플래시를 이용한 능동적 컴퓨터구조 학습도구의 개발

이강†, 서희암°
한동대학교 전산전자공학부
† yk@handong.edu, °bycydy@hotmail.com

Development of Active Learning Tool for Computer Architecture using Flash

Kang Yi, Heeam Seo
School of Computer Science and Electronic Engineering, Handong Global University, Pohang

요 약

본 논문에서는 폰노이만 컴퓨터의 동작 원리를 쉽게 이해하도록 도와주는 애니메이션 학습 도구의 개발을 소개한다. 개발된 프로그램은 일정 제약조건을 만족하는 임의의 명령어 집합 구조를 가상으로 시뮬레이션 하고 각 명령어의 실행과정을 플래시 애니메이션을 사용하여 학습자에게 보여준다. 시뮬레이션 대상의 ISA 설정과 메모리의 내용을 변경함으로써 임의의 프로세서구조에 대한 명령어의 실행을 시뮬레이션할 수 있다. 본 학습 도구는, 애니메이션을 이용함으로써 초보자라도 학습 내용을 쉽게 이해하게 할 수 있게 하였고 사용자와 쌍방향 의사소통이 가능한 능동적 학습을 가능하게 하고 여러 수준의 학습자를 수용할 수 있는 장점이 있다.

1. 서 론

컴퓨터 구조에 대한 기초 이해는 많은 사람들에게 전공을 불문하고 요구되고 있다. 전산학을 전공하는 저학년은 물론이고, 전산학을 전공하지 않더라도 교양으로 정보처리 개론 등의 과목에서 폰노이만 컴퓨터 구조의 기본을 배운다. 그러나, 대부분의 학생들이 책에 있는 글로 된 내용만으로는 명령어와 데이터가 어떤 방식으로 읽혀지고 실행되는지 정확하게 이해하지 못하는 경우가 많다. 따라서, 글이나 단순한 그림이 아닌 실제 컴퓨터 내부에서의 명령어와 데이터의 흐름을 동영상으로 보여주는 애니메이션 학습 도구의 개발이 요청된다.

IT 교육 분야에서 학습자 스스로가 능동적으로 컴퓨터 시뮬레이션 도구를 통하여 학습의 여러 중요한 개념을 쉽고도 빠르게 이해하도록 하는 연구가 많이 되어 왔다 [1,2,3]. 자바나 ActiveX 컨트롤 등을 이용하여 멀티미디어 도구를 만들어서 논리 회로, 기초회로 이론, 전자기학 이론 등에 관한 원리 설명을 시도하여 왔다. 그러나, 컴퓨터 구조 과목에 관한 멀티미디어 학습 도구는 아직 없으며 애니메이션을 이용한 학습 도구도 찾아볼 수 없다.

본 논문에서는 폰노이만 컴퓨터 구조의 특성을 알기 쉽게 애니메이션으로 보여주는 도구를 설계하고 개발하였다. 폰노이만 방식 컴퓨터 구조의 특성인 동일 메모리에서 명령어와 데이터가 저장된다는 사실과 프로그램 카운터의 역할과 개념, 각 기계어 명령어의 실행 절차 및 다양한 어드레싱 모드에 대한 설명을 애니메이션으로 보여줌으로써 학습자에게 컴퓨터 구조의 주요 개념을 이해시키는 것을 목적으로 한다. 본 연구에서 개발된 학습 도구는 사용자가 자신만의 ISA (Instruction Set Architecture)를 자유롭게 표현할 수 있도록 허용하고 있다. 개발도구는 사용자와 프로그램간의 쌍방향 의사소통을 지원하고 애니메이션 표현 능력이 탁월한 플래시를 채택하였다.

본 논문의 구성은 다음과 같다. 다음 2 장에서는 플래시를 이용한 컴퓨터 구조 학습도구의 설계 및 구현을 설명하고, 3장에서는 이 도구를 이용하여 실제로 간단한 8비트 프로세서에 대한 ISA를 모델링하여 작동하시키는 예시를 제시한다. 4장에서는 요약 및 향후 과제에 대한 제시를 하며 결론을 내린다. 부록에서는 임의의 ISA를 자유롭게 표현할 수 있는 외부 설정 파일의 문법 전체를 제시한다.

2. 학습 도구의 설계 및 구현

2.1 전체 흐름도

다음 그림 1은 본 학습 도구의 작동 환경 및 흐름을 보여주고 있다. 제작된 플래시는 일정 조건을 만족하는 임의의 ISA를 애니메이션으로 동작을 보여줄 수 있다. 사용자는 외부 설정 파일(Configuration File) 을 통해서 시뮬레이션의 대상이 되는 마이크로프로세서의 ISA를 자세히 기술하도록 되어 있다. 그림1에서 Our Flash Program은 본 논문에서 제작한 플래시로 된 실행 파일 프로그램이다. Configuration은 학습할 대상이 되는 프로세서의 ISA와 각 명령어 별로 애니메이션 동작 스텝을 기술한 텍스트 파일이다. Memory Contents는 주어진 ISA 상에서 실

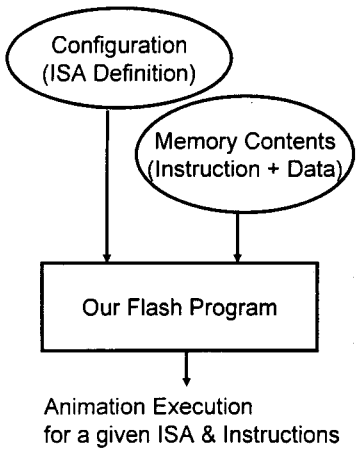


그림 1. 학습 도구 소프트웨어의 전체적인 흐름

행하려는 기계어로 된 명령어들과 데이터들의 이진값을 기술한 파일을 의미한다. 즉, Configuration과 Memory Contents를 사용자가 바꿈으로써 원하는 ISA를 가진 프로세서의 원하는 기계어 프로그램의 실행을 가상적으로 실행하여 애니메이션으로 학생들에게 프로세서 동작을 보여줄 수가 있는 것이다.

2.2 제약사항

다음 표 1은 본 학습도구에서 보여줄 수 있는 프로세서의 ISA의 옵션 및 제약사항을 설명하고 있다. 즉, 본 학습 도구는 표 1과 같은 제약사항을 만족하기만 하면 어떤 프로세서의 ISA 라도 표현이 가능하다.

2.3 플래시를 이용한 구현 방식

본 논문에서 제작한 Flash Program은 위의 제약 조건하에서 주어진 외부 설정에 따라서 다른 그림들을 보이고 다르게 동작해야 한다. 즉, 명령어의 길이, 레지스터의 개수 명령어 양식, 주소 모드 등을 모두 가변시킬 수 있어야 한다. 이를 위해 허용되는 모든 ISA를 다 수용할 수 있도록 필요한 애니메이션 동작과 장면들의 그래픽 컴포넌트들을 미리 구현하여서 하나의 플래시 실행화일 내에 모두 집적시킨다. 프로그램의 실행 시에 외부 설정에 따라서 여러 숨은 기능들 중에서 필요한 부분만 사용자에게 보여진다. 예컨대, 미리 플래시 실행화일 내에 4개의 레지스터를 위한 그래픽을 구현해두고 1개의 레지스터만 필요한 경우에는 애니메이션 상에는 레지스터가 하나뿐인 그래픽으로 보여준다.

한 화면에 메모리의 내용과 레지스터의 내용이 모두 보이게 하고 명령어의 실행에 따라 메모리와 레지스터의 내용이 바뀌는 것을 애니메이션으로 보여준다. 명령어의 실행을 Fetch-Decode-Execute의 3단계로 나누어서 실행을 순차적으로 보여준다. 명령어의 작동을 보여주는 방식은

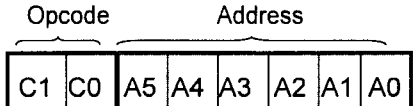
표 1. 표현 가능한 명령어의 조건들

제약조건 범주	제약사항
명령어길이	8비트 또는 16비트 (가변길이 수용)
주소 모드	직접 어드레싱 (direct) 레지스터 어드레싱 (register) 레지스터 간접 어드레싱 (register indirect) 즉치 어드레싱(immediate)
범용 레지스터의 개수	1개 (AC만 사용하는 경우), ~ 4개
ALU의 연산자	덧셈, 뺄셈, AND, OR, XOR, NOR, NOT, 좌우쉬프트
플래그 종류	N(Negative),Z(Zero),C(Carry)
Opcode의 길이	1비트 ~ 명령어길이
한 명령어 내의 피연산자의 개수	1개 ~ 3개
외부 데이터 버스 폭	8비트
외부 주소 버스 폭	8비트 또는 16비트
외부 메모리 개수	1개 (명령어와 데이터 메모리 구분 없음)

한 명령어씩, 한 RTL 동작별, 연속 실행의 3가지의 방식이 제공된다.

3. 8비트 마이크로프로세서 교육에의 응용

그림 2는 제안된 도구의 활용성을 보여주기 위한 예시로 만든 간단한 8비트 프로세서의 명령어 형식을 나타내고 있다. 예제로 사용한 이 마이크로 프로세서는 명령어 양식의 C1과 C0 두 비트에 의해서 LD, ST, ADD, JZ의 4가지 명령어를 나타내다. 이 프로세서는 하나의 AC 레지스터만을 범용 레지스터로 가지며, 주소 모드는 직접 어드레싱만 존재한다고 가정한다.



C1 C0	Pseudo	Description
0 0	LD	AC ← M[A5...A0]
0 1	ST	M[A5...A0] ← AC
1 0	ADD	AC ← AC + M[A5..A0]
1 1	JZ	PC ← A5...A0

그림 2. 예제 8비트 프로세서의 명령어 양식

다음 그림 3은 위 예시 프로세서의 실행을 위한 메모리 내용 입력 장면이다.

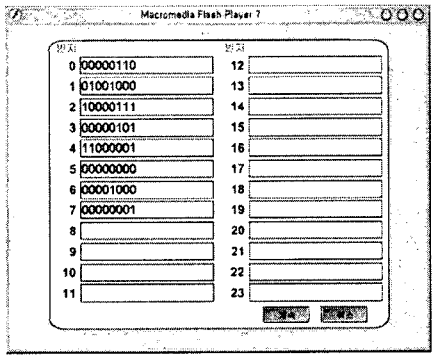


그림 3. 예제 8비트 프로세서의 메모리 내용 입력 장면

다음 그림 4는 위 예시 프로세서의 실행을 위한 첫 명령어를 메모리에서 읽어오는 장면이다.

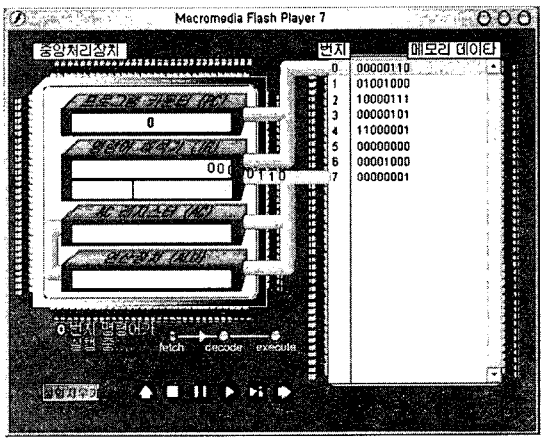


그림 4. 예제 8비트 프로세서의 첫 명령어 읽혀오는 장면

다음 그림 5는 위 예시 프로세서의 LD 명령어 실행을 위한 데이터 주소가 메모리로 전달되는 장면이다.

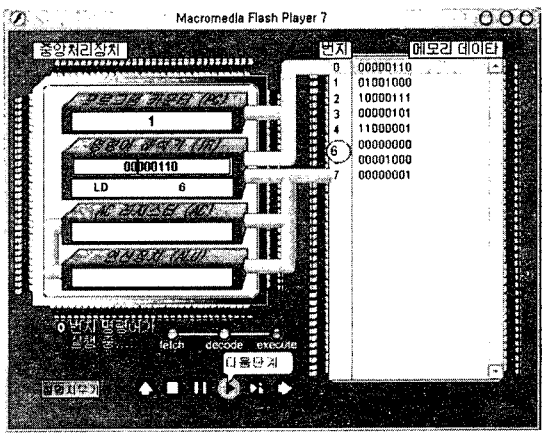


그림 5. 예제 8비트 프로세서의 LD 명령어 실행 장면

4. 결론

본 논문에서는 컴퓨터 구조를 쉽게 이해할 수 있도록 돕는 애니메이션 학습도구를 플래시를 이용하여 개발하였다. 여러 다양한 마이크로프로세서의 명령어 구조를 수용할 수 있도록 ISA의 임의 설정이 가능하다. 앞으로, 사용자가 ISA를 직관적으로 입력할 수 있도록 하는 입력 도구의 개발과 RISC 파이프라인에 대한 이해를 돕는 애니메이션이 추가될 필요가 있다.

참고문헌

[1] 김동식, 한희진, 서삼숙, 김숙희, " ActiveX 컨트롤을 이용한 단순화된 웹기반 디지털 논리회로 시뮬레이터" , 공학교육학회 논문지, 제 6권 1호, pp. 5~14, 2003년 6월

[2] 김영선, 최경, 이기식, " 멀티미디어를 이용한 웹기반의 전자기학 교재 개발" , 공학교육학회 논문지, 제 6 권, 호 2, pp. 42-53, 2002년 12월.

[3] 멀티미디어를 이용한 웹기반 디지털 논리회로 가상현실의 구현, 공학교육학회 논문지, 제 5권 1호, 2002년 6월.

부록. Configuration File의 전체 Syntax (BNF 형식)

```
configuration_file ::= reg_def_part instr_def_part alu_part opr_part
opaction_part inst_description_part

reg_def_part::= regnum regname
regnum ::= regNumber = integer
regname ::= regName= string

instr_def_lpart::= instMaxLength = integer {instr_def}
instr_def ::= formatName_def  formatLength_def {field_def}
formatName_def ::= formatName format_id = string
formatLength_def ::= formatLength format_id = integer
field_def::= field_size field_name field_type
field_size ::= fieldtBit format_id field_id = integer
field_name ::= fieldName format_id field_id = string
field_type ::= fieldType  format_id field_id = using_type
usingtype::="operator"|"regsource"|"regtarget"|"immeadiate"

alu_part ::= {aluop aluop_id = aluOperation}
aluOperation ::= "+"| "-" | "OR" | "AND" | "XOR" | "NOR" | "NOT"
| "SHIFTL" | "SHIFTR"
aluop_id ::= integer

opr_part ::= {opr_code opr_format opr_name opr_desc}
opr_code ::= opr  opr_id = bin_string
opr_format ::= oprFormat opr_id =string
opr_name ::= oprName opr_id =string
opr_desc ::= oprDesc opr_id=string
opr_id ::= integer

opaction_part::= {opaction opr_id  seq_id = direction}
direction ::="PC2IR"|"PC2Maddr"|"IR2PC"|"IR2Maddr"|"Maddr2PC"|
"Maddr2IR"|"IR2Reg"|"IR2ALU"|"IR2Mdata"|"Reg2IR"|"Reg2ALU"|"R
eg2Mdata"|"ALU2IR"|"ALU2Reg"|"ALU2Mdata"|"Mdata2IR"|"Mdata2
Reg"|"Mdata2ALU"|"ifZ"|"ifC"|"ifN"|"ifReg"
seq_id ::= integer {c}
instr_description_part::= {instDesc opr_id  seq_id  = string }
```