論文95-32B-1-3

마이크로프로그래밍을 위한 구조적 마이크로어셈블러 설계

(A design of structured microassembler for microprogramming)

申 鳳 熙*, 金 成 鍾**, 李 俊 模****, 申 仁 澈***

(Bong Hi Shin, Seong Jong Kim, Joon Mo Lee, and In Chul Shin)

요 약

본 논문에서는 시스템 설계 변경과 광범위한 마이크로아키텍처에 대응하여 사용자가 쉽게 마이크로명령어 집합과 형식을 정의할 수 있는 머신독립적이며 구조적 마이크로어셈블러를 설계하였다. 하드웨어의 설계와 함께 수행되는 마이크로프로그래밍 과정에서 마이크로프로그래머는 명령어 집합과 실행코드 형식을 손쉽게 변경 또는 개선할 필요를 갖게된다. 그러나 이러한 환경구축에는 대형이며 고가의 소프트웨어가 요구된다. 본 논문에서는 사용자가 직접 마이크로명령어 집합과 실행 코드인 마이크로명령어 형식을 정의 하도록 하였고 소프트웨어의 각부분을 모듈별로 구성하여 쉽게 기능의 첨삭과 활용이 가능하도록 하였다. 심볼테이블 구성을 어셈블러로부터 분리하여 마이크로명령어 집합과 의사명령어를 심볼테이블에 삽입하도록하고 마이크로명령어 정의어를 설계하여 마이크로명령어 형식을 자유롭게 정의하도록 하였다. 이들 모두 IBM-PC상에서 C언어,FLEX와 BISON으로 구성하였다.

Abstract

In this paper, a independent and structured microassembler was designed for easily changing the system design, and for designing various microarchitecture. When the designer's hardware and microprogramming process were made concurrently, it is needed to easily change or improve the instruction set and executable code format. But this type of developed environment requires a high cost and a large software system. A proposed microassembler was designed so the designer directly defines the microinstruction set and format to be executed. And we implemented a module from each part of the software, so it is now possible to use practically and upgrade the function of each part. First, the symbol table was separated from the assembler. And then microinstruction was copied into it. The microinstruction format was designed using the defined language that was designed for free microinstruction. This was implemented in an IBM-PC by using the C-language, FLEX, and BISON.

* 正會員, 市立仁川專門大學 電子計算科
(Dept. of Computer Science, Junior College of Incheon)
** 正會員, 檀國大學校 電子工學科
(Dept. of Electronic Eng.,Dankook University)
**** 正會員, 關東大學校 電子工學科
(Dept. of Electronic Eng.,Kwandong University)
接受日字 : 1994年 8月 26日

Ⅰ. 서 론

마이크로프로그램 제어방식은 시스템의 제어점에서 요구되는 제어신호를 WCS(writable control store)에서 제공하는 방식으로 프로그램을 이용한 펌웨어(firmware)적인 제어방식이다. 이 방식은 제어부의 규칙적인 하드웨어 구조로 인하여 순차제어(sequential)에 의한 하드와이어드(hardwired) 제어 방식 보다 수정및 확장이 용이하면서 저가격으로 다기능을 수행 할 수 있다는 장점이 있다. 그러나 마이크로프로그래밍의 생산성을 높이고 마이크로프로그램의 개발시간과 비용을 단축하기 위한 환경을 필요로하게 되며 마이크로프로그래밍을 위한 환경 구성은 어셈블러와 시뮬레이션들로 이루어지며 이들에 대한 연구가 활발히 진행되고 있다.[1-11]

대부분 어셈블러들은 특정 마이크로프로세서 또는 컴퓨터에대한 실행 코드를 생성하도록 설계되어 있어 명령어 집합과 실행코드는 사용자에 의해 변경 될 수 없고 기호언어로써의 난해한 문제점등을 갖고 있다. 따라서 마이크로어셈블러에서 사용자가 마이크로명령어 집합과 마이크로명령어 형식을 정의하도록 하는 기능을 갖추면 이들 단점을 제거할 수 있어 바람직하다.

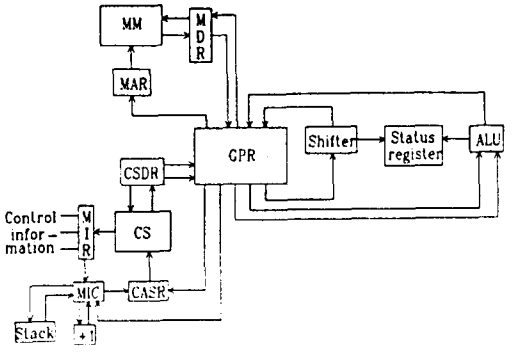


그림 1. 마이크로프로그래블 머신 기본 구조
Fig. 1. Organization of a simple microprogrammable machine.

본 논문에서는 마이크로프로그래머가 직접 마이크로명령어 집합과 실행 코드인 마이크로명령어 형식을 표준화하고 절차형식의 언어로 정의하도록 하였다. 즉 시스템 설계 변경과 광범위한 마이크로아키텍처에 대응하는 머신독립적이며, 전체 프로그램을 심볼테이블, 어휘분석, 파서(parser)와 실행 코드 형식 정의 구성등의 각 모듈별로 분리 설계한 구조적 마이크로어셈블러를 구성하였다.

설계한 구조적 마이크로어셈블러는 어셈블리 언어의 문서 구성 요소인 심볼들을 정적심볼과 동적심볼로 분류하여 정적심볼인 키워드(keyword)들을 마이크로어셈블리 언어 사용자가 마이크로명령어 집합과 의사명령어들로 정의한후 심볼테이블 초기화에서 심볼테이블을 구성하고 이들을 복사되도록 하여 마이크로명령어 집합을 사용자가 정의하도록 하였다. 또한 어휘분석 단계와 파서를 구성하는 구문및 의미 분석단계에서 동적심볼인 레이블(Label)과 수치데이타들이 검출될 때 마다 심볼테이블에 복사하도록 하였고, 델리미터(delimiter)와 연산자(arithmetic)들을 델리미터로 분류하여 파서단계에서 처리하도록 하였다. 어휘분석기는 원시프로그램으로 부터 심볼과 델리미터를 한쌍으로 검출하면 파서는 중간코드를 생성한다.

또한 마이크로명령어 형식 정의어를[12] 이용하여 마이크로명령어를 제한없이 정의하도록 하여 중간코드를 원하는 마이크로명령어 형식에 의존하여 마이크로명령 실행코드를 생성하도록 하였다. 이들 모두 IBM-PC상에서 C언어,FLEX와 BISON으로 구성하였다.

Ⅱ. 마이크로프로그래밍

마이크로프로그래머블 머신은 마이크로프로그래밍에 의한 제어부의 구성을 하드웨어 자원과 기본연산을 근거로 시스템 설계자가 직접 마이크로프로그래밍을 한다. 하드웨어와 관련된 일련의 기본연산들이 적절히 조합되어 복합연산들이 정의되며, 복합연산을 구성하는 각각의 기본연산들을 마이크로명령(microoperation)이라하고,복합연산을 마이크로프로그램 또는 마이크로루틴이라한다. 마이크로프로그램은 제어메모리(control memory)에 저장된다.

1. SMM과 마이크로프로그램

기본적인 SMM(simple microprogrammable machine)은 그림 1과 같다.[13] 제어메모리내의 각 마이크로명령은 MIR을 통하여 시스템 전체의 제어점에 제어신호를 공급하며 CSDR과 CADR을 통하여 CPU의 레지스터 화일과 데이타를 직접 주고 받을수도 있다. 마이크로명령은 자신의 하드웨어 구성에 근거하여 분류정의된다.

SMM의 하드웨어 자원과 데이타에 대해 동작하는 마이크로명령어들은 일반 컴퓨터의 기계어 명령에 해당하는 모든 마이크로루틴을 작성할 수 있도록 타당성이 있어야 하며 체계적으로 표 1.과 같이 분류되고 명칭이 부여된다.[12] 마이크로명령의 명칭들은 집합으로 심볼테이블에 삽입된다.

표 1. 마이크로오퍼레이션
Table 1. SMM microoperation and mnemonic.

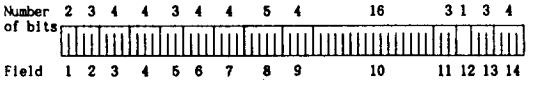
Operation	Microoperation	Mnemonic
Simple Register Transfer	MAR <- GPR(General Purpose Register)	MOVE_GPR_MAR
	MDR <- GPR	MOVE_GPR_MDR
	GPR <- MDR	MOVE_MDR_GPR
	CSDR <- GPR	MOVE_GPR_CSDR
	GPR <- GPR	MOVE_GPR_GPR
	GPR <- CSDR	MOVE_CSDR_GPR
Transformation	GPR <- shifted GPR	SHIFT_RIGHT_GPR
		SHIFT_LEFT_GPR
		SHIFT_RIGHT_LITER
		SHIFT_LEFT_LITER
	GPR <- GPR binary operation GPR (+,-,+with carry)	ADD
		SUB
		ADD_CARRY
	GPR <- unary operation GPR	NOT_GPR
		SET_GPR
		INCREMENT_GPR
		DECREMENT_GPR
		AND
		OR
Sequenceing	MIC <- GPR	MOVE_GPR_MIC
	MIC <- MIR (or part of the MIR)	MOVE_MIRPART_MIC
	PUSH stack to MIC	PUSH
	POP stack to MIC	POP
Memory	READ main memory	READ
	WRITE main memory	WRITE
	CSAR <- GPR and READ control store into CSDR	READ_CSDR
	CSAR <- GPR and WRITE from CSDR into control store	WRITE_CSDR
Branch	Uncondition	GOTO
	Condition	IF_ZERO_GO
		IF_NOT_ZERO_GO
		IF_LESS_ZERO_GO
		IF_CARRY_GO
		IF_SHIFTOUT_ZERO_GO

정의된 마이크로명령의 명칭들을 사용하여 마이크로프로그램을 작성한 후 마이크로프로그램을 마이크로명령어 형식에 의존하여 마이크로프로그래머블 머신의 제어점에서 요구하는 제어신호로 변환하여 출력하게 된다.

2. 마이크로명령어 형식

마이크로명령은 시스템 하드웨어 자원을 제어하기위한 명령이다. 마이크로명령어 형식을 정의하기 위하여 하드웨어 자원의 제어에 관련한 마이크로명령어내의 존속할 필드를 추출하고, 각 필드들을 배치한다. 추출된

마이크로명령어 필드의 배치방식에 따라서 수직(vertical)과 수평(horizontal) 마이크로명령어로 나뉘어 구성된다. 수직 마이크로명령어는 단일 마이크로오퍼레이션을 수행하도록 연산자 필드를 제외한 나머지 피연산자 부분을 오퍼랜드 필드, 리터럴 필드, 작업 처리 순서 필드 및 분기 필드들로 구분없이 정의 구성한다. 수평 마이크로명령어는 동시에 여러 마이크로오퍼레이션을 수행하도록 필드별 용도와 위치를 구별시켜 구성한다. SMM의 수평 마이크로명령어 형식을 그림 2와 같이 구성하였다.[12]



Field code points

1 Memory operation
2 Register transfer
3 GPR1
4 GPR2
5 Tranformation operator GPR3 <- GPR3 operator GPR4
6 GPR3
7 GPR4
8 Shift operator
Bit 1-2 shift operation
Bit 3 Shift direction
Bit 4 Shift type
Bit 5 Shift out
9 GPR5 or shift literal
10 literal
11 Condition
12 True (1) or false (0)
13 Sequenceing operation
14 GPR5

그림 2. SMM의 수평 마이크로명령어 형식
Fig. 2. A horizontal microinstruction format for the SMM.

Ⅲ. 마이크로어셈블러의 설계

구조적 마이크로어셈블러는 사용자가 정의한 마이크로명령어 집합을 이용해서 마이크로프로그램을 작성하고, 마이크로프로그램을 마이크로명령어 형식에 의존하여 마이크로프로그래머블 머신의 제어점에서 요구하는 제어신호로 변환하여 출력하는 마이크로프로그래밍을 위하여 설계한 언어이다.

사용자가 직접 마이크로명령어 집합을 정의하므로써 마이크로프로그램의 작성 및 내용파악이 용이하며, 마이크로프로그램 자체가 문서화 기능을 갖춘 마이크로어셈블러 언어를 정의한다. 또한 마이크로명령어 형식을 표준화하고 절차형식의 언어로 정의하여 광범위한 마이크로아키텍쳐에 대응되는 머신독립적으로 구성한다. 이들 소프트웨어는 각 부분별로 구성하여 각 단계별 수정 및 사용이 용이하도록 하였으며 마이크로프로그램의 정확성과 생산성을 향상시키고, 기호언어로서의 난해함등을 제거한 마이크로프로그래밍을 위한 효과적인 도구를 구성한다.

설계 방법으로서, 마이크로어셈블러의 어휘 및 파서 과정에서 참조되는 심볼테이블에 대한 기능을 초기화와 검색기능으로 분리 구성하였다.

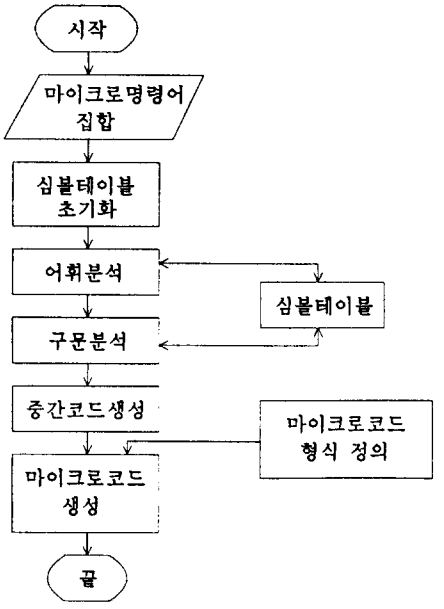


그림 3. 마이크로어셈블러
Fig. 3. Microassembler.

심볼테이블 초기화 과정을 어휘분석 전단계에서 이루어지도록 하여 마이크로프로그래머가 정의한 마이크로명령어 집합과 의사명령어의 정보들을 심볼테이블에 삽입하도록 하였다. 검색기능을 어휘분석 단계에 포함시켜 레이블이나 수치데이타에 해당하는 새로운 심볼들을 처리하도록 하였다. 어휘분석과 파서 과정에서 생성된 중간코드를 정의한 마이크로명령어 형식에 의존하여 마이크로명령 실행코드를 생성시킨다. 구조적 마이크로어셈블러의 개략도는 그림 3과 같다.

1. 문서 구성

구조적 마이크로어셈블리 언어의 문서 구성 형식은 고급 프로그램 언어와 동일하게 자유 형식(free format)을 사용한다. 문서 구성 요소는 사용자가 직접 정의하는 마이크로명령어 명칭과 의사명령어 및 주석기능을 포함한 델리미터로 구성하였으므로 마이크로프로그램 자체로써의 문서화 기능을 높였다. 심볼은 마이크로명령어, 의사명령어와 레이블 및 수치데이타로 분류되며 마이크로명령어는 SMM의 예로써 표 1과 같이 정의하였다. 의사명령어는 일반 어셈블리 언어에서 공통적으로 사용되는 것들을 선택하여 사용하므로 일반성을 높였다. 이는 표 2와 같다.

표 2. 의사명령어
Table 2. Pseudo nstruction.

예약어	용도 및 의미
eject	리스트 페이지
org	프로그램 시작 번지 지정
equ	상수의 정의
set	심볼 상수 정의
dw	데이타 길이 정의
end	프로그램 종료

델리미터는 심볼과 심볼들을 구별해내는 의미 기호로써 기능과 의미를 표 3에 도시하였다.

표 3. 데리미터와 오퍼레이터
Table 3. Delimiter and Operator.

기 호	용 도
:	레이블 지정
;	문장 마지막 표시
,	항목 분류
-	뺄셈
+	덧셈

또한 연산 기호를 사용하므로 마이크로프로그램에서 번지계산을 용이하게 하였다.

2. 심볼테이블

심볼테이블 구성을 어셈블러로 부터 분리하기 위해서 심볼테이블에 대한 기능을 초기화기능과 검색기능을 구성하였다. 심볼을 사용자가 정의하는 마이크로명령어와 의사명령어인 정적심볼과 레이블 및 수치데이타인 동적심볼로 분류하고, 정적심볼들을 초기화 기능을 사용하여 어휘분석 전단계에서 심볼테이블에 수록하도록하고, 검색기능을 어휘분석단계에 포함시켜 동적심볼들이 검출될 때 마다 심볼테이블에 수록하므로 마이크로명령어 집합을 사용자가 정의하도록 하였다. 심볼테이블의 구성 필드별 용도는 표 4와 같다.

표 4. 심볼테이블 구조
Table 4. Structure of symbol table.

필 드	용 도
status	심볼의 기재 및 미기재 사항을 구분
name	마이크로명령어와 의사명령어 명칭
type	심볼타입을 5가지로 종류 구분
subtype	의사명령어 분류
attribute	절대번지와 상대번지의 구분
value	상수일 경우 실제값, 레이블일 경우 번지값
size	데이타의 단위 길이

3. 어휘분석

어휘 분석은 첫번째 심볼을 찾고 다음 델리미터를 찾는다. 원시 프로그램은 공백으로부터 시작되거나 또는 주석으로 시작될수 있으므로 우선적으로 공백과 주석을 처리한다. 다음 첫문자를 검출하여 숫자인가를 판정한다.

표 5. 토큰 구성
Table 5. Token structure.

필 드	용 도
type	분리형,계산형,수치데이타 구분
subtype	분리형,계산형,수치데이타 세분화
new	심볼테이블의 등록여부 판정
value	상수

숫자인경우는 첫문자는 아라비아숫자 0으로 시작하고 두번째 영문자로 2진, 8진,10진,16진을 구분하여 수치데이타에 해당하는 심볼토큰을 구성한다. 아니면 레이블이나 마이크로명령어에 해당하는 심볼들이므로

심볼테이블을 검색하면서 레이블인 경우 새로운 것이면 심볼테이블에 등록을 시킨후 이에 해당하는 토큰을 생성한다. 공백과 심볼에 대한 처리를 하고 난후 델리미터를 찾아 이들에 대한 토큰을 심볼토큰과 델리미터토큰으로 생성한후 구문 및 의미분석기에 전달한다. 토큰의 구조는 표 5와 같다.

4. 구문 및 의미 분석

어휘분석기로 부터 전달된 심볼과 델리미터 토큰 한 쌍을 구문 및 의미 분석에서 3단계로 나누어 처리하였다. 첫단계에서 델리미터 토큰이 콜론이면 심볼토큰이 레이블이므로 레이블로 처리하고, 아니면 두번째 단계에서 심볼토큰이 명령어인가 또는 의사명령어인가를 심볼테이블을 검색 비교하면서 판별한다. 의사명령어인 경우 마이크로어셈블러가 그에 해당하는 실제적인 처리를 하고, 마이크로명령어인 경우에는 세번째 단계에서 중간 코드를 생성한다. 마이크로어셈블러의 어휘분석과 구문 및 의미분석에 대한 순서도는 그림 4.와 같다.

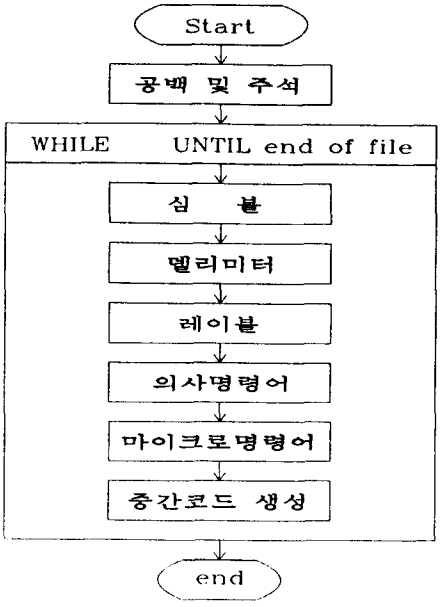


그림 4. 어휘분석 과 구문 및 의미 분석 흐름도
Fig. 4. Lexical and Parser flow chart.

Ⅳ. 결과 및 검토

마이크로프로그래머가 직접 정의한 마이크로명령어와 의사명령어와 델리미터들을 사용한 SMM의 곱셈 마이크로프로그램은 그림 5.와 같으며 고급 프로그램 언어가 갖고있는 문서화의 효과가 있다.

그림 5의 SMM의 곱셈 마이크로루틴을 입력으로 생성된 중간코드는 그림 6과 같다.

본 논문에서는 마이크로명령어 형식 정의 기능을[12]

```
/*============================ */
/*        MULTIPLY microprogram        */
/*============================ */
  /*set value */
  equ R1  = 0x1;
  equ R2  = 0x2;
  equ R3  = 0x3;
  equ R4  = 0x4;
  equ R11 = 0x11;
  equ R12 = 0x12;
  /* MTL routine */
org 0x83;
 MOVE_GPR_MAR R12; /* MAR <- R12 Fetch
 next */
  READ;  /* READ  operand  */
  MOVE_MDR_GPR R2; /* R2 <- MDR into R2
  */
  MOVE_GPR_MAR R11; /* MAR <- R11 Fetch
  second */
  READ; /* READ operand */
  MOVE_MDR_GPR R3; /* R3 <- MDR into R3
  */
  SET_GPR R1,0x0; /* R1 <- 0 Initialize R1 for
  product */
  SET_GPR R4,0x16;/* R4 <- 16 Initialize R4 for
  counting*/
RTN1:
  SHIFT_RIGHT_LITER R3,0x1; /* R3 <- R3 RS 1
  Multiply R2 by */
  IF_SHIFTOUT_ZERO_GO $ + 0x2; /* IF shiftout
= 0 R3 using a shift */
  /* GO TO current address+2 and add */
  ADD  R1,R2; /* R1 <- R1 + R2  algorithm */
  SHIFT_LEFT_LITER R2,1; /* R2 <- R2 LS 1
  and put */
  DECREMENT_GPR R4; /* R4 <- R4 - 1 the
  result */
  IF_NOT_ZERO_GO RTN1; /* IF NOT ZERO GO
  TO RTN1 into R1 */
  MOVE_GPR_MDR R1; /* MDR <- R1 Write
  result into */
  MOVE_GPR_MAR R12; /* MAR <- R12 memory
  at address */
  WRITE; /* WRITE of first operand */
  GOTO 0x64; /* GO TO 64 Branch to instruction
  */
  /* fetch microprogram*/
```

그림 5. SMM의 곱셈 마이크로프로그램
Fig 5. Multiply microprogram for SMM.

첨가하므로 기존 어셈블러의 단일 실행코드 생성의 단점을 제거하였다. 그림 6의 중간코드를 입력으로 SMM의 수평 마이크로명령어의 형식에 의거 번역 생성된 마이크로코드는 각각 17개의 필드이며 그림 7과 같다. 괄호안은 필드의 길이이다.

```
The Intermediate code of parser
   Address : 83
   Operand : 12
   Microinstruction : MOVE_GPR_MAR
   Microinstruction : READ
   Operand : 2
   Microinstruction : MOVE_MDR_GPR
   Operand : 11
   Microinstruction : MOVE_GPR_MAR
   Microinstruction : READ
   Operand : 3
   Microinstruction : MOVE_MDR_GPR
   Operand : 1
   Operand : 0
   Microinstruction : SET_GPR
   Operand : 4
   Operand : 16
   Microinstruction : SET_GPR
   Operand : 3
   Operand : 1
   Microinstruction : SHIFT_RIGHT_LITER\
   Operand : 95
   Operand : 2
   Operation : +
   Microinstruction : IF_SHIFTOUT_ZERO_GO
   Operand : 1
   Operand : 2
   Microinstruction : ADD
   Operand : 2
   Operand : 1
   Microinstruction : SHIFT_LEFT_LITER
   Operand : 4
   Microinstruction : DECREMENT_GPR
   Operand : 93
   Microinstruction : IF_NOT_ZERO_GO
   Operand : 1
   Microinstruction : MOVE_GPR_MDR
   Operand : 12
   Microinstruction : MOVE_GPR_MAR
   Microinstruction : WRITE
   Operand : 64
   Microinstruction : GOTO
```

그림 6. 중간코드 결과
Fig 6. The result of intermediate code.

F1	F2	F3	F4	F5	F6	F7	F81	F82	F83	F84	F9
F10	F11	F12	F13	F14							
(2) 0	(3) 1	(4) 12	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 1	(3) 0	(4) 0	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 3	(4) 2	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 1	(4) 11	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 1	(3) 0	(4) 0	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 3	(4) 3	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 7	(4) 1	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 7	(4) 4	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 16	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 0	(4) 0	(4) 0	(3) 0	(4) 0	(4) 0	(2) 2	(1) 1	(1) 0	(1) 0	(4) 3
(16) 1	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 0	(4) 0	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 10	(3) 4	(1) 1	(3) 0	(4) 0							
(2) 0	(3) 0	(4) 0	(4) 0	(3) 1	(4) 1	(4) 2	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 0	(4) 0	(4) 0	(3) 2	(4) 4	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 0	(4) 0	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 16	(3) 1	(1) 1	(3) 0	(4) 0							
(2) 0	(3) 2	(4) 0	(4) 1	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 0	(3) 1	(4) 12	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							
(2) 2	(3) 0	(4) 0	(4) 0	(3) 0	(4) 0	(4) 0	(2) 0	(1) 0	(1) 0	(1) 0	(4) 0
(16) 0	(3) 0	(1) 0	(3) 0	(4) 0							

그림 7. 마이크로코드 생성 결과
Fig 7. Microcode generation result.

V. 결 론

기존의 어셈블러를 사용하여 마이크로프로그래밍을 할 때 마이크로프로그래머가 명령어 집합과 실행코드형식을 변경하지 못하는 단점을 제거하기 위하여 구조적 마이크로어셈블러를 설계하였다. 설계된 구조적 마이크

로어셈블러는 IBM-PC상에서 C-언어, FLEX와 BISON을 이용하였다. 마이크로어셈블리 언어의 문서 구성 요소인 마이크로명령어 집합과 실행코드의 형식을 사용자가 직접 정의하므로 마이크로프로그래밍할 때 어셈블러의 단점인 머신 종속적이며 기호언어로서의 난해함과 같은 단점을 제거하였다. 따라서 구조적이고 머신 독립적이면서 시스템 설계변경과 광범위한 마이크로아키텍처에 대응하는 사용자 정의 마이크로어셈블러를 설계하였다. 구조적 마이크로어셈블러에서 마이크로명령어 형식을 제 약없이 정의하므로써 광범위한 마이크로아키텍처에 대응하고, 마이크로프로그램의 정확성과 생산성을 향상시키고, 기호언어로서의 난해함등을 제거한 마이크로프로그래밍을 위한 효과적인 도구이다. 또한 IBM-PC상에서 프로그램을 모듈화 하여 설계하였기 때문에 마이크로어셈블러 개발시간을 단축시키며 기존에 익숙한 어셈블리 언어를 그대로 마이크로프로그래밍에 적용할 수 있도록 편의성을 제공하였고, 개발 비용면에서 실험실 단위의 경제성 및 활용성이 높다. 추후 소형 제어메모리 내에 들어갈 수 있도록 마이크로프로그램 크기의 최소화를 고려하면 효과적인 마이크로코드를 생성할수 있다.

참 고 문 헌

[1] Myers G.J, Digital System Design with LSI Bit-Slice Logic,John Wiley & Sons, New York, p.338, 1980.

[2] AMD Data Book 32-bit Microprogrammable Product, 1988.

[3] Dasgupta,S., and Shriver,B.D. "Developments in Firmware Engineering." Advances in Computers, Vol.24. pp.101-176, 1985.

[4] Tredennick, N."The Cultures of Microprogramming",Proc. MICRO 15, pp. 79-83, Oct. 1982.

[5] Wilbum, D. L., and Schleimer, S., "STEP Development Tools: METASTEP Language System." Proceedings of the 18th Annual Workshop on Microprogramming, pp.64-78,Oct,1985.

[6] Takahasi, K., Takahasi, E., Bitoh, T. Sugimoto, T., et al., "A New Universal Microprogram Converter." Proceedings of the 17th Annual Workshop on Microprogramming, pp.264-266, Oct,1985.

[7] Tracz, W. J., "Advances in Microcode Support Software." Proceedings of the 18th Annual Workshop on Microprogramming, pp.57-60,Oct, 1985.

[8] Habib, s., Microprogramming and Firmware Engineering Methods. New York, Van Nostrand Reinhold, 1988.

[9] Auli,R., Antti,A., and Ari, O., "Automatic Synthesis of Structural HDL Descriptions From Graphic Specification of Embedded ASICS." Microprocessing and Microprogramming (27), pp.473-478,1989.

[10] Miroslav Sveda, "Microcontroller Software Engineering" Microprocessing and Microprogramming (34), pp.11-14 ,1992.

[11] Christos A. Papachristou and Satnamsingh B. Gambhir, "Microcontrol architectures with sequencing firmware and modular microcode development tools" Microprocessing and Microprogramming (29), pp.303-328,1991.

[12] 신인철,신봉희, "마이크로명령어 정의어 설계", 대한전자공학회 논문지 제31권 B편 제10호, pp.92-98, 1994

[13] Ashok K. Agrawala and Tomlinson G. Rauscher, Foundations of Microprogramming,Academic Press,New York,1976.

저 자 소 개

申 鳳 熙(正會員) 第31卷 B編 第10號 參照
현재 시립인천전문대 전자계산과 교수

金 成 鍾(正會員)
1964年 1月生. 1987年 2月 단국대학교 전자공학과 졸업(공학사). 1989年 2月 단국대학교 대학원 전자공학과 졸업(공학석사). 현재 단국대학교 대학원 전자공학과 박사과정 재학중. 주관심분야는 병렬처리, 혼돈이론, 컴퓨터 및 퍼지 제어등임.

李 俊 模(正會員)
1949年 3月生. 1975年 2月 명지대학교 전자공학과 졸업(공학사). 1979年 8月 명지대학교 대학원 전자공학과 졸업(공학석사). 1988年 3月 - 현재 단국대학교 대학원 전자공학과 박사과정 재학중 1980年 3月 - 현재 관동대학교 전자공학과 부교수.주관심분야는 멀티프로세서 디자인, 컴퓨터 및 퍼지 제어, 신경 회로망등임.

申 仁 澈(正會員) 第31卷 B編 第10號 參照
현재 단국대학교 전자공학과 교수