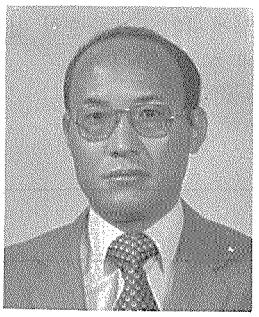


吳 吉 祿
韓国電子通信研究所
컴퓨터연구부장 / 工博

Microprogramming 기법에 관하여

"현재 생산하고 있는 컴퓨터들은 거의 대부분이 마이크로 차원 및 그 이하의 컴퓨터를 구성하고 있는 단계이므로 마이크로 프로그래밍의 장점들인 체계적 설계, 아키텍춰 수정의 용이성, 제어과정 설계의 융통성, 모델 상호간의 일치성 등의 필요성을 실감치 못하고 있다."

Ⅰ. 서 언

1951년초 Maurice V. Wilkes에 의해 제안된 마이크로 프로그래밍은 1950년대에는 몇몇 학교나 연구소를 제외하고는 거의 관심이 없었다. 그 이유는 이를 실용화하기 위해서는 고속과 저렴을 동시에 갖춘 기억소자가 필요한데, 그 당시 기억소자 설계기술은 이를 충족시킬 수준이 못되었다. 그러다가 1960년대에 들어서 갑작스런 기억소자 설계기술의 발전으로 많은 컴퓨터 설계자가 이방법에 관심을 갖게 되었고, 1960년대 중반 IBM사가 System/360의 몇 모델을 이 방법으로 설계하고부터 여러 컴퓨터사들이 이 방법으로 설계하기 시작했다.

마이크로 코드를 저장할 기억소자의 설계기술 향상과 마이크로 프로그래머블 마이크로프로세스의 개발은 마이크로 프로그래밍의 발전에 직결되어왔고 이의 발전은 마이크로 프로그래머 컴퓨터의 발전으로 나타났는데, 이들 하드웨어의 발전을 기반으로 한 마이크로 프로그래밍의 발전단계는, 많은 토론은 있었으나 실용화 되지 못했던 0세대, 실용화된 1세대, User 마이크로 프로그래머블한 머신이 출현한 2세대, 마이크로 프로그래밍을 위한 high level language 및 여러 응용분야가 개발된 3세대로 구분되며, 현재는 또 다른 차원의 영역이 개발되고 있는 제 4세대의 초기이다.

마이크로 프로그래밍은 그 동안 기억소자와 마이크로 프로그래머블 마이크로 프로세스 및 그 주변소자 등의 하드웨어 영역의 발전과 더불어 마이크로 아키텍춰, 컴팩션 등의 기본기술과 OS assistance, Graphics, Emulator, Microdiagnostics, Signal Processing, Program Enhancement 등의 여러 응용기술에서도 많은 발전을 해왔다.

본고에서는 0세대부터 제 4세대까지 마이크

로 프로그래밍의 불변의 기본과정인 하드웨어설계, 마이크로 인스트럭션 설계, 마이크로 프로그램 설계의 3단계를 기술한다.

Ⅱ. Microprogramming의 일반적기법

1. 개 요

마이크로 프로그래밍 기법은 그 마이크로 프로그래밍이 어떤 종류의 제어를 위한 것이냐에 따라서 아주 다양하나 여기서는 여러 기법 중 보편적인 것을 기술하기로 한다.

하고자하는 제어의 소양이 확정되면 이에 적합한 마이크로 아키텍춰를 마이크로프로그래밍 차원 및 기계어 차원을 동시에 고려하여 설계하고, 이의 이행을 위한 하드웨어를 구성하여 요구되어지는 기능을 하드웨어로 하여금 실행케 명하는 2진 코드 string의 sequence인 마이크로 코드를 설계하는데, 이러한 일반적인 과정을 크게 3단계, 즉 마이크로 아키텍춰의 이행을 위해 기본적으로 요구되어지는 「Basic 하드웨어 설계」단계, 이 하드웨어 자신들을 제어하기 위하여 필요한 정보들의 소양을 작성하는 「마이크로 인스트럭션 설계」단계, 이에 명시된 마이크로 오퍼레이션 혹은 마이크로 인스트럭션들로, 주어진 제어 기능을 적절히 수행하게 이의 sequences를 작성하는 「마이크로 프로그램 설계」단계로 구분하여 기술한다.

2. Basic 하드웨어 설계

마이크로 프로그래밍을 위해 기본적으로 필요한 하드웨어 콤포넌트는 「Control Storage」, 「Arithmetic and Logic Unit」, 「Sequencer」, 「Local Storage」, 「Main Storage」, 「Data Path」 등이 있으며, 이들 각각의 Functional Unit들의 구조 및 설계방법은 하드와이어드 머신과 차이가 있는데 이의 설계기준 및 유형을 기술한다.

가. Control Storage

CS는 마이크로 코드를 저장하는 특수한 기억장치이고 하드와이어드 머신에서는 필요치않은 콤포넌트이며, 초기단계에는 ROM Array를 이용했으나 현재에는 다이나믹하게 읽고 쓸 수 있는 기억소자로 구성하고 있다. 이의 설계시 중요한 요소들은 구조, 크기, 속도, 연관성, 레지스터 그리고 마이크로 인스트럭션 로드 방법 등이 있으며, 그 구조는 여러 유형으로 설계할 수 있는데 그 중 몇가지를 소개한다.

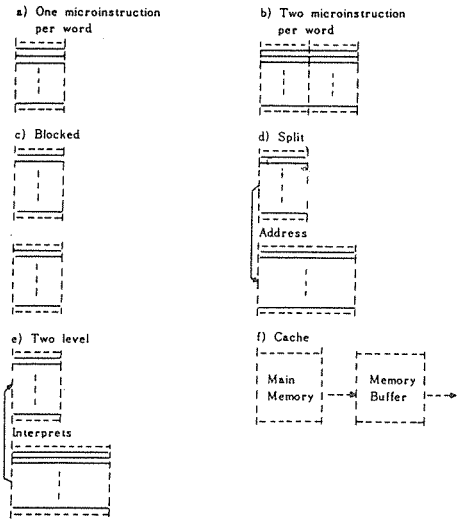


그림 1. CS구조

a) One microinstruction per word :

CS의 한 word가 하나의 마이크로 인스트럭션으로 구성되는 가장 단순하고 일반적인 방법이다.

b) Two microinstruction per word :

CS의 한 word가 두개의 마이크로 인스트럭션으로 구성되는 방법이며 이는 CS의 reference를 줄이는 장점이 있다.

c) Blocked

블록 주소와 블록내의 word 주소를 가지며 마이크로 인스트럭션을 위한 주소가 non-blocked 구조에 비해 짧다는 장점이 있다.

d) Split

2개의 word size가 다른 store unit로 구성되며 짧은 마이크로 인스트럭션이 자주 사용될 때 다른 유형에 비해 기억소자 비트 수가 적게 요구되는 장점이 있다.

e) Two level

마이크로 인스트럭션과 머신 인스트럭션의 관계와 유사하게 low level unit의 마이크로 인스트럭션이 upper level unit의 마이크로 인스트럭션을 인터퍼리테이션하는 유형으로 머신 인스

트럭션 및 upper level마이크로 인스트럭션 설계를 다양하게 할 수 있고 쉽게 바꿀 수 있는 장점이 있다.

f) Cache

마이크로 프로그램이 주기억장치나 버퍼내에 저장되어 있지만 그 시행은 Cache 메모리에서 하도록 설계하는 유형이며, 용량이 큰 주기억장치와 버퍼를 사용하기 때문에 마이크로 프로그램 설계자가 마이크로 프로그램 크기에 제한을 두지 않아도 되는 장점이 있다.

위에서 언급한 것 이외에도 주기억 장치에 마이크로 프로그램을 저장해 두고 바로 fetch 하여 시행시키는 유형 등등 여러가지가 있다.

나. Arithmetic and Logic Unit(ALU)

ALU는 종래의 하드와이어드 제어 머신과 마찬가지로 연산, 이동, 삭제 등의 오퍼레이션으로 마이크로 프로그램이 인터퍼리테이션하는 어떤 인스트럭션 세트를 이행하는 주역이며, 이의 설계시 고려되어야 하는 기본요소는 ALU의 구조, ALU가 이행해야하는 오퍼레이션의 종류, 이행속도, 오퍼런드와 시메트리 그리고 연관된 레지스터의 크기 등이 있으며 ALU의 구조에는 여러 유형이 있는데 그 예는 그림 2 와 같다.

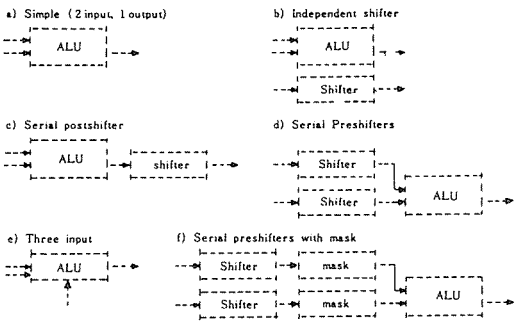


그림 2. ALU구조

a) Simple

ADD, SUBTRACT, AND, OR, NOT 같은 논리기능과 단순한 연산을 이행하는 구조이다.

b) Independent shifter

shift를 기본 ALU 오퍼레이션과 병행토록 설계하는 유형이다.

c) Serial postshifter

Postshift 기능을 이행할 수 있도록 기본 ALU의 출력단에 shifter를 둔 유형이다.

d) Serial preshifters

Preshift 기능을 이행토록 기본 ALU의 입력이 shifter를 거치게 설계하는 유형이다.

e) Three input

두 입력과 더불어 제 3 의 입력이 전단의 기본 ALU 오퍼레이션으로부터 입력되는 유형이다.

f) Serial preshifters with mask

Serial preshifters 유형에다 입력 오퍼런드 혹은 전단의 결과로부터 특정 비트를 택하도록 mask기능을 부여한 유형이다.

위에서 언급한 유형들을 기본으로 하여 전 마이크로 세트를 이행하는 복합 ALU를 구성하여야 한다.

다. Sequencer

Sequencer는 마이크로 인스트럭션의 sequence상에서 어떤 한 주소에서 다른 주소로 변이가 요구될 때 여러 경우의 다음 주소 중에서 현재 마이크로 인스트럭션상의 관련영역의 decode에 의해 다음 주소를 결정하는 논리회로이다. 다음 주소 결정 mechanism은 여러 유형이 있는데 일반적인 것은 그 입력으로 외부에서 직접 유입되는 입력 세트, ALU 내의 메모리의 한 출력 포터로부터 유입되는 데이터, stack 으로부터 유입되는 데이터, 프로그램 카운터의 주소 등이 있고 이들 중에서 마이크로 스탭마다 매번 관련 fields의 조건을 조사해서 다음 주소로 선택한다.

라. Local Storage

Local Storage는 마이크로 프로그래머 머신에서 주기억장치, CS, ALU 내부의 기억장치 이외의 기억장치인데 이들 세 기억장치들 사이를 왕래하는 데이터들을 일시적으로 보관하는 비교적 빠른 access time의 레지스터로 구성되며 한 머신내에 이용할 레지스터의 수와 그 크기, 그리고 구조 등을 마이크로 프로그래밍을 용이하게 할 수 있게 여러 유형으로 할 수 있다.

마. Main Memory

마이크로 프로그래밍 차원에서 주기억장치 구성에 영향을 미치는 것은 주기억장치의 크기, 즉 length 및 width 와 access 속도이다. 주기

억장치의 word수는 마이크로 프로그램 차원에서 그것을 직접 addressing할 수 있는 충분한 주기억장치 어드레스 레지스터(MAR)가 있어야 하고, 또 MAR은 이를 계산해내는 ALU 및 local 레지스터 같은 차원들에 영향을 미치므로 충분히 고려해야 하며 주기억장치의 width는 주기억장치 데이터 레지스터들의 크기에 또한 영향을 미친다. 그리고 주 프로세스 속도에 대한 주기억장치의 속도와 그들의 상호 관계는 마이크로 프로그래머 머신과 하드와이어드머신 구분없이 충분히 고려되어야 한다.

바. Data path

여러 기능의 콤포넌트들을 마이크로 프로그래밍의 관점에서 서로 유기적으로 연결시키는 그 크기(size)는 paths가 연결하는 레지스터 및 functional unit들의 크기를 반영하나 하드웨어의 비용과 마이크로 프로그램의 수행속도 등의 비중에 따라 이들의 path보다 훨씬 작거나 크게도 할 수 있다. 그 수는 마이크로 프로그래밍의 수행속도와 설계상의 융통성 및 용이성이 직접적으로 영향을 미치는데 이들의 비중에 따른 양극의 scheme을 융통성과 수행속도를 1로하여 전 하드웨어 자원들의 쌍마다 고유의 path를 둘 수 있는데 이는 최대의 융통성과 데이터 path이용의 concurrency를 제공할 수 있으나 하드웨어 설계상의 복잡성과 고비용의 문제가 있다. 그리고 용이성을 1로 하여 단 하나의 데이터 path로 할 수 있는데 이는 마이크로 프로그램 수행시 단 한 자원이 path상에 데이터를 실을 수 있고 또 다른 한 자원이 그 데이터를 이용할 수밖에 없어서 수행속도에 문제가 있다. 그러므로 이들 양극의 scheme을 적절히 절충하여 설계하여야 한다.

3. Microinstruction 설계

마이크로 인스트럭션 설계는 마이크로 프로그램을 위하여 요구되는 정보의 전체파토리를 마이크로 프로그래밍 및 하드웨어 구성의 관점에서 마이크로 word로 구성하기 위한 소양을 설계하는 것인데, 이는 결국 이에 대응하는 하드웨어 자원들을 제어하는 2진 코드의 string을 수행속도와 설계상의 용이성 및 융통성을 고려하여 생성하기 위한 것이다.

이의 설계를 위한 일반적인 단계는 다음과 같다.

첫째, 어떤 종류의 제어를 위한 시스템인가가 결정되면, 둘째, 그 시스템에서 이행할 기능들의 범위 및 종류를 정리하고, 세째, 그 기능을 이행하기 위한 세부 기능을 머신 인스트럭션 차원에서 분석하여 그룹을 정하고, 네째, 전 그룹에 공통적으로 필요한 하드웨어 자원을 분석하여 정리하고, 다섯째, 각 그룹의 머신 인스트럭션들을 실현하기 위하여 필요한 하드웨어 자원들을 마이크로 오퍼레이션 차원에서 분석하여 정리하고, 여섯째, 정리되어진 전 하드웨어 자원들의 각각을 제어할 수 있는 정보의 낱낱인 마이크로 오퍼레이션의 레파토리를 집약하고, 일곱째, 시스템의 특성과 마이크로 프로그램의 시행속도, 설계상의 융통성 및 용이성 등을 고려하여 마이크로 인스트럭션의 유형과 각 영역을 정하고, 여덟째, 한 마이크로 인스트럭션의 비트수를 영역수, decoding단계, 제어용 기억장치 등을 고려하여 최소화하여 정하고, 아홉째, 영역 및 영역내의 각 마이크로 오퍼레이션들의 배치를 시스템의 특성을 고려하여, 최적화하여 마이크로 인스트럭션 소양을 만든다.

이렇게 하여 만들어지는 마이크로 인스트럭션의 소양은 한 마이크로 인스트럭션에서 최대로 이용할 수 있는 마이크로 오퍼레이션의 수를 기준으로 크게 수직, 수평, 대각구조로 구분된다. 그림 3과 같은 2입력 1출력 ALU와 2포트의 16레지스터 그룹으로 구성되는 3paths 데이터 플로우의 단순한 구조를 바탕으로 마이크로 인스트럭션을 설계해보면 우선 마이크로 오퍼레이션은 다음 4가지가 있다.

- A path ⇦ 레지스터
- B path ⇦ 레지스터
- 한 ALU 오퍼레이션
- 레지스터 ⇦ C path

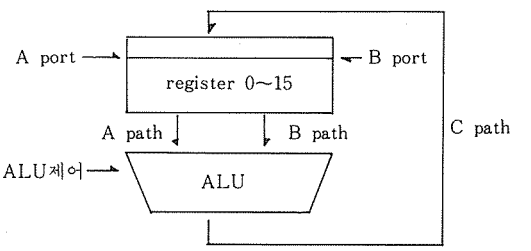


그림 3. MI 설계를 위한 기본구조

가. 수평구조

이 구조는 가능한한 여러 MO가 동시에 수행되도록 만드는 것이며 일반적으로 MI word는 여러 오퍼런드로 구성되며 크기(length)는 40~120비트 정도인데 70~80비트가 가장 많다. 이 구조의 장점은 하드웨어 자원들을 이용하는데 융통성이 많으며 존재하는 데이터 paths를 최대한으로 동시에 이용할 수 있고, 주어진 한 기능을 이행하는데 요구되는 MI의 수가 적기 때문에 시행시간이 짧다는 것들이 있는 반면, CS의 불충분한 이용과 마이크로 프로그램 및 컴파일러 설계의 어려움이 있다.

그림 3 에서 주어진 하드웨어 구조를 바탕으로 56비트 수평 마이크로 인스트럭션을 설계해 보면 다음과 같다.

(수평구조 MI 소양)

55 48	47 32	31 16	15 0
ALU op.	A path영역	B path영역	C path영역

(16진 코드)

0 1	0 0 2 0	0 0 4 0	0 0 8 0

ALU오퍼레이션 영역은 ALU내에서 수행될 오퍼레이션 중의 하나를 명시한다. 그리고 A path영역은 각각 A 및 B path로 게이트될 16레지스터 중의 하나를 명시하고, C path영역은 C path내용이 저장될 16레지스터 중의 하나를 명시한다. 만일 어떤 영역이 「all zero」로 명시되면 이는 그 마이크로 스탭에서 그 영역이 아무 의미를 갖지 않음을 뜻한다.

나. 수직구조

이 구조의 MI는 보통 하나의 ALU 오퍼레이션과 source와 destination을 명시하는 1, 2개의 오퍼런드로 이루어지며, MI word의 크기는 12~24비트 정도인데 16~20비트가 가장 많다. 이는 전 MI영역을 허비없이 모두 이용할 수 있고, MI소양이 단순하기 때문에 마이크로 프로그램 및 컴파일러 설계가 용이한 반면 MI의 스텝수가 많아지며 이는 곳 시행시간이 길어지는 단점이 있다.

그림 3 에서 주어진 하드웨어 구조를 바탕으로 수직구조를 설계해 보면 다음과 같다.

(수직구조 MI소양)

	17 16	15 0		
a)	A path 선택	레지스터	:	A path ← 레지스터
b)	B path 선택	레지스터	:	B path ← 레지스터
c)	ALU 선택	불사용 ALU op.	:	ALU op. 선택
d)	C path 선택	레지스터	:	레지스터 ← C path

(16진 코드)

a)	0	0	0	2	0	:	레지스터 5
b)	1	0	0	4	0	:	레지스터 6
c)	2	0	0	0	1	:	ADD
d)	3	0	0	8	0	:	레지스터 7

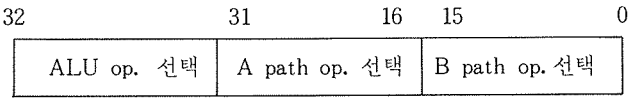
다. 대각구조

이 구조는 수평구조와 수직구조의 trade off 구조인데 마땅한 이름이 없어서 해학적으로 붙여진 이름이며, 이 구조의 마이크로 인스트럭션 word는 일반적으로 4~6 영역으로 이루어지고 그 크기는 48비트 정도가 대부분이다. 이 구조의 장단점은 수평 및 수직구조에서 언급한 항목들의 각각에 대해 그 중간점이 된다. 그림 3 을

바탕으로 앞의 두 구조에 대한 중간구조를 설 계해 보면 다음과 같다.

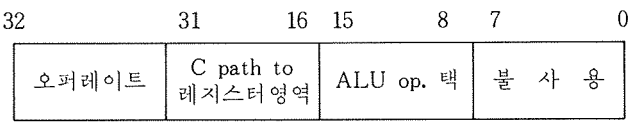
(대각 구조 MI 소양)

a)



: A path ← 레지스터
B path ← 레지스터

b)



: 오퍼레이트 레지스터 ← 결과

(16진 코드)

a) | 0 | 0 0 2 0 | 0 0 4 0 | : 레지스터 5, 6

b) | 1 | 0 0 8 0 | 0 0 2 0 | : 레지스터 7

4. Microprogram 설계

마이크로 프로그램을 설계한다는 것은 마이크로 인스트럭션 소양을 기본으로 하여 어떤 목적의 기능을 구현하는 마이크로 인스트럭션의 sequence를 만드는 것인데 불행하게도 여기에는 High level language프로그램을 작성하는 것과 달리 설계자가 알아야 하는 많은 하드웨어적 제약이 있는데 이를 완전히 이해하여야 한다.

마이크로 프로그램을 설계하는 단계는 일반적으로 다음과 같다.

첫째, 어떤 종류의 제어를 위한 것인가 파악하고, 둘째, 설계된 마이크로 인스트럭션 소양을 숙지하고, 세째, 거기에 관련되는 모든 시스템 dependency를 분석하고, 네째, 주어진 기능을 이행하기 위해 적합한 마이크로 프로그램 구조를 설계하고, 다섯째, 그 구조의 각 단계에서 이행해야 하는 세부 기능을 정하고, 여섯째, 각 단계별로 부여된 기능을 이행하는 마이크로 인스트럭션의 sequence를 설계한다.

마이크로 프로그램은 그 기능이 어떤 종류의 제어를 위한 것이냐에 따라서 설계 방법이 아주 다른데 마이크로 프로그래밍의 응용 중 대표적인 머신 랭귀지 인터프리테이션을 위한 에뮬레이터에 대해 그 구조를 간략히 기술하면, 일반적인 컴퓨터는 그 머신 랭귀지가 General, Instruction, Decimal Instruction, Floating Point Instruction, Input Output Instruction, Control Instruction으로 그룹지어지며 이들을

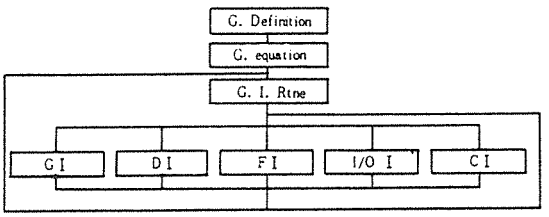


그림 4. 인스트럭션 종류별 구조

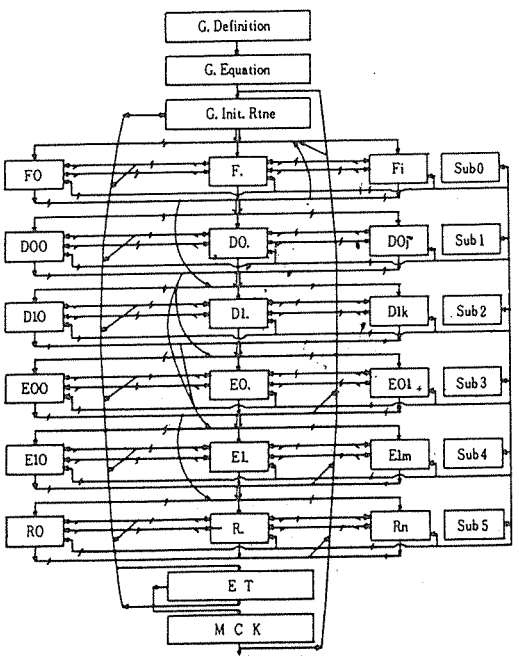


그림 5. 인스트럭션 수행단계별 구조

인터프리테이션하는 단계는, 크게 주기억 장치에서 이들의 각각을 읽어오는 Fetch 단계, 읽어온 머신 인스트럭션의 의미를 분석하는 Decode 단계, 인스트럭션에 명시된대로 시행하는 Execution 단계로 나누어지고 비정상 상태의 원활한 처리를 위해 Exception 처리 단계 및 기계 고장진단 단계 등이 덧붙여질 수 있으며, 이들 각 단계는 인터프리테이션 되어질 전 머신 인스트럭션의 각 경우를 적절히 처리하기 위한 부분들로 구성되어진다.

그림 4와 5는 각각 인스트럭션 종류별 구조와 시행단계별 구조이다.

인스트럭션 시행단계별 구조에서 각 루틴의 특성은 다음과 같다.

- G. Definition : 자원 정의 및 할당
- G. Equation : 상수와 심볼의 의미 부여
- G. Init. Rtne : 전 자원의 초기치화
- F0~Fi : IR ← 머신 인스트럭션, 인스트럭션 주소 검정
- D00~D0j : 오퍼런드 1과 기타 필요한 자원 준비
- D10~D1k : 오퍼런드 2와 기타 필요한 자원 준비
- E00~E01 : 부여된 기능 실행, 바이트 조정
- E10~E1m : E00~E01의 보조
- R0~Rn : 새로운 인스트럭션 추가 및 특별 인스트럭션을 위한 영역
- ET : 비정상 상태 처리
- MCK : 기계 고장상태 처리
- Sub0~Sub5 : 반복 루틴

Ⅲ. 국내외의 전망

마이크로 프로그래밍의 발전단계 중 제4세대에 속하는 현재, 거의 모든 컴퓨터가 부분적 혹은 전적인 마이크로 프로그래머 제어를 취하고 있으며, 조만간 한 표준 하드웨어상에 여러 OS가 동시 상주하는 Multiemulator 머신과 시스템의 기능이 여러차원에서 수행될 수 있는 Integrated hardware, firmware, software 시스템의 개발이 기대되고 있다.

마이크로 프로그래밍에 대한 선진국의 수준은 이렇게 완숙단계에 접어들고 있는데 우리나라는 아직 그 기본단계에 있으며, 단순한 교육목적을 제외한 이에 대한 연구는 일부학계의 마이크로 프로그래밍의 최적화에 대한 연구와 한국전자통신연구소에서 수행하고 있는 마이크로 프로그래밍 기법을 이용한 32Bit VM Machine 개발 프로젝트를 들 수 있는데 아직까지 상품화된 컴퓨터는 없으며, 현재 생산하고 있는 컴퓨터들은 거의 대부분이 마이크로 차원 및 그 이하의 컴퓨터를 구성하고 있는 단계이므로 마이크로 프로그래밍의 장점들인 체계적 설계, 아키텍춰 수정의 용이성, 제어과정 설계의 융통성, 모델 상호간의 일치성 등의 필요성을 실감치 못하고 있다. 그러나 조만간 컴퓨터의 구성 단계를 넘어서 점차 진정한 의미의 개발이 요구되어질 것이며, 그 시점에서부터 중형이상 뿐만 아니라 소형에 이르기까지도 마이크로 프로그래밍은 기본기술로 이용될 것이다.

참고문헌

(1) Samir S. Husson, Microprogramming Principles and Practices, Prentice-Hall Press, 1970

(2) Ashok K. Agrawala, Foundation of Microprogramming, Academic Press, 1976

(3) Marco Mezzalama and Paolo Prietto, A Hierarchial Description Model for Microcode, IEEE, 1983

(4) Dilip K. Banerji, Elements of Microprogramming, Prentice-Hall Press, 1982

(5) Ralph Grishman and Su Bogong, A Preliminary Evaluaton of Trace Scheduling for Global Microcode Compaction, IEEE, 1983

(6) Joseph A. Fisher, Trace Scheduling : A Technique for Global Microcode Compaction, IEEE, 1981

(7) Mario Tokoro, Optimization of Microprogram, IEEE, 1981

(8) David Landskov, Local Microcode Compaction Technique, Computing Survey, Sept. 1980

(9) Subrata Dasgupta, Some Aspects of High Level Microprogramming Surveys, Vol. 12, No. 3, Sept, 1980.