비트 슬라이스 소자를 이용한 마이크로프로세서 설계

신 봉 희

(시립 인천전문대학 전자계산과)

요 약

비트 슬라이스 소자들을 이용하여 마이크로프로세서를 설계할 때 마이크로프로그램 제어 방식을 채택한다. 이때 설계자는 효과적이고 경제적인 마이크로프로그램 개발 환경을 필요로한다. 마이크로프로그래밍 비용을 최소화 시키는 체계적인 마이크로프로그램 개발 환경을 마련하기 위해서, 본 논문에서는 마이크로프로그래밍 과정을 단계별로 고찰하여 마이크로프로그램 특성상 하드웨어와 밀접한 관계를 유지하며 효과적인 마이크로코드를 생성하는 마이크로명령어 정의어를 제안하였다.

Ⅰ. 서 론

디지털 시스템 개발시 투자에대한 경제성이 야기되고, 설계의 다양성이 기대되어 비트-슬라이스 개념이 등장하였다. 전문적인 기능 모듈로 컴퓨터의 각부분을 분활하고, 데이터를 몇 개의 비트 단위로 처리하는 개념이다.

대부분의 상용 마이크로프로세서는 내부구조, 명령어 길이(wordlength) 그리고 명령어 조합등이 이미 제조회사에 의해 고정되어 있다. 이에 비하여 비트-슬라이스 소자로 설계된 마이크로프로세서는 기능별로 제어장치, 자료처리장치들로 각각 분리된 칩으로 구성된다. 또한 동일한 여러개의 비트-슬라이스 소자를 단순 병렬 접속함으로 임의의 명령어 길이를 정의 할 수 있고, 연산능력을 확장시킬수 있다. 아울러 제어부의 마이크로프로그램 메모리의 내용을 변경할 수 있어 사용자가 시스템의 명령어조합을 정의할 수 있고 독자적인 시스템의 구성이 가능하다. 그러므로 특수 목적에 알맞은 효율적인 구성이 가능하며 또한 확장, 변경 및 하드웨어의 변경이 예상되는 부분에서도 전체 하드웨어의 재 설계없이 구성할 수 있다. 또한 마이크로프로그램에 의한 제어 방법을 사용하므로 설계자가 시스템의 제어 장치를 설계할 때 규칙성(structured and regular method)이 제공되며 시스템의 하드웨어를 완성한 후에도 마이크로 메모리의 내용을 변

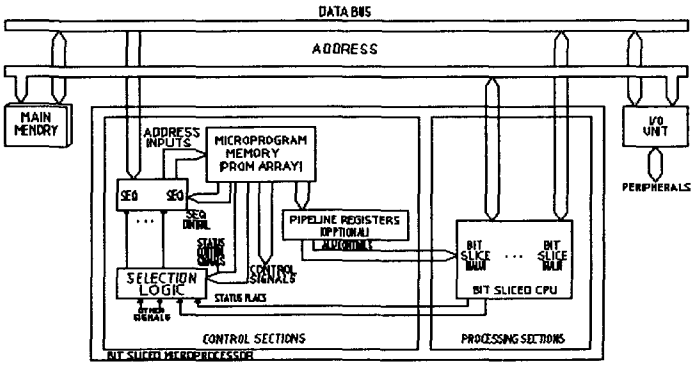
경하므로서 매크로명령의 변경이 가능하다. 때문에 시스템의 성능이나 기능을 쉽게 변경 할 수 있고, 시스템의 구조나 처리능력을 최적화 시킬수 있고, 시스템의 기능이나 성능을 변경, 개선 할수 있으므로 설계 상의 많은 융통성을 갖게 된다. 그러나 시스템 설계자는 마이크로프로그램을 직접 작성하여야 하는 문제점을 갖고있다. [1-8]

본 논문에서는 마이크로프로그램 명령어를 정의할수 있는 언어를 개발하여 마이크로프로그램 개발 환경을 마련하였다.

II. 본 론

1. 마이크로프로세서 구성

비트-슬라이스 소자들을 이용하여 마이크로프로세서를 구성할 경우 <그림 1>과 같이 제어장치와 자료처리장치로 분리하여 구성한다. 설계된 마이크로프로세서는 AMD(Advanced Micro Device Inc.)의 Am2900 Bit-slice 계열 소자를 중심으로 자료처리부와 제어부로 설계하였다.

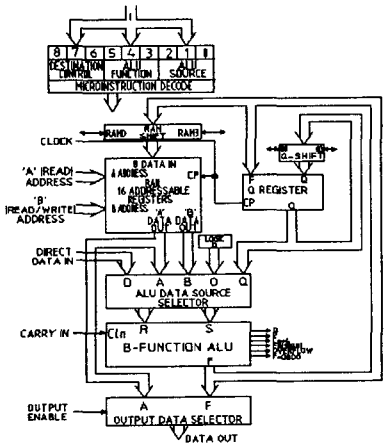


<그림 1> 비트 슬라이스 소자로 설계한 마이크로프로세서

1) 자료처리부

자료 처리 장치는 산술 및 논리 연산이 이루어지며 여러개의 자료 처리장치를 병렬로 연결하여 임의의 명령어 길이를 구성할 수 있다. 자료처리장치는 동일한 기능을 같고 있는 4개의 비트-슬라이스 RALU(Am2901)를 중심으로 구성되여 있다.

Am2901은 4비트 단위의 16word를 갖는 2-port RAM, Input MUX, Output MUX, ALU, Q레지스터, Status Line 등을 포함하고 있으며, <그림 2>는 RALU Am2901의 내부블럭도이다.



<그림 2> RALU Am2901 내부 블럭도.

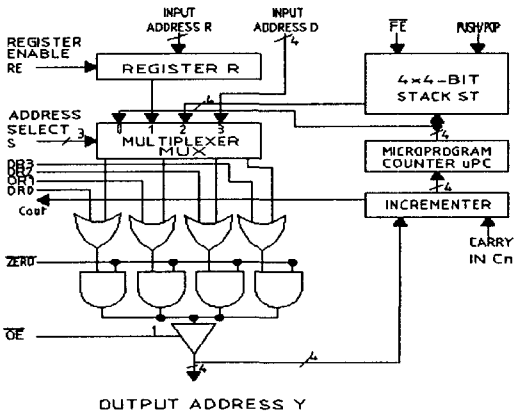
2-port RAM은 입출력이 모두 가능한 A port와, 입력만이 가능한 B port가 있으며 한 마이크로 싸이클에서 동시에 억세스가 된다. A,B port의 주소는 IR에 있는 매크로 명령어의 오퍼랜드나 마이크로명령어의 A,B주소 필드에서 공급 받는다. Input MUX는 ALU가 스크래치패드 레지스터의 A,B port나 데이터 입력 버스로부터 데이터를 선택하게 하며, Output MUX는 ALU의 출력이나 스크래치패드 레지스터의 값을 출력하게 한다. 그리고 Q 레지스터는 Double shift나 곱셈을 할 때 Double length를 위하여 사용된다. Status and shift Control Unit Am2904는 ALU/register Slice의 기능을 보강할 수 있는 Shift control, Carry control, status control을 갖고 있으며 그 구조와 명령 code는 Am2900 계열에 맞게 설계되어 있다.

2) 제어부

마이크로프로그램 제어기는 매크로명령어를 디코우드하여, 마이크로프로그램이 계속 수행될 수 있도록 다음에 수행할 마이크로명령어의 주소를 결정한다. 하나의 매크로명령어에 대하여 하나 이상의 마이크로명령어를 수행 하여야 할 경우에는 보통의 프로그램의 경우와 같이 바로 다음주소에 기억시켜 놓은 마이크로명령어를 수행 하도록 할

수도 있으며 마이크로 명령어 내부에 다음에 수행할 마이크로명령어의 주소를 포함할 수 있다.

Am2909/11은 이러한 기능이 있으며 각 소자는 4비트의 마이크로 word를 갖고 있으므로 4비트 단위로 확장할 수 있다. 그리고 마이크로 서브루틴을 위해 4×4의 스택 포인터를 가지고 있으며 다음에 수행될 주소는 Direct input, Address holding register, Stack pointer, Microprogram connter 중에서 선택된다. Am2909는 Y버스상에 OR게이트가 있어서 16-way Rranch Control Unit인 Am29803과 조합하여 multiway branch를 할 때 유용하다. <그림 3>은 Am2909//11 내부 블럭도이다.



<그림 3> Am2909/11 내부 블록도

2. 마이크로프로그램

마이크로프로그램에 의한 제어부의 구성을 하드웨어 자원과 기본연산을 근거로 시스템 설계자가 직접 마이크로프로그램을 설계한다. 일련의 기본연산들로 복합연산들이

정의되며 복합연산들은 제어부의 기억장치에 저장된후 운영된다. 복합연산을 구성하는 각각의 기본연산들을 마이크로명령이라하며, 마이크로명령들을 포함하고 있는 기억영역이 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.

1) 기본 연산 정의

기본 연산은 자신의 하드웨어 구성에 근거하여 정의된다. 기본 연산들은 매크로명령에 해당하는 모든 마이크로루틴을 작성할 수 있도록 타당성이 있어야 한다. 하드웨어 자원과 데이타에 대해 동작하는 기본 연산은 레지스터간 데이터 이송 연산(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 분류한다. 기본 연산들에 대해 <표 1>과 같이 분류하고 명칭을 부여한다.

<표 1> 기본연산과 명칭

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

2) 마이크로명령어 정의

마이크로명령어는 시스템 하드웨어 자원을 제어하기 위한 기본 연산들을 수행되도록 하는 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. 마이크로 명령어의 구성은 설계자에 따라 매우 다양하지만 대체로 수행되어야 할 마이크로 오퍼레이션을 제어하는 부분과 다음에 수행될 마이크로 명령어의 주소 또는 소스를 선택하는 부분으로 되어 있다. 비트 슬라이스 마이크로명령어 형식은 자료 처리 장치에서 수행될 마이크로명령과 마이크로명령의 동작 순서를 제어하는 부분으로 나뉘어 설계하였다. 마이크로명령어의 길이는 총 39비트로 구성하였고, 80비트 까지 확장하여 사용할 수 있도록 마이크로프로그램 메모리와 파이프라인 레지스터로 구성하였다. 마이크로명령은 자료처리를 위해 20비트 할당되었고, 제어용으로 19비트 할당하였다. <표 2>는 마이크로명령어 형식이다.

<표 2> 마이크로명령어

FIELD	Bit 수	CONTENTS
FIELD 1	4	Am2901 scratched pad register A address
FIELD 2	4	Am2901 scratched pad register B address
FIELD 3	3	Am2901 destination
FIELD 4	3	Am2901 function
FIELD 5	3	Am2901 source
FIELD 6	1	Am2901 carry control
FIELD 7	1	DIR control
FIELD 8	1	DOR control
FIELD 9	1	MAR Control
FIELD 10	1	MDR control
FIELD 11	1	IR control
FIELD 12	4	Am2909 OR gate
FIELD 13	4	Am29811 next control
FIELD 14	4	Am2922 condition code
FIELD 15	4	Am29803 multiway branch

3. 마이크로 명령 형식 정의어 설계

마이크로명령어 형식 정의어는 마이크로코드를 생성하기 위한 언어로써 마이크로명령어 형식 정의부, 마이크로코드 생성부와 마이크로코드 연산 및 제어부를 모듈별로 분리하여 설계하였다.

마이크로명령어 형식 정의부에서는 하드웨어의 기본 연산을 바탕으로 마이크로명령어 형식을 정의한다. 마이크로명령어는 주 마이크로명령어와 부 마이크로명령어로 나뉘어 정의되게 설계하였으므로 수직 또는 수평 마이크로코드 생성에 대응한다. 주와 부 마이크로명령어 형식 정의는 각각 머리부와 몸체부로 나뉘어 정의된다. 머리부는 마이크로명령어 명칭, 총길이와 필드의 갯수를 정의하고, 몸체부에서는 각 필드에 대한 명칭과 길이 값들을 정의한다. 마이크로명령어 형식 정의어 구조는 <그림 4>와 같다.

```
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 정의어 구조

4. 마이크로코드 생성

마이크로코드 생성부에서는 일단 마이크로프로그래머가 마이크로명령어 형식 정의어를 사용하여 마이크로명령어 형식을 정의하면 마이크로명령어 변역기가 이들을 번역하여 마이크로명령어의 형식 정의에 따라 마이크로명령어 심볼테이블을 구성하고, 이차로 마이크로명령들로 구성된 마이크로루틴이 입력된다. 마이크로명령들이 심볼테이블에 부마이크로명령어로써 등록되여 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.

```
AMD2900 : INSTRUCTION LENGTH(39) ; Microinstruction name & length
            FIELDS(15)              ; The number of field
            { F1 : LENGTH(4)         ; Am2901 scratched pad register A
address
              F2 : LENGTH(4)         ; Am2901 scratched pad register B
address
              F3 : LENGTH(3)         ; Am2901 destination
              F4 : LENGTH(3)         ; Am2901 function
              F5 : LENGTH(3)         ; Am2901 source
              F6 : LENGTH(1)         ; Am2901 carry control
              F7 : LENGTH(1)         ; DIR control
              F8 : LENGTH(1)         ; DOR control
              F9 : LENGTH(1)         ; MAR control
              F10: LENGTH(1)         ; MDR control
              F11: LENGTH(1)         ; IR controll
              F12: LENGTH(4)         ; Am2909 OR gate
              F13: LENGTH(4)         ; A29811 next address
              F14: LENGTH(4)         ; Am2922 condition code
              F15: LENGTH(4)         ; Am29803 multiway branch
          }
     FETCH : INSTRUCTION LENGTH(26)
             FIELDS(11)
           { F2 : LENGTH(4)
                  VALUES(15)
             F3 : LENGTH(3)
                  VALUES(3)
             F4 : LENGTH(3)
```

```
        VALUES(0)
F5 : LENGTH(3)
        VALUES(3)
F6 : LENGTH(1)
        VALUES(0)
F7 : LENGTH(1)
        VALUES(1)
F8 : LENGTH(1)
        VALUES(1)
F9 : LENGTH(1)
        VALUES(1)
F10 : LENGTH(1)
        VALUES(1)
F11 : LENGTH(1)
        VALUES(1)
F13 : LENGTH(4)
        VALUES(14)
}
```

III. 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 할 수 있다. 최소한의 비용과 노력으로 마이크로프로그램을 작성할 수 있어 체계적이며 효율적인 마이크로프로그래밍 개발 환경의 구성이 가능하다.

참 고 문 헌

[1] Myers G. J, Digital System Design with LSI Bit-Slice Logic,John Wiley

Sons, New York, pp. 338, 1980.

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

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

[4] 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.

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

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

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

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