論文94-31B-10-10

마이크로명령어 정의어 설계

(Design of Microinstruction Definable Language)

申 鳳 熙*, 李 必 宰**, 申 仁 澈***

(Bong Hi Shin, Pil Jai Lee and In Chul Shin)

要約

마이크로프로그래밍 비용을 최소화 시키는 체계적인 마이크로프로그램 개발 환경을 마련하기 위해서, 본 논문에서는 마이크로프로그래머블 머신에대한 마이크로프로그래밍 과정을 단계별로 고찰하여 마이크로프로그램 특성상 하드웨어와 밀접한 관계를 유지하며 효과적인 마이크로코드를 생성하는 마이크로명령어 정의어를 제안하였다.

Abstract

In this paper, each step of microprogramming was discussed using a microprogrammable machine that would grasp the characteristics of the microprogram in close relation to the hardwares and microinstruction definable language which make an effective microcode. This was proposed for an environment which minimizes costs.

I. 서 론

반도체 기술의 발달로 저가격의 고속 WCS (Writable Control Store)가 출현함에 따라 마이크로프로그래머블 프로세서(Microprogrammable processor)가 등장하였다. [1-2] 이들 마이크로프로그래머블 프로세서를 이용하여 시스템을 제어하면 규칙성 (Structured and regular method) 때문에 시스템의 분할설계가 용이하며 설계시간 감축으로 경비의 감소가 기대되어 전용 프로세서 설계에 마이크로프로그램을 이용한 제어방식이 활용된다. [3-4] 마이크로프로그램 제어방식을 채택하여 시스템을 설계할때 전체 비용에 대한 하드웨어의 비용은 현저하게 줄어들고, 반면에 소프트웨어인 마이크로프로그램 개발 비용이 상대적으로 크게 증가한다. 그러므로 소프트웨어 비용을 최소화 시킬수 있도록 체계적인 마이크로프로그램 개발 환경 개발이 필수적이다. [5-11]

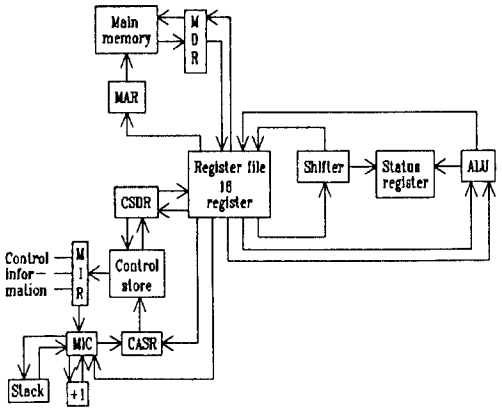
본 논문에서는 마이크로프로그래머블 머신(SMM: Simple Microprogrammable Machine) 기본 구조 [12]를 이용하여 마이크로프로그래밍의 과정인 하드웨어의 자원 및 기본연산(Primitive operation)을 분류 정의하고, 정의된 기본연산을 조합하여 복합연산

*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, 大宥工業專門大學 電子計算科
(Dept. of Computer Science,Dae Yeu Techical Junior College)
***正會員, 壇國大學校 電子工學科
(Dept. of Electronic Eng.,Dankook Univ.)
接受日字 : 1994年 5月 3日

(Complex operation)들을 정의하였다. 특히 기본연산에 대한 마이크로명령어 형식을 제한 없이 정의하며 효과적인 마이크로코드를 생성하는 마이크로명령어 정의어를 설계하였다. 마이크로명령어 정의어는 마이크로프로그램 특성상 하드웨와 밀접한 관계를 유지하며 마이크로프로그래밍을 할 수 있어 체계적인 마이크로프로그램 개발 환경을 지원한다. 제안된 마이크로명령어 정의어는 사용자가 마이크로오퍼레이션(Micro-operation)을 정의함으로서 고급 프로그램언어의 특징 즉 보다 간결하고, 이해하기 쉬우며, 정확성에 대한 자체 검증력이 높아 유지 보수가 용이하다. 또한 마이크로명령어 형식에 있어 명칭이나 갯수에 제한을 받지 않으므로 머신 독립적이며 보다 수정하기 쉽고 확장성이 뛰어난 특성을 갖게 하였다. 이들 모두 IBM-PC상에서 C언어, FLEX와 BISON으로 구현하였다.

Ⅱ. 마이크로명령어 정의어

마이크로프로그래머블 머신은 마이크로프로그래밍에 의한 제어부의 구성을 하드웨어 자원과 기본연산을 근거로 시스템 설계자가 직접 마이크로프로그래밍을 한다. 일련의 기본연산들로 복합연산들이 정의되며 복합연산들은 제어부의 기억장치에 저장된후 운영된다. 복합연산을 구성하는 각각의 기본연산들을 마이크로명령이라하며 마이크로명령들을 포함하고 있는 기억영역은 제어메모리(Microprogram Memory)이다. 마이크로명령은 시스템 하드웨어 자원을 제어하는 명령으로써 하드웨어와 밀접한 관계를 유지하면서 마이크로프로그래머가 마이크로명령과 마이크로명령어 형식을 이용하여 마이크로루틴(Microroutine)을 작성하면 이들은 번역되여 마이크로코드가 생성된다.



CSAR(control store address register), CSDR(control store data register)
MIR (microinstruction register), MIC (microinstruction counter)
MDR (memory data register), MAR (memory address register)

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

1. SMM과 기본연산

기본적인 SMM은 그림 1과 같다. 제어메모리내의 각 마이크로명령은 MIR을 통하여 시스템 전체의 제어점에 제어신호를 공급하게 되며 CPU의 레지스터 화일과 데이타를 직접 주고 받을수 있다. 마이크로명령은 자신의 하드웨어 구성에 근거하여 정의된다. 그림 1의 SMM에 대해 제안된 마이크로명령어 정의어를 사용하여 마이크로프로그래밍하는 첫단계로 하드웨어 자원과 데이타에 대해 동작하는 마이크로명령인 레지스터간 데이터 이송연산(Simple Register Transfer Operation), 단항 및 이항 연산인 데이타 변환 연산(Transformation Operation), 처리 순서 연산(Sequencing Operation), 기억부 입출력 연산(Memory Operation)들과 분기(Branching)제어들을 체계적으로 분류하고, 분류된 마이크로명령들은 일반 컴퓨터의 기계어 명령에 해당하는 모든 마이크로루틴을 작성할 수 있도록 타당성이 있어야 한다. SMM의 마이크로오퍼레이션들에 대해 표 1과 같이 분류하고 명칭을 부여한다.

표 1. SMM 마이크로오퍼레이션과 명칭
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

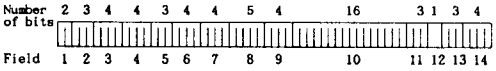
표 1에서 부여한 마이크로명령의 명칭들을 사용하여 SMM을 운영하기 위한 각각의 마이크로루틴이 정

의되며 곱셈에 대한 마이크로루틴의 예를 표 2에 나타내었다.

표 2. SMM의 곱셈 마이크로루틴

Table 2. SMM multiply microroutine.

Microroutine			Meaning
MOVE_GPR_MAR	12	MAR <- R12	Fetch first operand into R2
READ		READ	
MOVE_MDR_GPR	2	R2 <- MDR	
MOVE_GPR_MAR	11	MAR <- R11	Fetch second operand into R3
READ		READ	
MOVE_MDR_GPR	3	R3 <- MDR	
SET_GPR	1,0	R1 <- 0	Initialize R1 for product
SET_GPR	4,16	R4 <- 16	Initialize R4 for counting
SHIFT_RIGHT_LITER	3,1	R3 <- R3 RS 1	Multiply R2 by R3 using a shift and add algorithm and put the result into R1
IF_SHIFTOUT_ZERO_GO	94	IF shiftout = 0 GOTO 94	
ADD	1,2	R1 <- R1 + R2	
SHIFT_LEFT_LITER	2,1	R2 <- R2 LS 1	
DECREMENT_GPR	4	R4 <- R4 - 1	
IF_NOT_ZERO_GO	91	IF NOT ZERO GO TO 91	
MOVE_GPR_MDR	1	MDR <- R1	Write result into memory at address of first operand
MOVE_GPR_MAR	12	MAR <- R12	
WRITE		WRITE	
GOTO	64	GO TO 64	Branch to instruction branch microroutine



Field code points

1 Memory operation
- 0 - no operation
- 1 - read memory
- 2 - write memory

2 Register transfer
- 0 - No operation
- 1 - MAR <- GPR1
- 2 - MDR <- GPR2
- 3 - GPR1 <- MDR
- 4 - CS at GPR1 <- GPR2
- 5 - GPR1 <- GPR2
- 6 - GPR2 <- CS at GPR1
- 7 - GPR1 <- literal (field 10)

3 GPR1

4 GPR2

5 Tranformation operator GPR3 <- GPR3 operator GPR4
- 0 - No operation
- 1 - +
- 2 - -
- 3 - + with carry
- 4 - &
- 5 - v
- 6 - not

6 GPR3

7 GPR4

8 Shift operator
- Bits 1-2 shift operation
  - 0 - No operation
  - 1 - Shift amount in GPR5
  - 2 - Shift amount is literal
- Bit 3 Shift direction
  - 0 - left
  - 1 - right
- Bit 4 Shift type
  - 0 - arithmetic
  - 1 - logical
- Bit 5 Shift out
  - 0 - end off
  - 1 - circular

9 GPR5 or shift literal

10 literal

11 Condition
- 0 - no condition
- 1 - ALU result = 0
- 2 - ALU result < 0
- 3 - ALU carry out
- 4 - Shift out bit

12 True (1) or false (0)

13 Sequenceing operation
- 0 - Step
- 1 - Push CSAR+1 to stack
- 2 - Pop stack and assign to CSAR
- 3 - Skip
- 4 - CSAR <- GPR5
- 5 - CSAR <- literal (field 10)

14 GPR5

그림 2. SMM의 수평 마이크로명령어 형식

Fig. 2. A horizontal microinstruction format for the SMM.

2. 마이크로명령어

마이크로명령은 시스템 하드웨어 자원을 제어하기 위한 명령이다. 마이크로명령어 설계는 첫번째로 하드웨어 자원의 제어에 관련한 마이크로명령어내의 존속할 필드를 추출한다. 두번째로 마이크로명령어내에 필드들을 배치한다. 배치방식에 따라 수직(Vertical)과 수평(Horizontal) 마이크로명령어로 나뉘어 구성된다. 수직 마이크로명령어는 단일 마이크로명령을 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 마이크로명령을 수행하도록 필드별 용도와 위치를 구별시켜 구성한다. SMM의 수평 마이크로명령어 형식은 그림 2와 같다.

3. 마이크로명령어 정의어

SMM의 마이크로프로그래밍 단계를 검토한 결과 제안한 마이크로명령어 정의어가 아래와 같은 조건을 갖추면 최소한의 비용으로 효과적인 마이크로코드를 생성시킨다.

① 마이크로프로그램 문서화
② 머신 독립성을 유지하면서 다양한 마이크로아키텨쳐에 대응
③ 시스템의 하드웨어 설계 변경에 따른 마이크로명령어 형식 변경 및 전환이 용이
④ 일반 고급 프로그램언어에서 활용
⑤ 수직 및 수평 마이크로명령어 생성
⑥ 마이크로명령어 필드 연산
⑦ 마이크로프로그램의 모듈별 제어

하드웨어 자원을 체계적으로 분류한후 각 마이크로명령에 대한 명칭을 마이크로프로그래머가 정의할 수 있고, 이들 명칭을 사용하여 마이크로루틴을 작성한다면 마이크로프로그램은 문서로써 기능이높고, 저급언어가 갖고있는 기호언어로써의 난해함을 제거할 수 있게 된다. 따라서 마이크로프로그램의 이해도, 정확도, 검사 기능등을 높이는 효과가 있다.

머신 독립성을 유지고, 하드웨어와 밀접한 관계를 유지하며 다양한 마이크로아키텍쳐에 대응하고, 시스템 하드웨어 설계변경에 따라 마이크로명령어 형식 전환이 용이하며, 일반 고급 프로그램언어에 활용을 위해 마이크로명령어의 형식은 명칭이나 갯수에 제한을 받지 않고 마이크로프로그래머에 의해서 정의될 수 있어야 한다.

또한 수직과 수평 마이크로명령어 생성은 주와 부 마이크로명령어로 나뉘어 정의하면 가능하다. 수직 마이크로명령어는 주와 부 마이크로명령어형식을 구

분하지 않고 각각의 마이크로명령별로 마이크로명령어를 정의하여 해당하는 마이크로명령어만을 출력하도록하고, 수평 마이크로명령어는 주 마이크로명령어 형식에 각각의 마이크로명령에서 필요로하는 모든 연산자와 피연산자 필드들을 포함하게하고, 부 마이크로명령어 형식은 각각의 마이크로명령별로 필요한 연산자와 피연산자 필드로만 구성한다. 각각의 마이크로명령에 해당하는 부 마이크로명령어들을 주 마이크로명령어에 접목시켜 출력하도록한다.

마이크로프로그래머에 의해 부여된 명칭들을 이용하여 마이크로루틴들이 작성되면 마이크로루틴별로 작동 여부 및 효율성에 대한 검토를 하고, 아울러 수정을 할 수 있고, 마이크로루틴별 모듈이 완성되면 모듈간의 상호관계를 고려해서 이들을 결합시켜 하나의 전체 마이크로프로그램을 구성하도록 마이크로코드에 대한 연산 및 제어 기능이 마이크로명령어 정의어에 포함되어야 한다.

이상과 같이 마이크로프로그래머에 의해서 정의된 마이크로루틴은 마이크로명령어 정의어 번역기에 입력되어 효과적인 마이크로코드로 번역된다.

Ⅲ. 마이크로명령어 정의어 설계

마이크로명령어 정의어는 마이크로명령어 형식 정의부, 마이크로코드 생성부와 마이크로코드 연산 및 제어부를 모듈별로 분리하여 설계하였다. 이에 대한 순서도는 그림 3과 같다.

1. 마이크로명령어 형식 정의

마이크로명령어 형식 정의는 주와 부 마이크로명령어로 정의되며 형식은 머리부와 몸체부로 나뉘어 명칭,길이,항목들이 정의된다. 수직, 수평 마이크로명령어 형식을 동시에 정의하기 위하여 마이크로명령어를 주 마이크로명령어와 부 마이크로명령어로 나누어 정의하도록 하였다. 마이크로명령어를 정의할 때 사용하는 문자들은 아라비아 숫자, 영문자, 특수문자이고, 사용단어들은 예약어, 사용자정의어들이다. 이들을 사용하여 마이크로명령어 형식이 정의된다. 정의시 사용하는 문자와 단어는 다음과 같다.

사용문자

숫자 : 0에서 9까지 사용한다.(0-9)

영문자 : 알파벳 대소문자.(a-z, A-Z)

특수문자

";" : 주석용

":" : 마이크로명령어 명칭과 길이등을 구분

"{,}" : 마이크로명령어 몸체부 구분

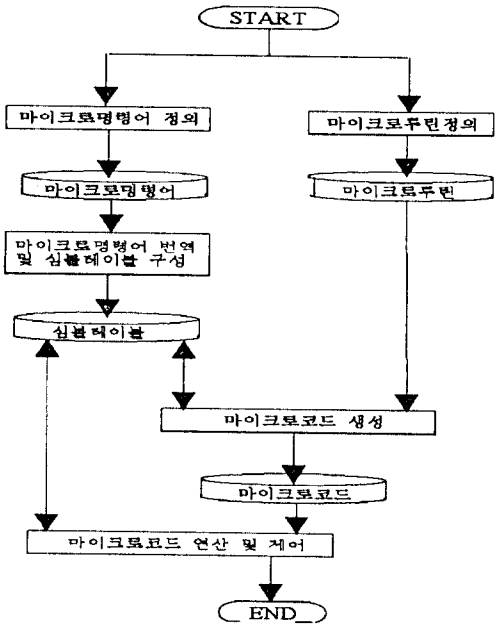


그림 3. 마이크로명령어 생성 과 마이크로코드 연산 및 제어 순서도

Fig.3. Flow chart for microinstruction generation and microcode operation and control.

microinstructions	microinstruction microinstructions microinstruction
microinstruction	head body
head	name TK_INSTRUCTION total_length fields
name	TK_IDENTIFIER ':'
total_length	TK_LENGTH '(' TK_CONSTANT ')'
fields	TK_FIELDS '(' TK_CONSTANT ')'
body	'{' field_statements '}'
field_statements	field_statement field_statements field_statement
field_statement	TK_IDENTIFIER ':' field_length TK_IDENTIFIER ':' field_length
field_length	TK_LENGTH '(' TK_CONST ')'
field_value	TK_VALUES '(' values ')'
values	TK_CONST values ',' TK_CONST

그림 4. 마이크로명령어 형식 정의 구조

Fig. 4. The structure of microinstruction format definition.

"(,)": 마이크로명령어길이, 필드의거수, 필드의 값에대한 상수 표시

"," : 필드의 값이 여러개 존재할때 구분

예약어

instruction : 주,부 마이크로명령어 이름과 길이 구분

length : 마이크로명령어 및 필드의 길이

fields : 마이크로명령어내의 필드 이름

values : 마이크로명령어내의 필드 값

사용자 정의어

: 마이크로명령어 또는 항목 명칭

주와 부 마이크로명령어 정의는 각각 머리부와 몸체부로 나뉘어 정의 된다. 머리부는 마이크로명령어 명칭, 총길이와 항목의 갯수를 정의한다. 몸체부에서는 각 항목에 대한 정보를 정의하고, 항목의 명칭과 길이와 항목이 갖을수 있는 값들을 정의한다. 마이크로명령어 형식 정의 구조는 그림 4와 같다.

그림 2의 SMM의 수평 마이크로명령어 형식을 마이크로명령어 정의어로 정의하면 그림 5와 같다.

```
MICROSMM : INSTRUCTION LENGTH(60) : Microinstruction name & length
           FIELDS(14)             : The number of field
           { F1 : LENGTH(2)       : Memory operation
                  VALUES(0,1,2)
             F2 : LENGTH(3)       : Register transfer
                  VALUES(0,1,2,3,4,5,6,7)
             F3 : LENGTH(4)       : GPR1
             F4 : LENGTH(4)       : GPR2
             F5 : LENGTH(3)       : Tranformation operator
                  VALUES(0,1,2,3,4,5,6)
             F6 : LENGTH(4)       : GPR3
             F7 : LENGTH(4)       : GPR4
             F8 : LENGTH(5)       : Shift operator
                  VALUES(0,1,2,3,4,5)
             F9 : LENGTH(4)       : GPR5 or shift literal
             F10: LENGTH(16)      : Literal
             F11: LENGTH(3)       : Condition
             F12: LENGTH(1)       : True or false
             F13: LENGTH(3)       : Sequencing operation
                  VALUES(0,1,2,3,4,5)
             F14: LENGTH(4)       : GPR5
           }
MOVE_GPR_MAR : INSTRUCTION LENGTH(6) : Move GPR to MAR
           FIELDS(2)
           { F2 : LENGTH(2)
                  VALUES(1)
             F3 : LENGTH(4)
           }
           .
           .
           .
IF_SHIFTOUT_ZERO_GO :
           {
           }
```

그림 5. SMM 수평 마이크로명령어 형식 정의
Fig. 5. Microinstruction format definition for SMM.

2. 마이크로명령어 심볼테이블

마이크로명령어의 형식이 정의된후 마이크로명령어 번역기에 입력되면 이를 번역하여 마이크로명령어 심볼테이블을 설정한다. 수평마이크로명령어인 경우 마이크로명령어 심볼테이블은 첫번째 주 마이크로명령어에 대하여 설정되고 다음에 부 마이크로명령어들에 대해서 차례로 설정된다. 마이크로명령어 심볼테이블은 마이크로코드 생성부에의해 참조되면서 마이크로코드를 생성하고, 마이크로코드에 연산 및 제어부에서도 참조한다. SMM의 수평 마이크로명령어에 대한 심볼테이블 구조를 그림 6에 나타내었다.

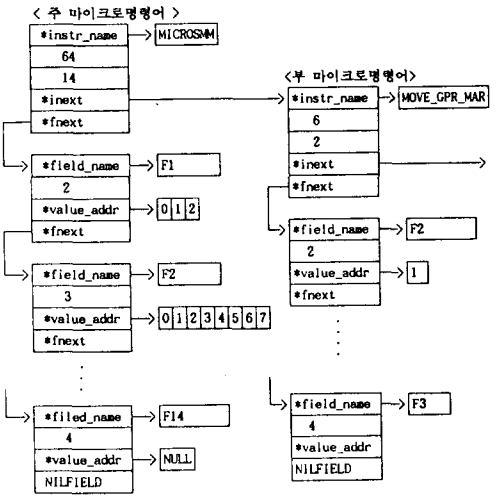


그림 6. SMM 수평 마이크로명령어 심볼테이블
Fig. 6. Symbol table of SMM horizontal microinstruction.

3. 마이크로코드 생성과 연산 및 제어

마이크로프로그래머가 마이크로명령어의 형식을 정의하면 마이크로명령어 번역기가 일차로 번역하여 마이크로명령어의 형식 정의에 따라 마이크로명령어 심볼테이블을 구성하고, 이차로 마이크로오퍼레이션들로 구성된 마이크로루틴이 입력된다. 일단 마이크로코드 생성부는 이 마이크로오퍼레이션들이 심볼테이블에 부 마이크로명령어로써 등록이 되여 있나를 확인하고 수평인 경우 부 마이크로명령어를 주 마이크로명령어 형식에 동일 필드의 내용을 복사한후 주 마이크로명령어 형식에 의존하여 마이크로코드를 생성하고 수직인경우 검출된 부마이크로명령어에 의존하여 마이크로코드를 생성한다. 마이크로명령어 정의어 사용 명령들을 표 3

표 3. 마이크로코드 연산 및 제어 명령
Table 3. Microcode operation and control.

명 령	용 도
EXECUTIONL	execute microprogarm for each line
EXECUTIONM	execute microprogram module
WRITE	write microprogram to ROM
LIST	print out microprogram
DISPLY	display microinstruction format
BITS	chang hex to bit
LENGTH	length of field
INPUT	input the value to each field
DEFAULT	default value for field
VALUES	definitions of names for field values
VALID	a list of valid values for the field

과 같이 구성하여 마이크로프로그램의 유지 및 보수와 설계시스템의 실행과정 검증성을 높였다.

표 2의 SMM의 곱셈 마이크로루틴이 그림 5의 마이크로명령어의 형식에의거 마이크로명령어 정의어에 의해 번역 생성된 마이크로코드는 각각 60비트이고 14개의 필드이며 그림 7과 같다.

```
***** SMM_MULTIPLY_MICROROUTINE *****
Field 1  2   3   4    5   6   7    8    9   10
      11 12 13 14

00 001 1100 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
01 000 0000 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
00 011 0010 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
00 001 1011 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
01 000 0000 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
00 011 0011 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
00 111 0001 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
00 111 0100 0000 000 0000 0000 00000 0000 000000000010000
000 0 000 0000
00 000 0000 0000 000 0000 0000 10100 0011 000000000000001
000 0 000 0000
00 000 0000 0000 000 0000 0000 00000 0000 000000001011110
100 0 101 0000
00 000 0000 0000 001 0001 0010 00000 0000 000000000000000
000 0 000 0000
00 000 0000 0000 000 0000 0000 11000 0010 000000000000001
000 0 000 0000
00 000 0000 0000 010 0100 1111 00000 0000 000000000000001
000 0 000 0000
00 000 0000 0000 000 0000 0000 00000 0000 000000001011011
001 0 101 0000
00 010 0000 0001 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
00 001 1100 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
11 000 0000 0000 000 0000 0000 00000 0000 000000000000000
000 0 000 0000
00 000 0000 0000 000 0000 0000 00000 0000 000000001000000
000 0 101 0000
```

그림 7. 마이크로코드 생성 결과

Fig. 7. Microcode generation result.

Ⅳ. 결 론

본 논문에서 하드웨어와 밀접한 관계를 유지하며 마이크로프로그래밍을 하는 마이크로명령어 정의어를 제안하였다. 제안된 마이크로명령어 정의어는 하드웨어와 기본연산을 바탕으로 설계되었기 때문에 대상 머신에 제한 없이 광범위하게 적용할 수 있다.

마이크로오퍼레이션 명칭과 마이크로명령어의 형식을 정의하는 권한을 마이크로프로그래머에게 부여하여 마이크로프로그램의 정확성과 생산성 및 신뢰도를 향상시켰다. 일반 고급 프로그램 언어 번역과정에서 생성되는 중간코드와 연관해서 마이크로코드를 생성할수 있어 일반 고급 프로그램 언어가 갖고 있는 특성을 유지하면서 마이크로프로그래밍을 할수있다.

또한 마이크로코드에 대한 연산 및 제어기능을 부여하였기 때문에 시스템의 설계 변경에 따른 마이크로프로그램 변경이 용이하다. 마이크로프로그램 모듈별 하드웨어 실행과정을 검증하는 기능이 있고, 마이크로프로그램 모듈별 검증후 각각의 모듈을 하나의 프로그램으로 연결 할 수 있는 기능들을 갖추었기 때문에 최소한의 비용과 노력으로 마이크로프로그램을 작성할 수 있어 체계적이며 효율적인 마이크로프로그래밍 개발 환경의 구성이 가능하다.

參 考 文 獻

[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" Microproce- ssing and Microprogramming (34), pp.11-14, 1992.

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

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

著 者 紹 介

申 鳳 熙(正會員)

1955年7月 1日生. 1977年 2月 인하대학교 전자공학과 졸업(공학사).1981年 2月 인하대학교 대학원 전자공학과 졸업(공학석사). 현재 단국대학교 대학원 전자공학과 박사과정 수료. 1991年~1993年 미국 아이오와 주립대학 교환교수. 1978年3月~현재 인천전문대학 전자계산과 교수. 주관심 분야는 마이크로프로그래밍 언어 설계, 다중처리 컴퓨터, 퍼지제어, 신경회로망 등임.

申 仁 澈(正會員)

1949年11月生. 1973年 2月 고려대학교 전자공학과 졸업(공학사). 1978年 2月 고려대학교 대학원 전자공학과 졸업(공학석사). 1986年 2月 고려대학교 대학원 전자공학과 졸업(공학박사). 1984年~1985年 미국 미시간 주립대학 교환교수. 1979年~현재 단국대학교 전자공학과 교수. 주관심 분야는 다중처리 컴퓨터, 퍼지 제어, 신경 회로망 등임.

李 必 宰(正會員)

1950年 2月生. 1973年 2月 고려대학교 전자공학과 졸업(공학사). 1975年 2月 고려대학교 대학원 전자공학과 졸업(공학석사). 현재 단국대학교 대학원 전자공학과 박사과정 재학중. 1981年 3月~현재 대유공업전문대학 전자계산과 부교수. 주관심 분야는 영상처리, 신경망 컴퓨터 및 퍼지 제어 등임.