論文 87-24-3-10

마이크로 아키텍춰 시뮬레이터

(A Micro-Architecture Simulator)

朴 炳 寬,* 裵 相 德,* 徐 大 和,* 尹 龍 鎬*

(Byung Kwan Park, Sang Duck Bae, Dae Wha Seo and Yong Ho Yoon)

要 約

마이크로 프로그램은 프로세서의 제어 구분을 구현하는 가장 중요한 기술이다. 그러나 마이크로 프로그램은 새로운 컴퓨터 시스템을 개발할 때 가장 시간을 많이 들여야하는 지루한 작업이다. 본 논문에서는 이러한 마이크로 프로그램의 개발 과정에서 유용하게 이용할 수 있는 대화 형식의 마이크로 아키텍춰 시뮬레이터(Conversational Micro Architecture Simulator : C-MAS)의 개발에 관해 기술한다. C-MAS 는 마이크로 프로그램 개발 도구 뿐만 아니라 마이크로 프로그램의 수행시의 환경을 분석하기 위한 도구로도 사용하였다. 본 논문에서는 마이크로 아키텍춰의 기술을 위한 언어에 대해 고찰하고, 설계 과정에서 고려했던 점 그리고 사용자 환경에 대해 기술한다. 개발한 시뮬레이터는 Berkeley UNIX 4.2 환경에서 C언어를 사용하여 개발하였다.

Abstract

The microprogram is the key technology of the implementation of the processor's control unit. But the coding and testing it is the most tedious process in the developing a new computer system. We developed the conversational micro architecture simulator (C-MAS) in order to use it as a microprogram development tool and a run time analyzer of the microprogram. We discuss the hardware description language (HDL) for a simulation, the design constraints of the C-MAS, and the user interface of it in this paper. We used the c language as the description language, and developed it on the berkeley UNIX 4.2.

I. 序 論

50년대말에 소개된 마이크로 프로그램은 IBM의 system/360 family의 개발에 사용된 이후[1] 현재 거의 모든 프로세서의 제어부분을 구현하는 기술로 사용하고 있다. 마이크로 프로그램 기술은 새로운 마이크로 프로세서의 개발이나 기존 아키텍춰의 구현(emulation) 뿐만 아니라, vertical migration[2], dynamic microprogramming[3] 등 그 영역을 넓혀가고 있다. 다양화하는 마이크로 프로그램의 이용에 따라 이를 개발함에 있어 효율적인 도구(tools)와 기술(techniques)에 대한 고려가 대두되고, 그 결과 "Firmware Engineering"의 중요성이 강조되고 있다.[4]

초기의 마이크로 프로그래머들은 하드웨어가 완성되기 전까지는 마이크로 프로그램 시험을 할 수 없었고, 완성된 하드웨어 위에서 시험을 할 경우에도 시험을 지원할 만한 적절한 도구가 없었기 때문에 하드웨어 에러와 마이크로 프로그램 에러를 구별하는 데에도 많은 시간을 들여야 했다. 많은 마이크로 프로그래머블한 시스템이 CPU를 구현하는데 bit-sliced processor

*正會員, 韓國電子通信研究所, 컴퓨터 開發部
(Electronics and Telecommunications Research Institute)

接受日字 : 1986年 8月 6日

를 연결 사용함이 많아짐에 따라, 이를 이용한 시스템의 개발도구로써 여러 development 시스템들이 개발되어 왔다.[5,6] 이러한 시스템들은 타겟 하드웨어와 인터페이스 하는 부분이 있어 타겟의 상태 등을 추적하는 기능과 prototype 하드웨어를 쉽게 빨리 구성할 수 있도록 마이크로 프로그래머블한 머쉰을 만들때 기본적으로 필요한 하드웨어(제어메모리, sequencer, 등등)를 제공하고 있다. 타겟과 거의 동일한 환경에서 프로그램을 시험할 수 있는 장점이 있지만, 하드웨어를 우선 구성하여야만 시험이 가능하고, 여러 프로그래머가 공유할 수 없으며 낮은 수준의 시험환경이 단점이다.

마이크로 아키텍춰의 소프트웨어 시뮬레이션은 일반적인 시뮬레이션과 같이 실제의 구현이 없이 각종 시험을 할 수 있는 장점과 마이크로 프로그램의 고유한 특징 때문에 좋은 시험 도구로 사용되고 있다. 마이크로 아키텍춰 시뮬레이터의 장점은 다음과 같다.

• 마이크로 프로그램의 시험과 디버깅을 하드웨어없이 할 수 있으므로, 하드웨어의 구현과 마이크로 프로그램을 병렬로 할 수 있어 전체의 개발 시간을 단축할 수 있다.

• 마이크로 프로그램의 시험을 위해 직접 하드웨어를 이용할 경우 비싼(일반적으로 마이크로 프로그램을 할 수 있는 시스템은 중형급 이상임으로) 시스템을 공유하기가 어렵지만, 소프트웨어 시뮬레이션은 쉽게 여러 사람이 공유할 수 있어 경제적이고 많은 개발 시간의 단축을 꾀할 수 있다.

• 하드웨어를 사용할 경우 발견하기 어려운 에러(stack overflow, incorrect memory access, …)를 잡는 기능 등 세련된 시험환경(sophisticated testing environment)을 제공함으로써 지루하고 많은 시간을 요구하는 디버깅 작업의 부담을 덜어줄 수 있다.

• Trace 기능 등을 이용하여 마이크로 프로그램과 하드웨어의 성능 측정을 할 수 있어 취약점 분석 등에 용이하다.

• 발견된 하드웨어의 취약점 등의 수정이 필요할 경우 쉽게(하드웨어 보다) 변형이 가능하다.

• 처음 마이크로 프로그래머가 하드웨어를 이해할 때 manual을 통하게되지만 시뮬레이터를 같이 사용하게 되면 더욱 쉽고 정확하게 이해할 수 있으므로 프로그램 시간의 단축 및 프로그램 에러 발생의 가능성을 줄일 수 있다.

본 논문에서는 F/4000 시스템의 CPU인 MPU 29[7]의 대화 형식의 마이크로 아키텍춰 시뮬레이터인 Conversational Micro Architecture Simulator (C-MAS)의 구현에 관해 기술한다. MPU29는 2901[8] bit-slice processor를 사용한 32비트, 마이크로 프로그램이 가능한 하드웨어로써, F/4000 시스템은 마이크로 프로그램을 이용하여 IBM System/370[9] 아키텍춰를 에뮬레이션(emulation)하고 있다. 64비트 수평적인 마이크로 인스트럭션를 제공하며, 타겟(System/370)아키텍춰의 효율적인 에뮬레이션을 위해 residual control[10] 방법을 많이 사용하는 것이 특징이다.

C-MAS는 마이크로 프로그램의 모듈화를 위한 연구[11]의 일환으로, 마이크로 프로그램의 run time environment의 분석과 마이크로 프로그램의 구조적인 특징 분석, 그리고 새로운 마이크로 프로그램의 개발 도구로 사용키 위해 개발했다. C-MAS는 시뮬레이션의 대상이 되는 MPU 29의 기술 부분(simulation nucleus)과 사용자 환경(user interface)과 시스템 관련 환경(system dependent environment, memory, bus, …) 등 크게 세 부분으로 구성되어 있다. 대화 형식이 사용자 환경을 제공하고 있으며, C언어[19]로 구현하였다.

2 장에서는 하드웨어 기술 언어(hardware description language)에 대해 고찰하며 본 시뮬레이터에서 사용한 언어의 선정 배경에 대해 언급한다. 3 장에서는 시뮬레이터의 구성과 제어의 흐름에 대해서 설명하며, 4 장에서는 시뮬레이터의 핵심 부분인 하드웨어 자원의 기술 방법과 기술상 고려했던 점에 대해 설명하고, 5 장에서는 시뮬레이터의 구성, 사용자 대화 부분의 구조, 그리고 사용자에게 제공되는 기능들에 대해서 기술한다. 끝으로 결론과 앞으로의 계획에 대해 언급한다.

Ⅱ. 하드웨어 기술 언어의 고찰

하드웨어 기술은 그림 1과 같이 설계 자동화(design automation) 도구(예를 들면 시뮬레이터, Verification system, high level microprogramming language (HLML) compiler, compactor, 등등)들에 입력되어 각 도구들이 특정 하드웨어에만 국한되지 않고 여러 시스템에도 사용할 수(retargetable) 있게 한다. 이런 하드웨어의 기술을 위해 하드웨어의 구조와 특성에 맞는 언어들이 계속 개발되고 있는데, 기술하는 하드웨어의 수준(circuit level, gate level, register transfer level …)에 따라 여러 기술 언어들이 개발되었고, 최근 연구 동향은 각 수준에 모두 사용할 수 있는 HDL에 대해 관심이 집중되고 있다.[12]

HDL은 크게 분류하면 기능적인 것(functional)과 구현적인 것(implementational)으로 나누어 진다. 기능적인 것은 대상 하드웨어의 입력과 출력의 관계를 기준으로 기술하는 언어이고, 구현적인 것은 하드웨어 각 소자의 연결 상태와 각 소자의 기능을 기술함으로써 하

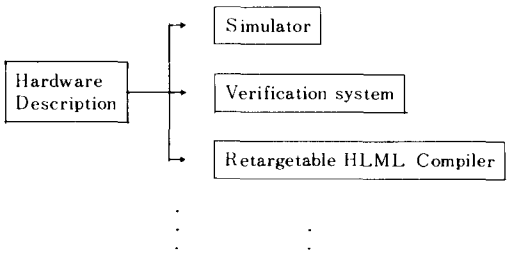


그림 1. 설계 자동화 시스템의 HDL 이용
Fig. 1. HDL in the Design Automation System.

드웨어를 표현하는 것이다. 전자는 주로 시뮬레이션과 하드웨어의 증명(verification)에 주로 이용되고 후자는 하드웨어의 자동화 설계에 적합하다.

그러나 현재의 HDL은 아직 언어와 하드웨어 사이의 차이를 완전히 메우고 있지 못하기 때문에 올바른 하드웨어의 설계 자동화나 쉬운 하드웨어의 기술이 실현되고 있지는 못하다. 또한 적절한 시뮬레이션 수준에 맞는 HDL을 선택하지 못할 경우는 본래의 의도와 매우 벗어나 매우 비효율적이거나 하드웨어의 기술이 어려울 수 있다.[14] 현재, 기계어 수준의 기술 언어인 ISPS[13]가 가장 많이 사용되는 HDL중 하나로 컴퓨터 아키텍춰의 시뮬레이션에 많이 이용되고 있다.

마이크로 아키텍춰의 시뮬레이션의 경우, 레지스터 전달 수준이라고 쉽게 생각할 수 있지만 다음과 같은 마이크로 아키텍춰의 특징은 HDL의 선정에 어려움을 주고 있다.[15]

• Types of microarchitecture : 수평적(horizontal) 마이크로 아키텍춰의 경우 마이크로 오퍼레이션의 수준은 레지스터 전달 수준의 기초적인 기능을 수행하지만, 수직적인(vertical) 경우 마이크로 오퍼레이션은 일반적인 머쉰 인스트럭션과 거의 기능상 동일 수준이 된다.

• Residual control : 마이크로 인스트럭션이 직접 마이크로 오퍼레이션을 조정하는 직접제어 방식(immediate control)에 반해 residual 제어방식은 residual control register에 임의의 값이 들어가면 다음 주기에 기능을 수행하거나 마이크로 인스트럭션의 디코드(decode)를 변화하게 한다.

• Timing : 다양한 clock 제어방식(single clock/multiple clock, single phase/multiple phase 등등)은 마이크로 아키텍춰의 기술상에 문제점을 제시하고 있다.

일반적인 HDL 시뮬레이터는 사용자와의 대화 부분이 비효율적이고(batch style), HDL 기술이 옳고 그름을 시험하는 도구들이 제공되지 않음으로 시뮬레이터의 증명이 힘든 단점이 있다. 이와 같은 이유때문에 기능적인(functional) HDL로 일반적인 고급언어를 사용하는 것이 많은데, 이는 소프트웨어 공학적(S/W engineering) 측면에서 볼 때도 세련된 개발환경을 제공하는 LISP, C, prolog와 같은 언어가 기능적인 HDL보다 훨씬 유리하다.[16]

본 연구에서는 이와 같은 여러 점들을 고려하여 Berkeley UNIX 4.2 환경에서 여러가지 개발 도구들이 제공되는 C언어를 사용하여 하드웨어를 기술하였고, 사용자 환경을 대화형식으로 구현한 시뮬레이터를 개발하였다.

Ⅲ. C-MAS의 구성

그림 2는 C-MAS의 구조를 기능적인 블럭으로 나타낸 것이다. 그림에서와 같이 C-MAS는 크게 3가지의 블럭으로 구성되어 있는데, 시뮬레이터의 핵심인 하드웨어의 기술부분과 주위의 하드웨어 부분인 머쉰 관련 환경, 그리고 사용자와의 정보교환을 위한 사용자 환경이 그것이다. 위에서 언급한 3부분과는 별도로 외곽에 존재하는 블럭들은 C-MAS가 시뮬레이션 중에 사용하는 화일과 테이블들이다.

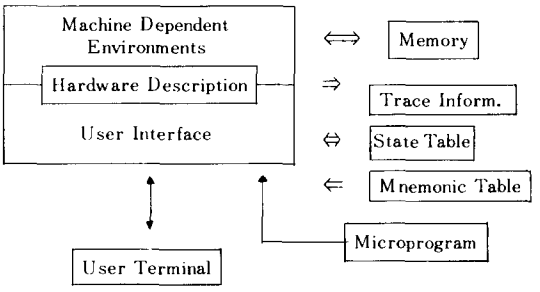


그림 2. C-MAS의 구성
Fig. 2. The Structure of C-MAS.

하드웨어 기술부분은 MPU29와 동일한 상태를 갖도록 C언어로 기술한 부분으로써 자세한 부분은 4장에서 언급한다. 사용자 환경은 마이크로 프로그램의 수행으로 상태가 변화하는 MPU29의 상태를 사용자에게 보여주고 또한 사용자가 시뮬레이션의 제어를 할 수 있도록 시뮬레이터의 제어를 담당한다.

MPU29는 시뮬레이션의 대상으로, F4000 시스템의 CPU에 해당하고, 버스에 연결되어 메모리와 입출력 프로세서와 같이 동작한다. 머쉰관련 환경(machine dependent environments)은 MPU29를 시뮬레이션 할 때 최소하게 필요한 MPU29의 주변의 하드웨어 들로

써, 시스템 버스와 메모리 등을 지칭한다. 입출력 프로세서는 MPU29의 시뮬레이션과는 거의 관계가 없기 때문에 버스와 메모리만을 본 시뮬레이터에서는 구현하였다. 메모리는 UNIX의 화일 형태로 존재하며, 메모리의 읽고 씀은 lseek( ), read( ), write( ), system call로 구현했다.

Mnemonic Table은 마이크로 프로그램의 disassemble을 위한 것으로 사용자가 새로운 mnemonic을 사용코자 할 경우에 C-MAS의 프로그램의 수정없이 이 테이블만을 고침으로써 가능하다. Trace information 은 시뮬레이션 과정중에서 수행한 마이크로 프로그램의 제어의 흐름을 담는 화일이다. State information은 시뮬레이터의 상태를 담고 있는 화일로써 임의의 시뮬레이터의 상태를 만들어서 시험에 사용하거나 시험중인 시뮬레이터의 상태를 저장하여 반복 사용하는데 필요하다. 사용자는 단말기를 통하여 모든 자원의 내용을 직접 볼 수 있고 또한 자원의 상태를 변형할 수 있는 대화형식으로 이루어지며, 시험 할 마이크로 프로그램은 binary 화일로 시뮬레이터에 공급된다.

그림 3은 C-MAS의 제어의 흐름을 나타내고 있다.

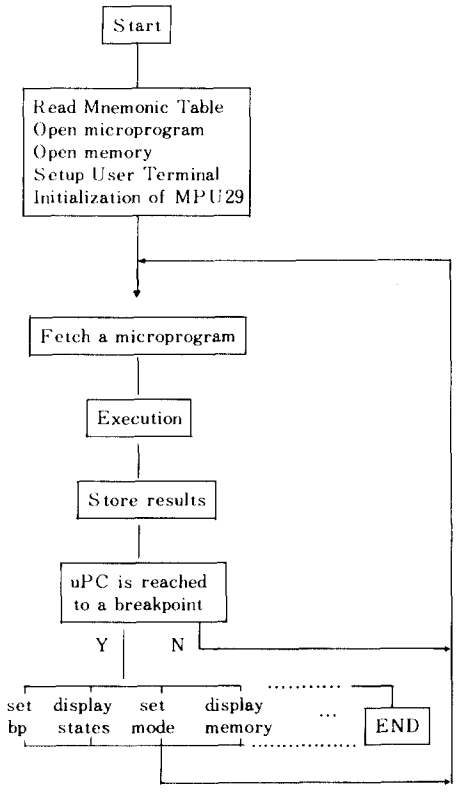


그림 3. C-MAS의 제어 흐름도
Fig. 3. The Flow Chart of C-MAS.

Ⅳ. 마이크로 아키텍춰의 기술(Description)

기능적인 하드웨어의 기술은 하드웨어의 구조나 구성 요소에 관계없이 입출력 신호간의 기능적인 관계를 기술함으로써 가능하다. 그러므로 일반 언어로도 기능적인 기술을 할 수 있는데 다음과 같은 점을 염두에 두면 보다 나은 하드웨어의 기술을 할 수 있다.

첫째, 하드웨어와의 대응 관계를 유지한다. 기능적인 기술은 하드웨어의 구성과는 전혀 관계없이 기술할 수 있지만 하드웨어의 확장이나 보완을 위한 수정 등을 고려할 때 가능한 한 하드웨어의 부품이나 기능 블럭과 대응관계를 유지하면서 기술하면, 기술 부분의 이해가 쉽기 때문에 유지보수에 월등히 유리하다. 이와 같은 점은 HDL이 일반 언어보다 우수한 점이라고 할 수 있다.

둘째, 하드웨어의 확장을 고려하여 기술한다. 시뮬레이터의 장점을 충분히 살리기 위해서는 쉽게 고칠 수 있게 해야 한다.

끝으로 시뮬레이션의 가장 큰 단점은 실제 시간과의 차이임으로 시뮬레이션의 성능향상을 위한 고려는 항상해야 한다. 위와 같은 점들은 상호 상충적인 관계가 있기 때문에 적절한 타협이 필요하다.

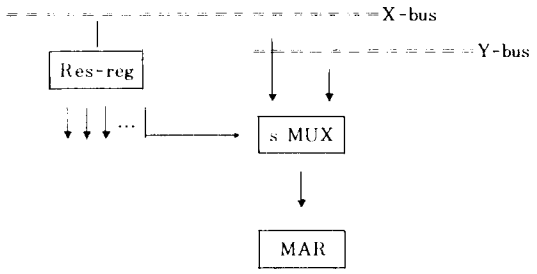
C-MAS의 하드웨어 기술은 마이크로 아키텍춰 수준에서 이루어졌으며, 기술의 형태는 마이크로 아키텍춰의 자원(resource), 기능적 블럭의 표현 그리고 조합 논리 회로로 나누어 생각했다.

• Data Structure of Micro-resources

마이크로 자원은 마이크로 프로그래머에게 보여지는 레지스터들로 버스 Latch, File(scratch pad memory) 등을 말한다. 레지스터들은 크기에 따라 C언어의 변수로 대응되고 메모리는 어레이(array)로 표시가 가능하다. 이때 변수의 type은 레지스터의 크기로 결정되고 scope of name은 global로 하는 것이 사용자 환경과의 인터페이스에서 유리하다.

마이크로 자원중 일반 레지스터와 별도로 고려해야 할 것은 1장에서 언급했던 residual 제어 레지스터의 기술이다. 이 레지스터는 특징은 레지스터의 내용의 저장이나 꺼내는 것은 동일하지만 저장된 값은 특정 필드(field)가 제어에 관여한다는 것이다. 두가지의 표현이 가능한데 첫째는 다른 레지스터와 동일하게 표현하고 제어에 관여할 경우만 필요한 필드를 마스크하는 방법과 둘째, 자원의 데이터 구조의 변환이다. 후자의 경우 프로그램의 이해가 쉬운 장점은 있는데 구현한 컴퓨터의 아키텍춰에 따라 비트의 순서가 뒤바뀔 수 있다.[17] 그림 4의 레지스터는 각 비트가 임의의 제어 신호로 사용하는 경우로 비트 0가 메모리 어드레스 레

지스터의 입력을 선택하는 것을 보여주고 있다. 그림 5는 그림 4를 C언어로 기술한 것으로써 residual-reg는 32비트의 메모리로 표현되며 버스상에서 레지스터를 읽고 쓸 때는 unsigned integer로 사용되고 각 비트가 제어를 담당할 때는 structure로 사용된다. X-bus( ), Y-bus( )는 해당하는 버스에 실린 값을 return하는 function이다.



Res-reg : Residual Control Register
MUX : 2-1 Multiplexer
MAR : Memory Address Register

그림 4. 하드웨어의 예
Fig. 4. Example Hardware.

```
union u_tag {
        struct {
                cntrl_31 : 1;
                cntrl_30 : 1;
                cntrl_29 : 1;
                    .
                    .
                    .
                cntrl_ 1 : 1;
                cntrl_ 0 : 1;
                } bit;
    unsigned int reg;
    } Residual_reg 0;
. . . . . .
Residual_reg 0.reg=x;
  /*Load x to Residual control register*/
. . . . . .
MAR = (Residual_reg 0.bit.cntr 10=0) ? X_bus( ):Y_bus( );
  /*Residual control*/
```

그림 5. 그림 4를 C언어로 기술한 예
Fig. 5. The Description of Fig. 4.

• Data flow

그림 7는 MPU29의 데이터의 흐름도를 나타내고 있다. 입력 버스(Ibus)와 출력 버스(Obus)를 중심으로 중심으로 ALU와 FILE(scrated pad memory)과 각종 레지스터, 그리고 sequencer 부분으로 나뉘어져 있다. 데이터의 흐름은 입력 버스의 한 자원(resource)이 마이크로 필드에 의해 선택되어 버스에 실리며, 데이터는 Rotator(SC field)와 Masker(N & AX field)를 지나 ALU로 제공되고 F field에 의해 연산의 종류가 결정된다. 연산 결과는 출력 버스를 통하여 E field에 선택된 레지스터에 저장됨으로써 한 주기가 이루어진다.

이와 같은 구조의 마이크로 아키텍춰를 I-bus( ), Rotator( ), Masker( ), ALU( ), Load( ) 등 5개의 function으로 분리하여 기술했다. I-bus( )는 마이크로 인스트럭션의 field를 받아서 해당되는 레지스터의 값을 return하는 function으로써 switch 문을 사용한다. Switch문의 형식(syntax)은 ISPS의 decode와 거의 같다. Rotator( ), Masker( )는 I-bus( )로 부터 매개변수를 받아 해당 마이크로 인스트럭션대로 데이터를 조작하여 결과를 return한다. ALU( )는 두개의 매개변수를 받으며, Load( )는 ALU( )의 결과를 레지스터에 저장하는 일을 한다. 이와 같은 일련의 오퍼레이션은 그림 8과 같이 C언어로 기술할 수 있다. Sequencer는 마이크로 프로그램의 제어를 담당한다. 시험코자 하는 마이크로 프로그램은 UNIX의 화일 형태로 시뮬레이터에 입력된다.

• Combinational Logic

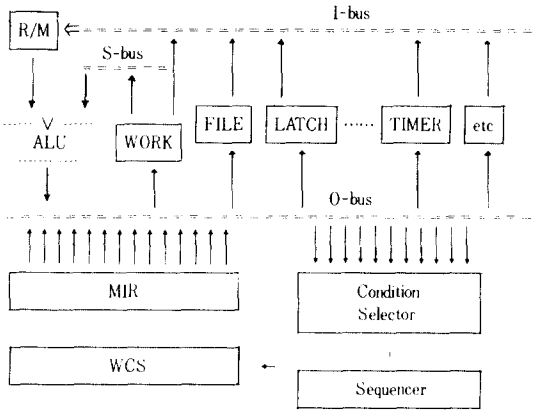
일반 sequential 언어로 기술하기 가장 적합치 않은

FIELD	P 1	BA	S	F	D	BX	DE	E	AX
SIZE	1	1	3	3	3	4	1	4	4

CS	AC	K	N	SC	AE	P 2	Total
8	4	8	8	2	1	1	64bit

P1 P2 : Parity bit
BA : Byte alignment Hardware enable
A : WORKing register address(A-port)
B : WORKing register address(B-port)
S : ALU Source
F : ALU Function Selection
D : ALU Destination & Shift selection
BX : "B" Auxiliary control field
DE : "E" field docode
E : I/O bus selection
AX : Auxiliary control field
CS : Condition Test Selection
AC : Next Address Control
K, N : Constant & Next Address
SC : Shift Control
AE : "A" field decode

그림 6. MPU29의 마이크로 인스트럭션
Fig. 6. The Microinstruction of MPU29.



FILE : 256×32bit Scratch Pad Memory
WORK : 16×32bit Dual Port Memory in 2901
MIR : Micro-Instrction Register
WCS : Writable Control Storage
R/M : Rotator and Masker
LATCH : Bus Latch
I-bus : Input Bus
O-bus : Output Bus
S-bus : Internal Bus of 2901
etc : Bus Address Register
Bus Data Input Register
Bus Data Output Register
Mode Register
Status Register

그림 7. MPU29의 블럭 구조
Fig. 7. The Block Diagram of MPU29.

부분중 하나가 이 부분에 해당된다. 하드웨어는 모든 부품이 동시에 동작하고 있지만 기술에 사용하고 있는 C언어는 이와 같은 특성이 없기 때문에 각 신호의 timing을 고려하여 매 주기의 시작부분과 끝 부분사이에서 그 논리회로의 기능을 표현했다.

V. 사용자 환경(Simulation environment)

보통 시뮬레이너의 사용자 환경의 중요성을 가볍게 여기기 쉽지만 이는 정확한 하드웨어의 기술 못지않게 중요하다.[18] 특히 마이크로 프로그램의 시험도구로 사용되는 것의 경우, 마이크로 프로그램의 디버깅 작업은 마이크로 프로그램 작업보다 지루하고 여러 측면에서 반복적인 시험을 해야하기 때문에 사용자 환경의 비효율성은 하드웨어 기술의 비효율성에서 오는 시뮬레이션 속도의 문제보다 더욱 사용자에게 부담을 줄 수 있다. C-MAS는 사용자 단말기를 통한 대화형식의 환경을 구현하였으며, 이는 사용자에게 여러 마이크로 프로그램 시험작업의 부담감을 덜어줄 뿐만 아니라 시뮬레이터 자체의 하드웨어 기술의 증명 시험에도

```
main(argc, argv)
/*Main Routine of Simulator*/
int argc;
char *argv[ ];
/*a file name containing microprogram*/
{
        .
        .
    Load(ALU(Masker(Rotator(I-bus( ))),WORK));
        .
        .
}
ALU(r, s)
unsigned int r;          /*R operand*/
unsigned int s;          /*S operand*/
{
    . . . . . .
    return(result);
}
I-bus( )
{
    switch(E) {
        case 0:
            return(LATCH);
        case 1:
          .
          .
          .
        case 15:
}
O-bus(result)
unsigned int result;     /*ALU result*/
{
    switch(E) {
        case 0:
            LATCH=result;
            return;
        case 1:
        case 2:
          .
          .
          .
    }
}
```

그림 8. 그림 7을 C언어로 기술한 예
Fig. 8. The Description of Fig. 7.

큰 도움을 주었다.

사용자 환경의 구현은 UNIX의 window 관리 library를 사용하여 구현했으며, 이는 시뮬레이션의 속도와 마이크로 아키텍춰의 변형 등에 따른 사용자 환경부분에 영향을 최대한 줄일 수 있게 하드웨어의 기술 부분과 독립성을 유지하고 있다. 그림 8과 같은 데이터 구조는 하드웨어 기술 부분에서 사용하는 변수와 사용자 환경에서 사용하는 변수를 따로 선언하고 필요에 따라 사용자 환경의 데이터 구조에 하드웨어 기술부분의 변수의 포인터를 연결시킴으로써 시뮬레이션 과정에 시

간적인 부담을 주지 않으면서 각 자원의 내용을 볼 수 있고 변형할 수 있다. 또한 하드웨어의 기술의 변화에 대해 포인터의 연결 상태를 변화시킴으로 가능하기 때문에 새로운 하드웨어를 시뮬레이션할 때 사용자 환경은 수정없이 사용이 가능하다.

C-MAS의 사용자 환경은 다음과 같은 기능을 제공하고 있다.

```
struct  {
        WINDOW *win;      /*Window pointer*/
        int *p;           /*Resource pointer*/
        char X;           /*X position in window*/
        char y;           /*Y position in window*/
        } display-element[MAX-NUM];
          /*Display Element*/
```

그림 9. 사용자 환경 부분의 데이터 구조
Fig. 9. The Data Structure of User Interface.

Single step : 각 마이크로 인스트럭션을 한개씩만 수행하는 기능으로 수행후 각 자원의 상태와 수행한 인스트럭션을 disassemble하여 스크린에 보여줌으로써 각 인스트럭션의 기능의 이해 및 디버깅 작업의 시작 단계나 세밀한 디버깅 작업에 유용하다.

Break point : 마이크로 프로그램 어드레스의 break point를 지정함으로써 프로그램의 제어의 흐름을 따라갈 수 있다. 이때 어드레스에 don't card digit를 사용할 수가 있기 때문에 break point를 넓은 범위에서 사용할 수 있다. (예1 120x;1200번지부터 120f번지까지) 또한, 마이크로 프로그램의 제어가 지정한 break point를 지나가지 않고 무한 loop을 돌 경우 인터럽트를 걸어 수행을 멈출 수 있는 dynamic break point 기능을 제공한다.

Resource update : 수행중 언제든지 각 자원의 상태를 임의로 바꿀 수 있는 기능이다. 전체가 완성되지 않은 상태에서 일부 프로그램 모듈을 시험때, 혹은 한두개의 자원의 상태 때문에 원하는 시험을 할 수 없을 때 사용자가 임의의 상태를 만드는데 사용한다. 원하는 머쉰의 상태를 에디터를 사용하는 것 처럼 만들 수 있으므로(이 기능이 없을 경우는 마이크로 프로그램을 수행시켜서 만들어야함) 시뮬레이션 시간의 절약과 사용자의 부담을 현저히 줄일 수 있다.

State dump : 시뮬레이터의 상태를 일정한 형태로 화일에 저장하는 기능이다. 시험의 종류에 따라 필요한 환경을 저장하여 후에 restore 기능으로 시뮬레이터의 상태를 재현할 수 있고, 혹은 에디터를 이용하여 필요한 상태로 변경이 가능하다. 이 기능은 임의의 상태에서 시험을 반복해야 할 경우나, 전체가 완성되지 않은 프로그램의 시험, 특히 같은 기능을 여러 경우에 따라 반복적으로 시험해야 하는 마이크로 프로그램 디버깅 작업에서 매우 유용하게 이용이 된다.

State restore : 앞에서 언급한 dump 기능과 함께 자주 사용되는 기능으로, 저장된 시뮬레이션 상태를 저장한 화일을 읽어서 그 상태를 재현한다. 이 기능은 resource update 기능과 유사한 것으로 전자는 한두개 자원의 상태를 일시적으로 변화시키는데 유용하며, 본 기능은 반복적으로 사용이 필요한 상태(예, 인스트럭션의 fetch를 시작할 때, 인터럽트가 걸렸을 때 등등)의 재현에 필요한 기능으로 dump 혹은 에디터로 만들어진 머쉰의 상태를 한꺼번에 각 자원에 넣는다.

Trace : 수행한 마이크로 프로그램의 어드레스를 추적하여 저장하는 기능이다. 이 기능은 마이크로 프로그램의 수행시(run time)의 특성의 분석과 통계를 제공함으로써 마이크로 프로그램과 하드웨어의 취약점 발견에 좋은 도구로 사용할 수 있다.

Previous state display : 현재의 상태를 만든 바로 전 머쉰의 상태를 보여 주는 기능으로, 마이크로 프로그램의 제어가 전 상태의 조건으로 결정되기 때문에 사용자는 지난 상태를 항상 재검토케 되어 가장 자주 사용되는 기능이다.

Mode : 시뮬레이션의 모드를 결정하는 기능이다. 이 기능은 시뮬레이션의 용도에 따라 위의 나열한 기능(Trace, Previous state display)들을 on/off하여 시뮬레이션의 수행 속도를 향상시킬 수 있다.

Etc. : 그 밖에 메모리의 읽고 쓰는 기능, 사용법을 가르쳐 주는 help 기능 등이 있다.

Ⅵ. 結 論

C-MAS는 Berkeley UNIX 4.2환경에서 C언어로 개발했으며, 전체 프로그램의 크기는 약 3,500line 정도 된다. 부분별로 보면 사용자 환경이 전체의 반을 차지하며, 하드웨어 기술 부분이 전체의 약 1/3, 나머지가 메모리와 버스 등 시스템 관련 환경이다. 사용자 환경은 대화 형식으로 구현했으며 여러가지 기능을 제공하고 있다. 구현에 소요된 시간은 타겟 머쉰의 분석 과정을 제외하고 순수히 설계와 구현에만 4man-month 정도된다. 시뮬레이션 속도는 초당(VAX 11/750 CPU time) 마이크로 인스트럭션 50개 정도를 수행할 수 있다. 시뮬레이터의 하드웨어 기술의 증명을 위한 시험은, 첫 단계로 임의의 마이크로 인스트럭션의 수행후 각 자원의 변화 상태가 올바른가를 조사하는 과정을 거친 후, 두번째 단계로 현재 F/4000 시스템에서 돌고

있는 마이크로 프로그램을 이용하여 에뮬레이션하고 있는 IBM의 각 인스트럭션이 제대로 수행되는지를 검사함으로써 마쳤다. C-MAS는 마이크로 프로그램의 수행시의 환경 분석에 주로 많이 이용을 하였으며 앞으로 F/4000의 마이크로 프로그램을 변형하여 새로운 머쉰의 에뮬레이션이나 vertical migration 등의 연구에 이용할 예정이다.

參 考 文 献

[1] Siewiorek, Bell, Newell, Computer Structures: Principles and Examples, *McGraw-Hill*, 1982.

[2] John Stockenberg, Andries van Dam, "Verical migration for performance enhancement in layered hardware/firmware/software system", COMPUTER, May 1978.

[3] Tomlinson G. Rauscher and Ashok K. Agrawala, "Dynamic problem-oriented redefinition of computer architecture via microprogramming", *IEEE Trans. Comp.* vol. C-2, no. 11, nov. 1978.

[4] Scott Davidson and Bruce D. Shriver, "An overview of firmware engineering", *COMPUTER*, May 1978.

[5] System 29 Microprocessor Development System User Manual,Advaned Micro Devices, 1978.

[6] Franklin Prosser and David Winkel, "The logic engine development system support for microprogrammed bit-sice development", *Proc. 16th Microprogramming workshop*, Oct. 1983.

[7] F/4000 Student Study Guide, Formation Inc.

[8] Bipolar Microprocessor Logic and Interface, *Advanced Micro Devices*, 1983.

[9] IBM System/370 Principles of Operation, *IBM*, 1981.

[10] Glenford J. Myers, Digital System Design with LSI Bit-Slice Logic, *New York: Wiley*, 1980.

[11] 박병관, 서대화, 박승규, 모듈화 마이크로 프로그램에 관한 연구, 연구보고서, 한국전자통신연구소, Jan. 1986.

[12] Surrendra Dudani, Edward Stabler, "Types of hardware description", *Proc. of 7th International Conf.* on Computer Hardware Description Languages, 1983.

[13] M.R. Barbacci, "Instruction set processor specification (ISPs): The notation and its application," *IEEE Trans. Comp.* vol. C-30, no. 1, Jan. 1981.

[14] 백남석, 마이크로 프로그램 개발 시스템에 관한 연구, 석사학위논문, 한국과학기술원 전산학과, 1985.

[15] Subrata Dasgupta, "Hardware description languages in microprogramming systems", *COMPUTER*, Feb. 1985.

[16] Norihisa Suzuki, "Concurrent prolog as a efficient VLSI design language", *COMPUTER*, Feb. 1985.

[17] Danny Cohen, "On holly wars and a plea for peace", *COMPUTER*, Oct. 1981.

[18] Glenford J. Myers and David G. Hocker, "The use of software simulators in the Testing and Debugging of Microprogramming Logic", *IEEE Trans. Comp.* 1981.

[19] Brain W. Kernighan, Dennis M. Ritchie, The C Programming Language, *Prentice-Hall*, 1978.