실시간 에뮬레이터의 설계 및 제작

Design and Implementation of Real-Time Emulator

* 전 문 수 (Jeon, Moon Su)
** 최 항 식 (Choi, Hang Sik)
*** 박 민 용 (Park, Min Yong)
**** 이 상 배 (Lee, Sang Bae)

요 약

본 논문에서는 기존의 ICE(In-Circuit Emulator) 기능을 갖춘 사용이 간편하고, 쉽게 이동이 가능한 저가격 범용 8비트(bit) 마이크로프로세서의 실시간 에뮬레이터를 설계, 제작하고자 한다.

ICE의 기능을 구현하기 위해서 2개의 보드 즉 에뮬레이션 보드와 콘트롤 보드를 사용하는 구조로 고안하였다. 에뮬레이션 보드에는 CPU 8085(Intel)를 사용하고, 콘트롤 보드에는 표적시스템 (Target System)의 CPU와 같은 CPU를 사용하였다. 이러한 구조는 표적 CPU가 바뀔때 콘트롤 보드만 교환하면 된다는 점에서 실용적이다.

에뮬레이션 보드는 범용 8비트 마이크로프로세서에 대해서, 콘트롤 보드는 표적 CPU가 Z-80인 시스템에 대해서 제작하였다. 또한, 에뮬레이터의 기능에 의해, 표적 CPU 자체의 기능이 상실됨을 회복시켰다.

ABSTRACT

This paper presents a design of low-cost and portable In-Circuit Emulator (ICE).

To implement the functions of ICE dual module technique, emulation module and control module, is used and RS-232C port is added to communicate with a host or terminal station. Emulation module uses a CPU 8085, and control module uses the same CPU as target CPU.

The structure is very practical with respect that only control module is changed according to the target CPU.

In this study emulation board is implemented for all common 8-bit microprocessors, but control board is implemented only for target CPU Z-80.

Efforts has been made to compensate the target function which is degraded due to the emulation.

* 정회원·연세대학교 대학원
** 정회원·연세대학교 대학원
*** 연세대학교 전자공학과 교수
**** 연세대학교 전자공학과 교수

Ⅰ. 서 론

마이크로프로세서가 발달함으로써, 마이크로프로세서 개발연구 못지않게 중요한 것이 개발장비의 연구임을 실감하고 있다. 즉 하드웨어의 테스트와 디버깅 (debugging)의 필요성은 감소되고, 메모리의 가격저하에 따라 소프트웨어의 테스팅, 인테그레이션 (integration), 디버깅의 필요성이 급속히 증가되고 있다.

이에 따라 등장한 장비로 MDS (Microprocessor Development System), ICE (In-Circuit Emulator) 등이 있다.[1] MDS는 가격이 매우 비싸고, 이동이 불편하며, 사용법이 매우 복잡하다.
대부분의 마이크로프로세서의 중요한 개발장비인 Intel사의 ICE는 MDS장비의 일부에 속한다.
일반적인 debugger와는 실제의 운영조건과 동일한 입출력조건하에서 실시간 (real-time)으로 소프트웨어를 테스트할 수 있다는 점에서 다르다.[7,8]
또한 에뮬레이터는 실제 운영조건하에서 표적 시스템에서 동작되는 마이크로프로세서를 모니터 (monitor)하고, 표적 시스템이 없는 경우 소프트웨어를 시뮬레이션 (simulation)하는 장비로서 하드웨어 break등의 기능을 갖고 있다.

본 연구의 주요목적은 RS-232C 소형 컴퓨터를 이용하여[4] 기존의 ICE 기능을 갖춘 MDS 보다 사용이 간편하고, 쉽게 이동이 가능한 저가격의 범용 마이크로프로세서의 에뮬레이션을 실현시켜 다른 8비트 마이크로프로세서에도 응용 가능하며, 필요시 쉽게 확장 가능토록 설계코자 한다. 또한, 이러한 기능을 살리다보면 실제 CPU 자체의 기능이 상실되기 쉬운데, 본 연구에서는 최대로 표적 CPU Z-80A 자체의 기능을 회복한다.

Ⅱ. 시스템 구성

전체 시스템 구성은 그림 1과 같다.

표적 시스템의 프로세서 대신 콘트롤 보드의 40핀 케이블을 연결하며 에뮬레이션 보드와 소형 컴퓨터는 RS-232C 케이블로 연결하며 소형 컴퓨터에서 작성된 프로그램을 download하여 프로그램을 시뮬레이션 하거나 표적시스템의 에뮬레이션을 할 수 있다.
에뮬레이터는 주CPU와 종속 CPU로 구성되며 주CPU가 종속 CPU를 제어하여 표적 시스템의 에뮬레이션이 가능하도록 되어있다.

그림 2는 에뮬레이터의 구성도이다.

1. 에뮬레이션 보드 (Emulation Board)

에뮬레이션 보드는 CPU, PROM, RAM, USA-

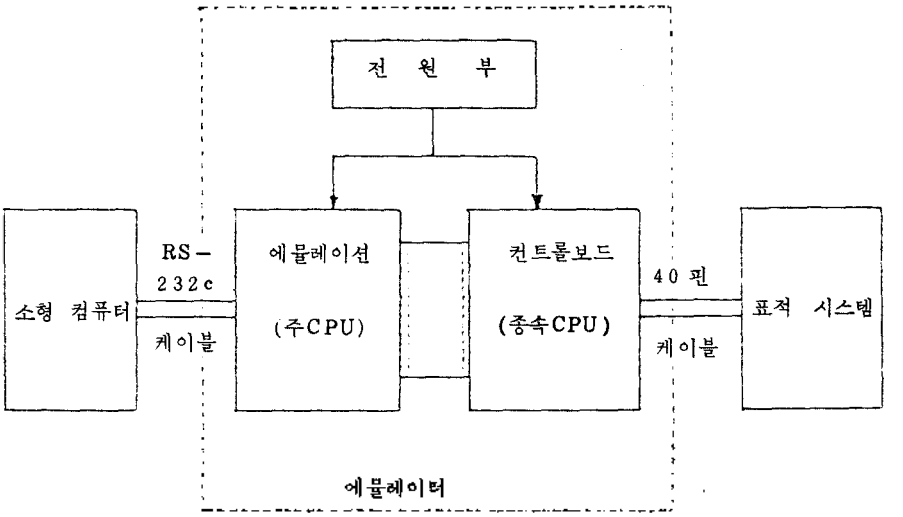

그림 1. 시스템 구조
Fig.1. Structure of the system

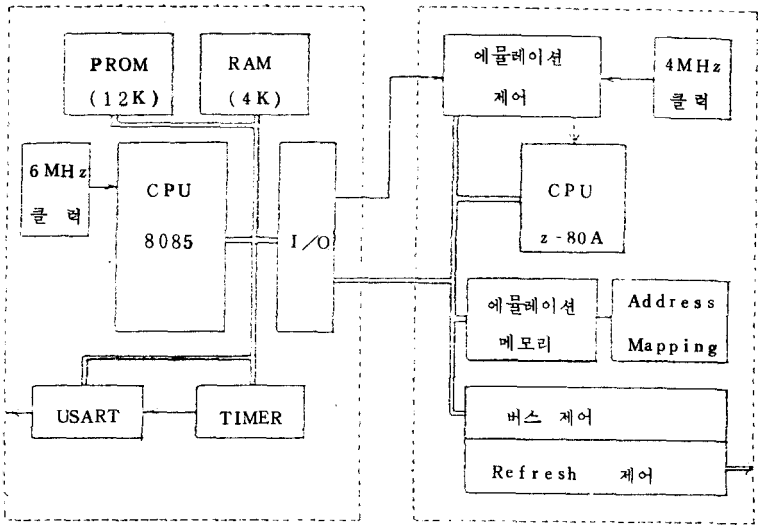

그림 2. 에뮬레이터의 블록 구성도

Fig.2. Block diagram of Emulator

RT(Universal Synchronous/Asynchronous Receiver/Transmmiter), TIMER, I/O로 구성되어 있으며 콘트롤 보드와 PROM의 교환만으로 모든 8비트 마이크로프로세서의 에뮬레이션이 가능하도록 하였다.[9]

주CPU는 6 MHz 클럭을 가진 Intel 8085로 채택하였다. RST 7.5 RST6.5는 USART의 데이터를 신속히 처리한다. INTR, INTA, TRAP은 콘트롤보드와 에뮬레이션 보드간의 케이블 연결상태 및 표적 시스템의 전원 투입 유무를 확인한다.

메모리는 어셈블러 (Assembler) 및 운영 소프트웨어를 내장한 PROM(12K byte) TRACE용 RAM(4K byte), work area용 8155의 RAM 256 byte로 구성된다. TIMER는 USART에 필요한 RxC, TxC를 공급하며 USART는 RS-232C 케이블을 통해 소형 컴퓨터와 연결되어 있다.

I/O는 콘드롤 보드의 종속 CPU의 번지와 데이타 버스의 내용을 읽거나, 데이타 버스에 데이타를 쓰며, 종속 CPU의 콘트롤 단자의 내용을 읽어 마이크로프세서의 status를 감시하거나, 콘트롤 단자를 직접 제어하기 위해 5개의 8비트 포트를 갖는다. 또한, USART의 Baud Rate와 7/8비트, Enable/Disable parity, Even/Odd parity를 선택하기 위해 1개의 6비트 포트를 갖는다. 표-1은 에뮬레이션 보드의 I/O와 그의 향방이다.

2. 콘트롤 보드 (Control Board)

콘트롤 보드는 CPU, 에뮬레이션 메모리, 번지 선택회로, 버스제어, 에뮬레이션제어 및 리후레쉬 (Refresh) 제어회로로 구성된다.

종속 CPU는 Zilog Z-80A이며, 스위치에의해 콘트롤 보드내의 4 MHz나 표적 시스템의 클럭을 선택함으로써 실시간 에뮬레이션을 가능하도록 하였다.

에뮬레이션 메모리는 8 K 바이트의 static RAM으로 구성되며 번지 선택회로에 의해 메모리 공간에 대해 8K 바이트 단위로 어드레싱 (Addressing) 이 가능하다.

3. 하드웨어의 설계

에뮬레이터 종속 CPU의 상태를 변화시키지 않고, 각 번지 및 데이타, 콘트롤 단자의 내용을 읽을 수 있어야 하며 싱글 사이클 스텝 동작을 실시할 수 있어야 한다.

Z-80 에뮬레이터의 경우 $\overline{NMI}$ 단자를 이용하여 에뮬레이션을 한 시스템이 있으나, 표적 시스템으로부터의 inturrupt에 대응하지 못하는 결점이 있어, 본 시스템은 에뮬레이션 제어부에 의한 $\overline{WAIT}$ 입력에 따라 wait 상태가 자유롭게 걸어져 에뮬레이션 보드

표-1. 에뮬레이션 보드의 I/O와 그의 향방
Table-1. I/O on Emulation board and its orientation to Control board

에뮬레이션 보드의 I/O		Z-80 콘트롤 보드의 입출력 대응 및 제어 단자
사용 부품	포트명	
8255	PA	Z-80의 하위 번지 (A0-A7)
	PB	Z-80의 상위 번지 (A8-A15)
	PC	Z-80의 데이타 (D0-D7)
8155	PA0	Z-80의 $\overline{MI}$ 입력
	PA1	〃 $\overline{WAIT}$ 〃
	PA2	〃 $\overline{BUSAK}$ 〃
	PA3	〃 $\overline{WR}$ 〃
	PA4	〃 $\overline{RD}$ 〃
	PA5	〃 $\overline{HALT}$ 〃
	PA6	〃 $\overline{MERQ}$ 〃
	PA7	〃 $\overline{IORQ}$ 〃
	PB0	$\overline{ENIT}$ 제어
	PB1	$\overline{EWI}$ 〃
	PB2	$\overline{EBUSRQ}$ 〃
	PB3	$\overline{EW2}$ 〃
	PB4	$\overline{EW3}$ 〃
	PB5	$\overline{ENMI}$ 〃
	PB6	$\overline{EBUS}$ 〃
	PB7	$\overline{EME}$ 〃

에서 I/O를 통해 Z-80 버스의 내용을 읽을 수 있다. 시스템의 에뮬레이션 제어회로는 그림 3과 같다.

EW_3가 machine cycle에서 wait상태가 계속된다. 이때 주 CPU는 I/O를 통해 버스상의 내용을 읽어들인 후 $\overline{EW_1}$, $\overline{EW_2}$, $\overline{EW_3}$에 의해 $\overline{WAIT}$ = H로 하여, 다음의 machine cycle로 진행시킨다.

$\overline{EW_3}$는 실시간 에뮬레이션을 위한 제어신호로서 $\overline{EW_3}$ = "L" 일때 실시간 에뮬레이션을 실행하며, $\overline{EW_3}$ = "H" 일때 $\overline{EW_2}$에 의해 싱글사이클 스텝 동작을 실현시킬 수 있다. 즉, $\overline{EW_2}$="↑"로 하므로써 Z-80의 wait를 해제하며, 이후 machine cycle의 종료시에는 EW_0 = "L"로 되어 다음 machine cycle의 T_2 사이클 이후 wait 상태가 계속된다. $\overline{EW_1}$은 표적 시스템으로 부터의 $\overline{TWAIT}$상태에 관계없이 $\overline{WAIT}$ = "H"로 하여, 다음의 machine cycle로 진행시킨다.

본 논문에서 첫자가 "E"로 시작하면 에뮬레이션 보드의 신호, 첫자가 "T"로 시작하면 표적 시스템의 신호, 본래 표시대로 있는 신호는 콘트롤보드(Z-80) 자체의 신호이다.

버스 제어 회로는 그림 4와 같으며, $\overline{CSE}$는 그림 5에서와 같이 에뮬레이션 메모리를 8K-byte 단위로 어드레싱 하기 위한 번지 선택 (address mapping) 회로에서 발생한다.

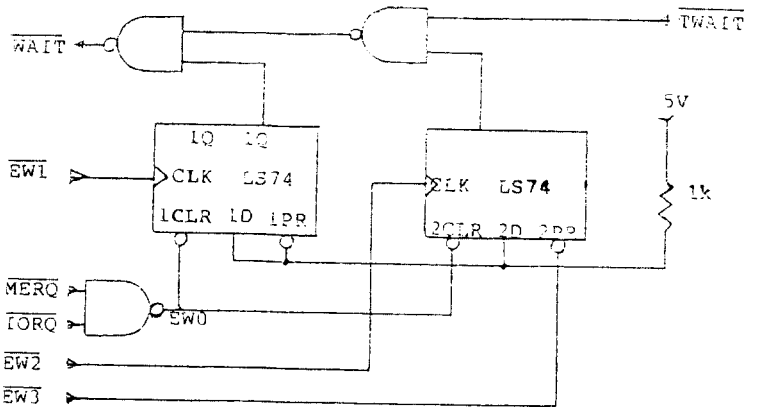

그림 3. 에뮬레이션 제어 회로
Fig.3. Emulation control circuit

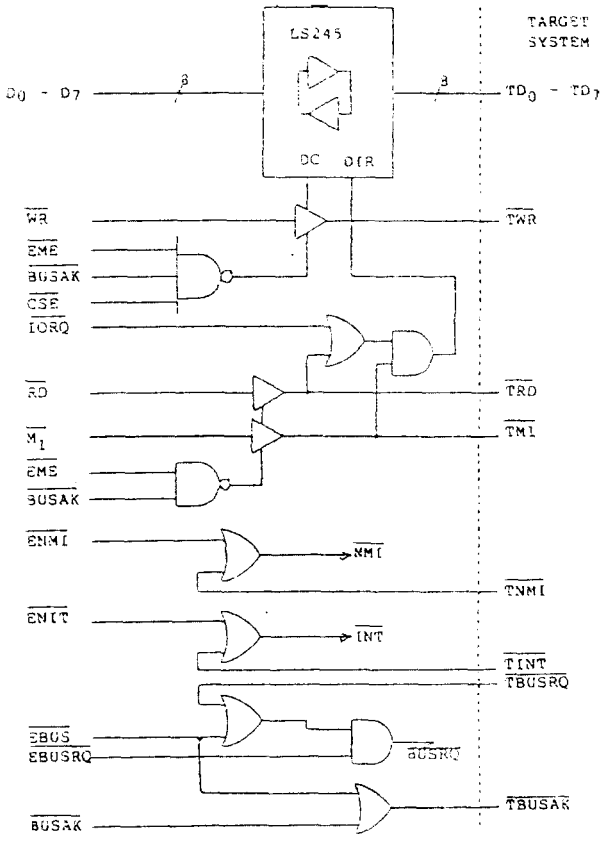

그림 4. 버스 제어 회로

Fig.4. Bus control circuit

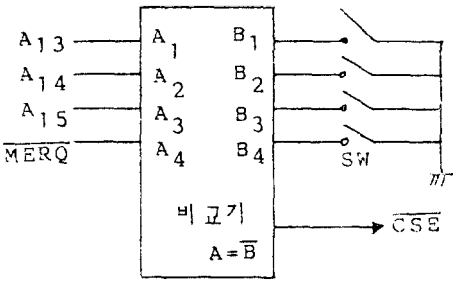

그림 5. 번지 선택회로

Fig.5. Address mapping circuit

표 1 에서와 같이 8155의 포트 PB를 제어하여, 버스 및 콘트롤 신호를 제어하며 EINT, EBUS, ENMI, EBUSQR, BUSAK, NMI, RD, WR 데이타 버스를 제어한다.

본 에뮬레이터에서는 에뮬레이터 자체가 표적 시스템의 DRAM(Dynamic RAM)을 Z-80 CPU의 리후레쉬 (refresh) 기능을 수행하여야 하는데, 주CPU가 Z-80에 wait를 가하여 그 기능을 수행하지 못한다. wait상태 동안, 그 외부 카운터로 부터 리후레쉬 번지를 발생시킨다.[2]

그리고 wait상태에서 벗어난 Z-80가 리후레쉬 기능을 수행하게 될때, 내부 레지스터 R의 내용은 이 외부 카운터의 내용과 일관성이 없게 되므로 외부카운터를 사용하여 계속해서 한번씩 리후레쉬 번지를 발생시키도록 하고 그때의 콘트롤 신호는 Z-80으로부터 직접 나오게 한다. 그림 6은 버스 제어 회로의 일부만 데이타 버스전환 방식으로 개조한 부분을 나타낸 그림이다. 여기서 RE신호 (주CPU가 종속 CPU에 wait를 걸면 active high가 됨)가 발생하면 LS374를 tri-impedence 상태로 만들어 표적 시스템과 차단되고, RE신호가 low이면 두개의 LS374는 양방향 버퍼로 동작한다.

멀티플렉서에 의해 A_0-A_7이 CPU로 부터 나가게 되고, Z-80 CPU가 8085 CPU로 부터의 웨이트를 장시간 받을 경우 (RE가 active), 카운터에 연속적으로 리후레쉬 번지와 함께 TRFSH, TMREQ가 low로 되어 연속해서 순차적으로 표적 시스템의 DRAM을 리후레쉬하고 Z-80 CPU가 정상 동작을 하여 OP code 페치후 MREQ, RFSH 신호를 사용하여 1개의 리후레쉬 번지를 발생시킨다. 이와 같이 실시간 동작에는 영향을 주지않으며 다른 회로 및 프로그램에 독립되어 있다.

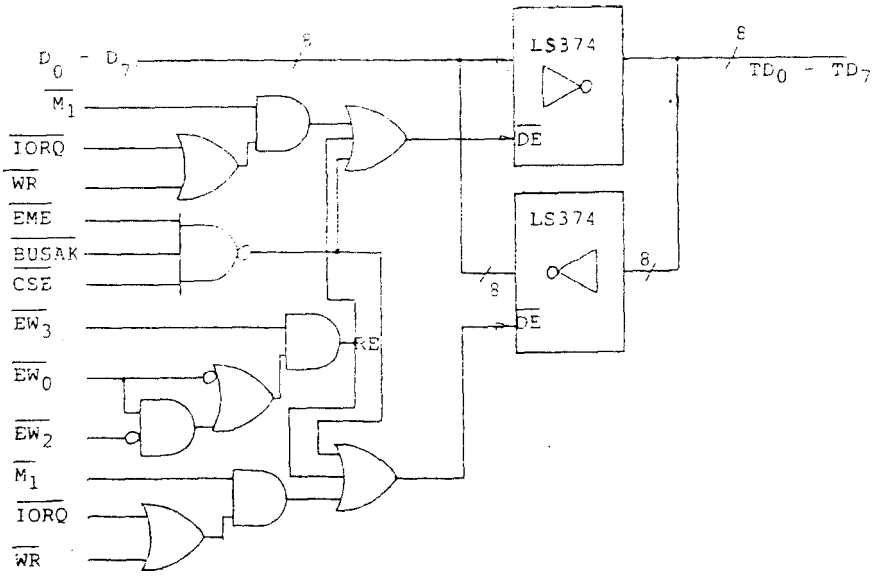

그림 6. 리후레쉬 제어를 위한 버스 전환회로

Fig.6. Bus switchover circuit for refresh control

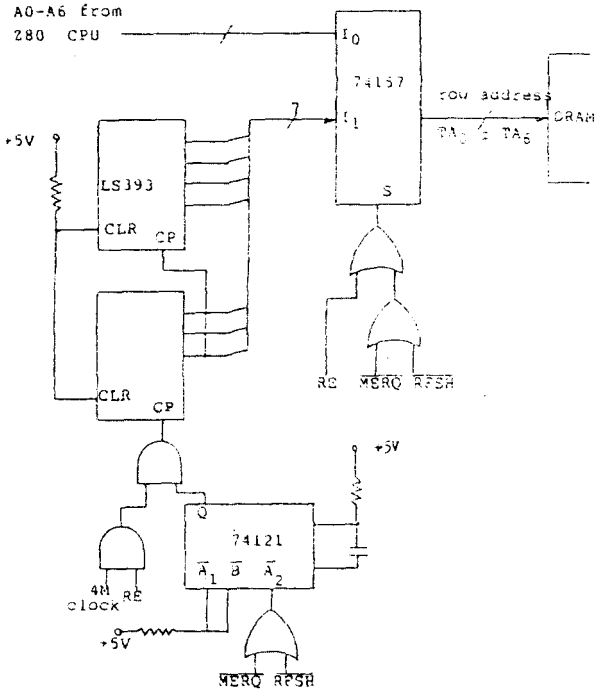

그림 7. 리후레쉬 제어 회로

Fig.7. Refresh control circuit

Ⅱ. 소프트웨어의 설계

소프트웨어의 전체적인 구조는 에뮬레이터를 초기화 시킨 다음 ROM과 RAM을 테스트하며 USART를 초기화 시킨다. 이후 모니터 프로그램에서는 소형컴퓨터로 부터 명령을 받아, 각 명령에 해당하는 일을 처리하도록 되어 있다. 이에 대한 유통도는 그림 8과 같으며 에뮬레이션을 위한 명령은 일반적인 ICE의 범용 명령 형식과 같게 했다.

본 장에서는 앞의 장과 관련하여 명령 삽입 프로그램과 TRACE 명령처리 프로그램에 대해서 설명하고자 한다. 전체 프로그램은 8085 Assembly Language로 작성한다.[5]

명령 삽입 프로그램은 에뮬레이션 보드에서 Z-80에 직접 명령 및 데이타를 부여하며 표적 시스템의 메모리 및 I/O, Z-80 내부의 레지스터의 내용을 1바이트 단위로 읽기 및 변경할 수 있는 프로그램으로 이의 유통도 그림 8과 같다.

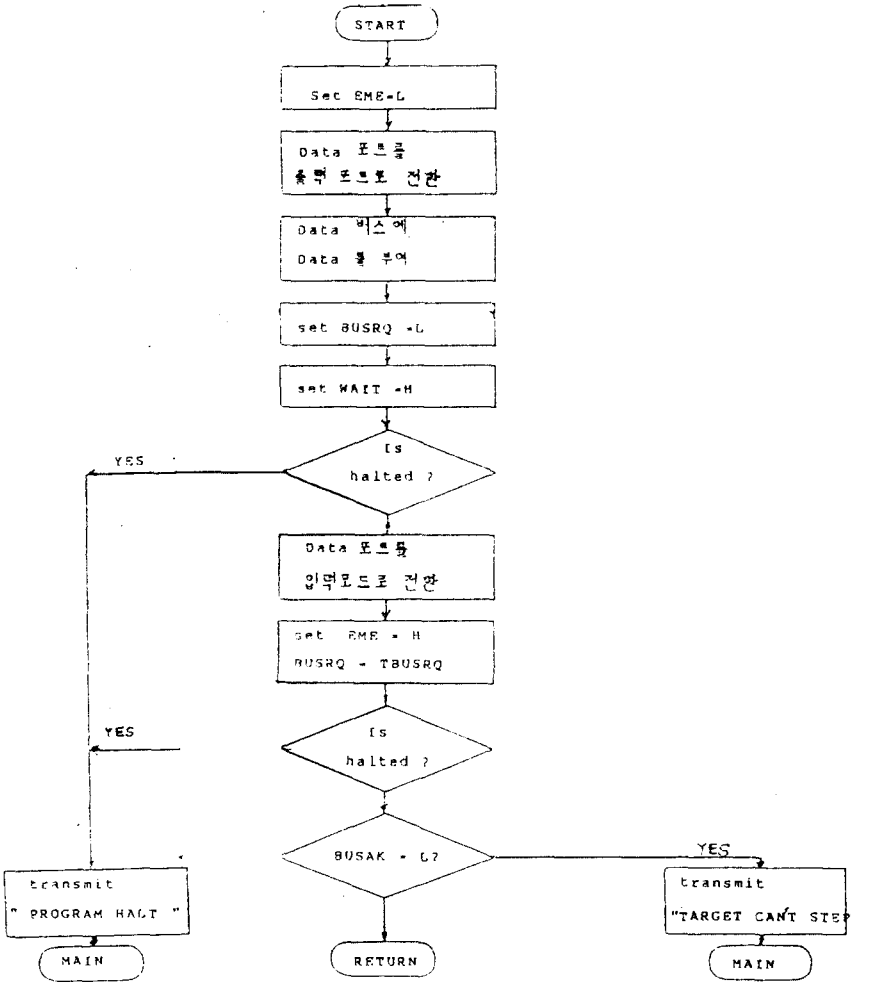

그림 8. 명령 삽입 프로그램의 유통도

Fig.8. Flow chart of instruction insertion program

Z-80의 WAIT 입력에 의해 싱글 사이클 스텝 동작을 시키면서 명령 페치 사이클이 되면 필요한 명령 및 데이타를 순차적으로 부여한다. 이때 CB_H, DD_H, ED_H, FD_H로 시작되는 명령의 경우, 페치할때 M_1이 2개 발생되므로 주의하여야 한다. 표-2에서와 같은 명령에 의해 표적 시스템의 메모리, I/O와 Z-80 내부의 레지스터들을 읽거나 변경시킬 수 있다.

표-2. 메모리, I/O 레지스터의 Read,write 명령
Table-2. Read/write instruction of memory and I/O register

기 능		명령 및 내용
Memory	read	LD HL, nn (번지 지정) LD A, (HL)〔read〕
	write	LD HL, nn〔번지 지정〕 LD (HL), A〔write〕
I/O	read	IN A, (n)〔read〕
	write	OUT(n),A〔write〕
Register (A-L)	read	PUSH qq〔read〕 pop qq
	write	PUSH qq pop qq〔write〕
Register (Ix,Iy)	read	PUSH IX〔read〕 pop IX
	write	PUSH IX pop IX〔write〕
Stack pointer	read	PUSH AF pop AF〔번지read〕
	write	LD HL, nn〔번지 지정〕 LD SP, HL〔write〕
P.C	read	〔현재 번지 read〕
	write	JP nn〔write〕

TRACE는 싱글 사이클 동작 하에서 그때의 번지 및 데이타와 Z-80의 status를 기록하여, 소프트웨어의 디버깅이 용이하도록 한다. 소형컴퓨터,또는 터미날로 부터 시작 번지와 종료 번지, 트리거 어드레스를 받은 후 계속적인 싱글 사이클 스텝 동작에 의해 트리거 어드레스와 Z-80 어드레스가 일치하면 Z-80의 어드레스, 데이타, 스테이터스를 RAM에 저장하기 시작하여 256사이클이 되거나 종료 어드레스와 일치하면 TRACE는 끝나게 된다. 이에 대한 유통도는 그림 9와 같다.

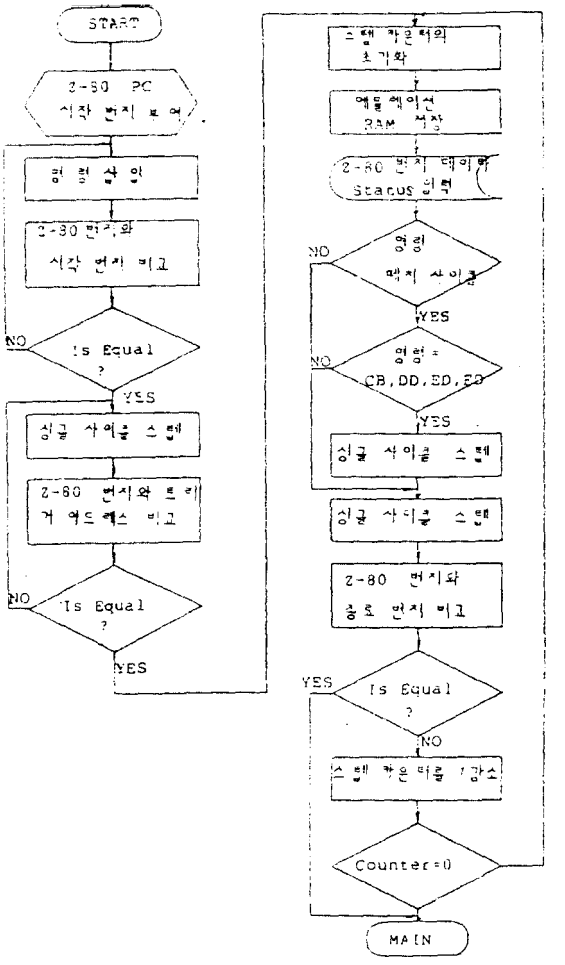

그림 9. TRACE 명령의 유통도
Fig.9. Flowchart of TRACE command

Ⅳ. 실험 결과 및 검토

에뮬레이터의 소프트웨어 개발을 위해 EPROM 에뮬레이터를 사용하였으며[3] TRS-80 MODⅢ를 사용하여 프로그램을 작성하였다. 에뮬레이터를 실험하기 위한 시스템 구성은 그림 1과 같이 소형 컴퓨터로서 터미날(GDA-7100A), TRS-80, MODⅢ, AppleⅡ를 번갈아 사용하였으며 표적 시스템은 1K 바이트의 모니터 프로그램을 가진 SRC-80A Z-80 Start kit 싱글 보드 마이크로 프로세서를 사용하였으나, 실험

은 SRC-80A의 모니터 프로그램을 분석하는 것으로 대신하였다. 우선 TRS-80 MODⅢ의 Disassembler와 에뮬레이터의 Disaassembler에 의해 SRC-80A의 모니터 프로그램을 리스트한 결과 에뮬레이터에 의한 리스트의 경우 부록1에서와 같이 서브루틴의 라벨과 보통 라벨을 구분하여 리스트되게 하였으므로 프로그램 분석이 보다 용이하도록 하였다. 그리고 F(Foreward Trace) 명령을 한뒤 L(List Trace) 명령에 의해 나타난 결과와 실제의 SRC-80A의 모니터 프로그램을 비교하였다. Trace 한 결과는 부록2와 같고 그것의 일부만 나타냈으며, IF-ADDR은 명령 페치 사이클일때의 번지를 뜻한다. 부록1에서 Line $\phi\phi\phi4$의 JR $\phi\phi3B$의 명령 실행에 대한 Trace 결과를 부록2에서 살펴보면 $\phi\phi\phi6$번지의 명령과 데이타를 받은 후 다음의 명령은 $\phi\phi3B$에서 페치하고 있음을 알 수 있다. 또한 $\phi\phi3B$번지 명령 실행에 의해 I/O $\phi3$번지로 데이타 CFH를 내보내고 있음을 알 수 있다. 이것은 부록1에서 Line $\phi\phi\phi3$에 의해 Accumulator에 부여된 데이타가 I/O $\phi3$번지로 출력됨을 알 수 있다. 부록2에서 Trace 결과 모든 JUMP 및 CALL 명령에 따라 주어진 번지로 jump하는 것을 알 수 있으므로 Trace가 정상적으로 수행되었다고 결론 지을수 있다.

다음은 TRS-80 MODⅢ, AppleⅡ, 터미날(GDT-7100A)를 소형 컴퓨터로 사용할 때를 비교하였다. TRS-80 MODⅢ는 CPU가 Z-80A이므로 내장된 소프트웨어에 의해 파일을 작성한 뒤 upload/download 명령에 의해 소프트웨어 테스팅이 가능하며 작성된 파일의 저장이 간편하였다. AppleⅡ의 경우는 CPU가 6502이므로 키보드에 의해 입력된 데이타를 download하여 파일저장 하기가 불편하였으며, 에뮬레이터를 위한 운영 프로그램이 복잡하였다.

그러나 CPU가 6502라는 것을 생각하면 매우 유용하게 사용할 수 있다. 또한 터미날(GDT-7100A)를 사용할 때는 소프트웨어를 갖추고 있지 않기 때문에 많은 양의 프로그램은 불편하나 적은 양의 테스팅은 충분하였다. 이상과 같이 에뮬레이터를 실험하여 만족할만한 결과를 얻었으나 에뮬레이션 하는데 WAIT 신호를 이용함으로써 리후레쉬 기능이 상실되어 버스 제어부를 데이타 버스 전환방식으로 개조하고 리후레쉬 제어부를 삽입하여 리후레쉬 기능을 회복시켰다. 즉 DRAM을 read/write 경우를 제외하고는 멀티플렉서의 출력인 DRAM의 열 어드레스(column address)의 단자를 체크해본 결과 열어드레스가 계속하여 순차적으로 출력되었다.

Ⅴ. 결 론

본 연구에서 설계, 제작한 에뮬레이터를 동작시키기 위해 사용되는 소형 컴퓨터와 디스크 드라이버, 프린터, EPROM writer를 갖추면 충분히 개발 장비로 사용 가능하다.[6] 본 시스템의 실험을 위해 소형 컴퓨터는 TRS-80 MODⅢ, AppleⅡ, 터미날(GDT-7100)을 사용하였으며, 표적 시스템인 SRC-80A의 모니터 프로그램을 분석하였다.

그의 Disassembler와 트레이스의 결과는 완전히 일치하였고, 본 시스템 내의 소프트웨어 모니터 프로그램도 만족스럽게 동작을 했으나 Editor Assembler를 이용해서 파일을 작성하고자 할때, UP/DOWN LOAD에 의한 에뮬레이터 내의 Edidor Assembler보다는 소형 컴퓨터 내의 Edidor Assembler가 보다 편리하였다.

본 시스템은 Z-80의 WAIT 단자를 사용하여 에뮬레이션을 실시하였다. 이에 따라 Z-80의 주요기능인 Dynamic RAM의 리후레쉬 기능이 상실됨을 리후레쉬 제어부와 버스 전환방식을 사용하여 리후레쉬 기능을 회복시켰다. MDS의 ICE는 각종 마이크로프로세서에 대해 에뮬레이션이 가능하나, 본 시스템의 콘트롤 보드는 교체되지 않고서 기존의 ICE에 비해 아주 저가격으로 제작이 가능하며 RS-232C를 통해 소형 컴퓨터 및 터미날에 사용 가능하고 소형이기 때문에 휴대가 간편한 잇점이 있다.

에뮬레이션 보드의 ROM과 콘트롤 보드를 교체함에 의해 모든 8비트 마이크로프로세서의 에뮬레이터 제작이 가능하나 16비트에 대해서는 연구가 계속되어야 할 것이다.

이 논문은 1984년도 학술진흥재단의 학술연구조성비에 의하여 이루어 졌으며 후원에 감사한다.

부록 1. Disassemble된 SRC-80A의 모니터 프로그램의 일부

LINE	LABEL	SOURCE CODE	LOC	OBJ
0001		LD A, 07	0000	3E07
0002		OUT (03), A	0002	D303
0003		LD A,CF	0004	3ECF
0004		JR 003B	0006	1833
0005		NOP	0008	00
0051		NOP	0036	00
0052		RET	0037	C9
0053		JP 83E5	0038	C3E583
0054	B003B	OUT (03),A	003B	D303
0055		LD A,BO	003D	3EB0
0056		OUT (03),A	003F	D303
0057		LD HL,83D1	0041	21D183
0058		LD (83F1),HL	0044	22F183
0059		LD SP,83C8	0047	31C883
0060		LD HL,83AO	004A	21A083
0061		LD (83FE),HL	004D	22FE83
0062		CALL 012B	0050	CD2B01
0063		IN A,(01)	0053	CB01
0064		AND 10	0055	E610
0065		JP Z,0353	0057	CA5303
0066		LD A,09	005A	3E09
0067		LD (83F3),A	005C	32F383
0068	B005F	CALL 0081	005F	CD8100
0069		JR 0068	0062	1804
0070		NOP	0064	00
0071		NOP	0065	00
0072		JR 00CB	0066	1863
0073	B0068	CALL 0136	0068	CD3601
0074		LD HL,83F6	006B	21F683
0075		JR NZ,0073	006E	2003
0076		LD (HL),A	0070	77
0077		JR 005F	0071	18EC
0078	B0073	LD A,(HL)	0073	7E
0079		OR A	0074	B7
0080		JR NZ,005F	0075	20E8
0081		INC A	0077	3C
0082		LD (HL),A	0078	77
0083		CALL CALL 00C2	0079	CDC200
0084		CALL CALL 0150	0070	CD5001

부록 2. SRC-80A의 Trace 결과

IFADDR	ADDRESS	DATA	STATUS	IFADDR	ADDRESS	DATA	STATUS
0000	0000	3E	S		0052	01	R
	0001	07	R		83C7	00	W
0002	0002	D3	S		83C6	53	W
	0003	03	R	012B	012B	AF	S
	0703	07	O	012C	012C	32	S
0004	0004	32	S		012D	FD	R
	0005	CF	R		012E	83	R
0006	0006	18	S		83FD	00	W
	0007	33	R	012F	012F	32	S
003B	003B	D3	S		0130	EC	R
	003C	03	R		0131	83	R
	CF03	CF	O		83EC	00	W
003D	003D	3E	S	0132	0132	32	S
	003E	BO	R		0133	ED	R
003F	003F	D3	S		0134	83	R
	0040	03	R		83ED	00	W
	B003	B0	O	0135	0135	C9	S
0041	0041	Z1	S		83C6	53	R
	0042	D1	R		83C7	00	R
	0043	83	R	0053	0053	DB	S
0044	0044	22	S		0054	01	R
	0045	F1	R		0001	BO	I
	0046	83	R	0055	0055	E6	S
	83F1	D1	W		0056	10	R
	83F2	83	W	0057	0057	CA	S
0047	0047	31	S		0058	53	R
	0048	C8	R		0059	03	R
	0049	83	R	0058	005A	3E	S
004A	004A	21	S		005B	09	R
	004B	A0	R	005C	005C	32	S
	004C	83	R		005D	F3	R
004D	004D	22	S		005E	83	R
	004E	FE	R		83F3	09	W
	004F	83	R	005F	005F	CD	S
	83FE	A0	W		0060	81	R
	83FF	83	W		0061	00	R
0050	0050	CD	S		83C7	00	W
	0051	2B	R		83C6	62	W

참 고 문 헌

1. R. Zaks & A. Lesea "Microprocessor interfacing techniques" pp. 399-435, SYBEX, 1979.
2. R. Zaks & A. Leasa "Microprocessor interfacing techniques" pp. 50-53 pp. 399-435, SYBEX, 1979.
3. Eric C. Rehnke "Build on EPROM Emulatro" pp. 194-203. BYTE 1982. 2.
4. Ric Hammond "ICE turns personal computer into development tool" pp. 141-147. Electronic Design 1982. 3.
5. Lance A. Levental "8080A/8085 Assembly Language Programming" Osborne & Associates.
6. John D. Ferguson "In-Circuit Emulator for the Apple II Computer" pp. 419-444, BYTE, 1938, 9.
7. E.G. Mallach "Emulation Practice and Principles" Honeywell Information Systems. Oct. 1972.
8. J.H. Aylor, Edward A. Parrish "A Modular Microcomputer Development System" IEEE Trans. Ind. Electron. Contr, Instrum. Vol. IECI-26 No. 1 FEB 1979.
9. A. Osborne & J. Kane "An Introduction to Microcomputers" Vol. 2, Vol, 3. Selected, pp. 4.65-4.85, pp. 4.113-4.122, pp. 5.4-5.52, pp. 7.1-7.47, Adam Osborne & Associates, 1977.