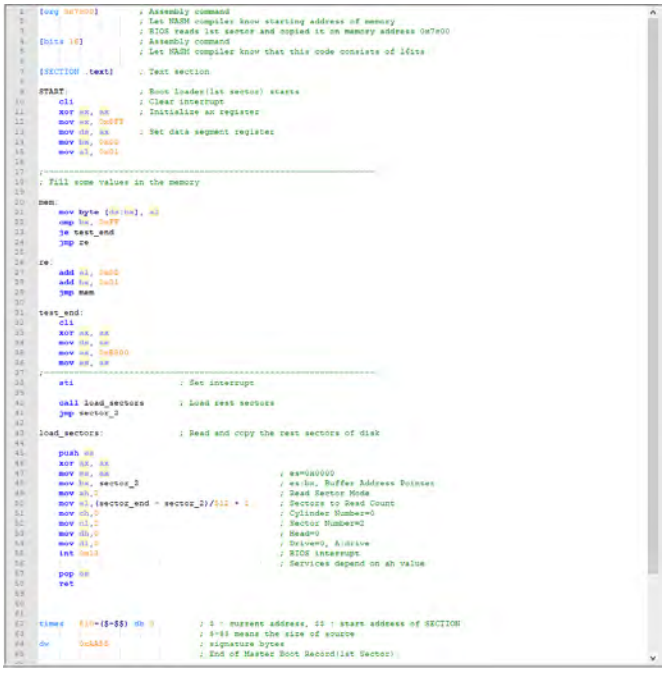
2.2. NASM

NASM (Netwide Assembler)은 x86 아키텍처용 어셈블러이다. 오픈 소스 소프트웨어이며, 교육 과정에서는 2.11.08 버전을 이용한다. 예를 들어 어셈블리 소스코드인 filename.asm 파일을 filename.bin 실행파일로 컴파일을 하고자 한다면 다음과 같은 명령어를 사용하면 된다.

nasm -f filename.bin filename.asm

2.3. Workstation Player

컴퓨터에서 보조 운영 체제를 실행할 수 있도록 해주는 VMware 의 Workstation Player 는 NASM 으로 컴파일 되어 생성된 바이너리 파일을 실행시켜주는 역할을 하는 가상머신 프로그램이다. x86 아키텍처 기반으로 어셈블리 코드가 작동하도록 하는 부트로더가 담긴 소스코드를 학생들에게 제공하여 개발 환경이 제대로 구성이 되었는지 확인한다. 학생들에게 제공된 부트로더 코드는 (그림 1)과 같다.



(그림 1) x86 프로세서 부트로더 어셈블리 소스 코드.

3. 교육 과정

수업은 6 주로 매주 1 시간씩 교육이 진행된다. 학생들은 어셈블리 프로그래밍을 통해 x86 프로세서 아키텍처를 더 심도 깊게 이해할 수 있다. 또한 학생들에게 매주 수행해야할 과제가 부여된다. 이 교육 과정에서 학생들은 매주 3 시간씩 전공 교과 과정에서 학습한 x86 프로세서 이론을 실습한다. <표 1>은 본 논문이 제안하는 교육 과정을 보여준다.

<표 1> 교육 과정

주차	강의주제
1	강의 소개
2	Real Mode 에서 문자열 출력
3	Protected Mode 에서 GDT, LDT
4	Far Jump, Far Call/Return, Call Gate Descriptor
5	TSS 와 Task Switching
6	IDT 와 ISR

3.1. 강의 소개

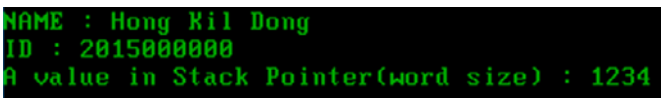
학생들에게 강의 일정과 함께 개발 환경을 소개하며 학생들과 함께 단계별로 개발 환경을 구성한다. 또한 제공한 부트로더 소스코드가 잘 실행이 되는지를 확인한다.

3.2. Real Mode 에서 문자열 출력

먼저 NASM 어셈블러에 대해 설명한다. NASM 에서 사용하는 명령어들을 설명하고, NASM 이 허용하는 구문과 허용되지 않는 구문을 학생들이 직접 코딩하여 확인하게 한다.

그리고 1 주차에 제공했던 부트로더 어셈블리 코드를 풀어서 설명한다. 특히 선행된 x86 프로세서 이론에서 학습한 Real Mode 에서의 세그먼트 주소지정 방식에 대해 복습한다. 또한 문자열을 출력하는 방법을 설명한다. 이때 x86 프로세서는 ARM 프로세서와 다르게 리틀 엔디안으로 동작한다는 것을 다시 한번 상기시킨다. Workstation Player 에 출력하기 위한 엑스트라 세그먼트를 이용하여 메모리 주소를 지정하는 방법에 대해 설명한다.

교육 후, 학생들에게 첫번째 과제가 주어진다. 과제는 학생들에게 첫번째 과제용으로 배포한 어셈블리 코드를 이용하여 (그림 2)와 같이 학생의 이름과 학번, 그리고 제공된 스택 포인터의 값을 출력하도록 한다.



(그림 2) 첫번째 과제의 결과물의 출력 화면.

3.3. Protected Mode 에서 GDT, LDT

학생들은 전공 교과에서 이미 배운 Real Mode 와 Protected Mode 에 대해 복습하게 된다. Real Mode 로 전환하기 위해 GDT(Global Descriptor Table)와 GDTR 레지스터를 어떻게 활용하는 지에 대해 다룬다. 이때 GDT 의 주소를 GDTR 에 저장하기위해 lgdt 명령어가 사용됨을 학생들에게 상기시킨다. 더불어 CR0 레지스터의 PE(Protection Enabled) 비트가 0 일시 Real Mode 로 동작하고, 1 일시 Protected Mode 로 동작함을 설명한다.

또한 전공 교과 이론 수업에서 다루지 않은 내용인 어셈블리 코드 내의 레이블을 이용하여 코드 세그먼트를 점프하는 방법도 설명한다. 그리고 어셈블리 코드를 이용하여 GDT, LDT(Local Descriptor Table) 생성 방법도 학생들은 학습하게 된다.

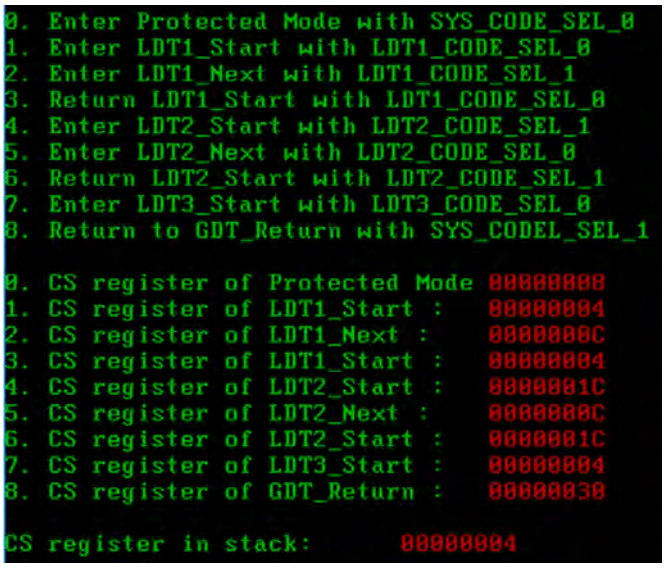
교육 후, 학생들에게 두번째 과제가 주어진다. 학생들에게는 먼저 GDT 와 LDT 를 어떻게 구성해야 하는지에 대한 정보가 주어진다. 또한 학생들은 두번째 과제용으로 배포한 어셈블리 코드가 제시하는 흐름으로 다른 코드 세그먼트로 이동시키는 코드를 작성해야 한다. 만약 학생이 올바른 어셈블리 코드를 작성했다면, Workstation Player 에서는 (그림 3)과 같이 강의자가 의도한 코드 세그먼트의 값이 출력된다.

(그림 3) 두번째 과제의 결과물의 출력 화면.

3.4. Far Jump, Far Call/Return, Call Gate Descriptor

학생들은 전공 교과 수업에서 학습한 Far Jump, Far Call/Return, Call Gate Descriptor 에 대해 복습한다. 특히 Far Jump, Far Call/Return 명령어가 사용됐을 때, 스택에 CS, EIP, 호출 위치가 저장됨을 상기시킨다.

교육 후, 학생들에게 세번째 과제가 주어진다. 학생들에게는 두번째 과제와 마찬가지로 먼저 GDT 와 LDT 를 어떻게 구성해야 하는지에 대한 정보가 주어진다. 그리고 학생들은 세번째 과제용으로 배포한 어셈블리 코드를 이용하여 Far Jump, Far Call/Return, Call Gate Descriptor 로 코드 세그먼트를 넘나드는 코드를 구현해야 한다. 만약 학생이 올바른 어셈블리 코드를 작성했다면, Workstation Player 에서는 (그림 4)과 같이 코드 세그먼트의 이동 흔적이 출력된다.

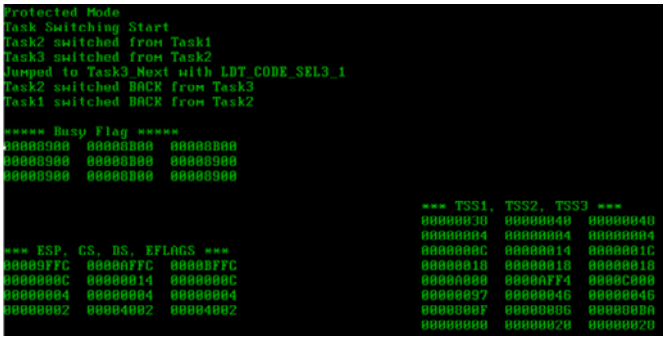


(그림 4) 세번째 과제의 결과물의 출력 화면.

3.5. TSS 와 Task Switching

학생들은 전공 교과 수업에서 학습한 TSS(Task State Segment)와 CALL/IRET 명령어를 이용한 태스크 스위칭에 대해 복습한다. 또한 태스크 스위칭을 Task Gate Descriptor 를 사용하거나 사용하지 않는 방법이 있음을 상기시킨다.

교육 후, 학생들에게 네번째 과제가 주어진다. 학생들은 네번째 과제용으로 배포한 어셈블리 코드를 이용하여 여러 태스크들을 스위칭 하는 어셈블리 코드를 작성해야 한다. Workstation Player 에서는 (그림 5)와 같은 화면이 출력된다. 상단에는 태스크 스위칭 흐름이 출력되고, 하단에는 각종 레지스터의 값들이 출력된다.



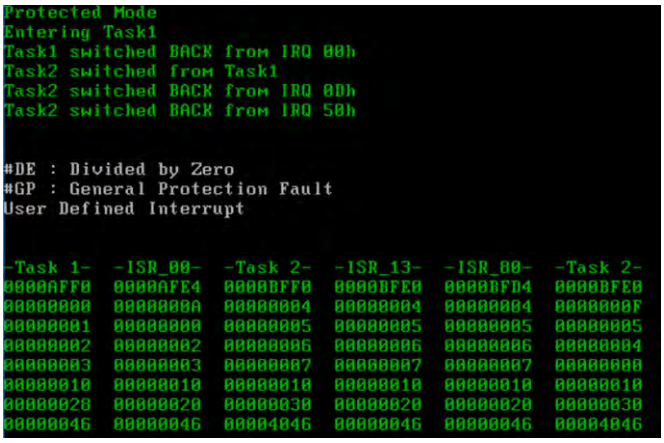
(그림 5) 네번째 과제의 결과물의 출력 화면.

3.6. IDT 와 ISR

학생들은 전공 교과 수업에서 학습한 IDT(Interrupt Descriptor Table)에 대해 복습한다. Exception 과 ISR(Interrupt Service Routine)의 차이점에 대해 자세히 설명한다. 또한 ISR 이 실행되었을 때 스택에 EFLAGS, CS, EIP, Error Code 가 Push 되고, iret 명령어를 사용하였을 때 EFLAGS, CS, EIP 가 Pop 되는 것을 다

시한번 강조한다.

교육 후, 학생들에게 마지막 과제가 주어진다. 학생들은 마지막 과제용으로 배포한 어셈블리 코드를 이용하여 여러 태스크들을 스위칭 하면서 Exception 과 ISR 을 발생시켜야 한다. Workstation Player 에서는 (그림 6)과 같이 프로그램의 흐름이 출력된다.



(그림 6) 마지막 과제의 결과물의 출력 화면.

4. 평가 방법

이 교육 과정의 목표는 어셈블리 프로그래밍을 통해 x86 프로세서의 이해의 향상을 도모하는 것이다. 따라서 모든 주차에서 전공 교과 수업에서 학생들이 학습한 이론 내용을 반드시 복습하도록 되어있다. 학생들은 매주 부여된 과제를 통해 어셈블리 프로그래밍을 하여 매주 전공 교과 수업에서 배운 이론을 복습하고, 깊게 이해하는 시간을 가질 수 있다.

모든 과제는 1 주일의 시간이 주어지며, 매주 강의 시간에 학생들의 결과물 시연을 통해 과제에서 요구하는 값들의 출력 여부에 따라 PASS/FAIL 이 평가된다. 만약 요구사항을 하나라도 충족하지 못하였다면 그 학생은 FAIL 로 평가된다. 또한 시연 평가 후에 각 학생들의 코드를 비교하는 표절 검사도 진행된다.

또한 본 논문이 소개하는 교육 과정은 이론 수업으로 이루어진 전공 교과 과정과 함께 맞물려 진행되는 수업이다. 학생들은 해당 전공 교과목에서 본 논문이 소개한 교육 과정과 관련된 문제가 포함된 시험을 봄으로써 한 번 더 평가를 받게 된다. 시험에는 여러 참고자료가 학생들에게 주어지므로, 학생의 수업에 대한 이해가 중요하다. 시험 문제 중 한 문제는 매년 이 논문이 제안한 교육 과정에서 학생들이 학습한 모든 내용을 망라하는 어셈블리 코드가 학생들에게 주어졌다. 학생들은 그동안 학습한 내용을 토대로 어셈블리 코드를 분석하고 문제의 답을 채우면 된다.

따라서 학생들은 과제에서만 평가를 받는 게 아니라, 이론 수업에서도 이 논문이 소개하는 교육 과정 내용을 평가 받는다.

5. 결론

이 논문에서는 어셈블리 프로그래밍을 통해 x86 프로세서의 아키텍처의 이해 능력을 향상시키는 교육 과정을 제안하였다. 특히 매주 진행되는 과제를 통해서 학생들은 전공 교과 수업으로 먼저 학습한 내용을 복습하고 실습하는 경험을 쌓는다. 학생들은 이론 수업에서 학습한 내용을 매주 복습하고, 그것과 관련된 내용으로 과제를 수행해야 한다. 따라서 성실히 과제를 제출한 학생들의 경우 x86 프로세서를 더 넓고 깊게 이해하는 것이 가능했다.

대다수의 학생들이 충실하게 과제를 해왔으며, 매주 이론 내용으로 과제를 진행해야 했기에 학생들의 전공 교과 수업에서 높은 집중력을 보여주었다. 다만 초반에 낙오된 학생들의 경우 교육 과정이 끝날 때까지 과제를 수행해오지 못하는 모습을 보였다. 따라서 이런 학생들에 대한 추가적인 지도와 관심이 필요하다고 판단된다.

ACKNOWLEDGMENT

This work was supported in part by the Ministry of Science and ICT (MSIP), South Korea, through the G-ITRC Support Program supervised by the Institute for Information and Communications Technology Promotion (IITP) under Grant IITP-2018- 20150-00742.

참고문헌

[1] Oshorne, Adam. "An Introduction to Microcomputers: Volume I, Basic Concepts." (1980).

[2] Sangiovanni-Vincentelli, Alberto, and Grant Martin. "Platform-based design and software design methodology for embedded systems." IEEE Design & Test of Computers 18.6 (2001): 23-33.

[3] Camposano, Raul, and Jörg Wilberg. "Embedded system design." Design Automation for Embedded Systems 1.1-2 (1996): 5-50.

[4] Blem, Emily, Jaikrishnan Menon, and Karthikeyan Sankaralingam. "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures." 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2013.

[5] Karaki, Hussein, Haitham Akkary, and Shahrokh Shahidzadeh. "X86-ARM binary hardware interpreter." 2011 18th IEEE International Conference on Electronics, Circuits, and Systems. IEEE, 2011.

[6] Bandyopadhyay, Debasis, and Jaydip Sen. "Internet of things: Applications and challenges in technology and standardization." Wireless personal communications 58.1 (2011): 49-69.