論文 91-28B-7-5

마이크로명령어 기술언어를 사용한 마이크로코드 자동생성 시스템

(An Automatic Microcode Generation System Using a Microinstruction Description Language)

李 相 靜* 曺 永 勳** 林 寅 七**

(Sang Jeong Lee, Young Hoon Cho, and In Chil Lim)

要 約

본 논문에서는 마이크로아키텍쳐 기술언어인 MDL(microinstruction description language) 언어를 개발 사용한 머신 독립적인 마이크로코드 자동생성 시스템을 제안한다. 제안된 MDL 기술언어는 다양한 마이크로아키텍쳐의 하드웨어 요소 및 마이크로오퍼레이션의 오퍼랜드 선택을 논리적으로 기술 가능한 C언어 형태의 고급 기술언어로 설계된다. 마이크로코드 자동생성 시스템은 MDL 기술언어로써 각 대상 마이크로아키텍쳐의 구조정보를 기술하고, 고급 마이크로프로그래밍 언어인 HLML-C(high level microprogramming language C) 언어의 중간언어를 마이크로오퍼레이션 동작정보로써 입력 받아 마이크로코드를 자동생성한다. 제안된 시스템은 SUN 워크스테이션(4.3BSD) 상에서 YACC와 C언어로 구현한다.

Abstract

This paper proposes a machine independent automatic microcode generation system using a microinstruction description language, MDL. The MDL, which has similar structure to C language, is a high-level microarchitecture description language. It defines the hardware elements and the operand selection of microoperations. The proposed system generates microcode automatically by describing the structure information of a target microarchitecture and accepting the behavioral information of microoperations which are generated as a intermediate language from HLML-C. This proposed system is implemented with C language and YACC on a SUN workstation (4.3 BSD).

I. 서 론

마이크로프로그래밍을 이용한 제어방식은 프로그램을 이용한 소프트웨어적인 제어가 가능하고 수정(flexiblity) 및 확장(extensibility)이 용이하면서 적은 비용으로 많은 기능을 수행할 수 있는 장점을 가지고 있어 현재 컴퓨터 시스템의 제어에 널리 사용되고 있다. 마이크로프로그램은 프로세서 내부의 하드웨어 동작을 직접 제어하는 낮은 수준의 프로그램이기 때문에 프로그램의 작성과 유지가 어렵고, 특히 마이크로오퍼레이션의 병렬처리를 위하여 수평 마이

*正會員, 順天鄕大學校 電算學科
(Dept. of Compt. Sci., Soonchunhyang Univ.)
**正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)
接受日字: 1991年 4月 18日
(※ 이 논문은 1990년도 문교부지원 한국학술진흥재단의 지방대 육성 학술연구 조성비에 의하여 연구되었음.)

크로 명령어 형식(horizontal microinstruction format)을 갖는 프로세서가 보편화되고 마이크로프로그램의 규모가 복잡해져서 기존의 마이크로어셈블리 언어나 마이크로코드로써 마이크로프로그램의 작성은 비능률적이고 어렵게 되었다. 따라서 최근에는 고급 마이크로프로그래밍 언어(high level microprograming language, HLML)를 사용하여 마이크로프로그램을 작성하는 도구(tool), 특히 마이크로코드의 자동생성을 위한 표준 소프트웨어 도구(standard software tool)에 대한 연구개발을 집중하고 있다.

HLML을 사용하여 마이크로코드를 자동생성하는 경우 고급언어의 원시 마이크로프로그램으로 부터 대상머신(target microarchitecture)의 마이크로코드를 머신 독립적으로 자동생성해야 하는 retargetability 문제가 대두된다. 즉, HLML의 컴파일러로 부터 생성된 중간언어가 어느 특정 대상머신에 귀속됨이 없이 다양한 특성을 갖는 각 대상머신에 대하여 효율적으로 마이크로코드를 자동생성하는 시스템의 개발이 급선무가 되어야 한다. 따라서 HLML 언어의 개발과 더불어 다양한 마이크로아키텍쳐에 대한 정보를 기술하여 마이크로코드 생성을 시도하는 연구가 활발히 진행중이다. 대표적인 연구로서는 Sint의 MIDL[1], Balakrishnau의 MGS[4], Gieser의 HLL[2], Hwang의 MDSS[3]가 있다. MIDL, MGS, MDSS의 경우는 머신 독립적인 HLML과의 연결을 고려하지 않고, 단순히 저수준의 마이크로오퍼레이션상의 마이크로프로그램을 입력으로 받아 마이크로코드를 생성하였고, HLL의 경우는 머신 독립적인 HLML과 함께 하드웨어 기술을 설계하였으나, 구현되지 못하였다.

본 논문에서는 HLML-C[9] 언어로 작성되어 생성된 중간언어로 부터 각 대상머신의 마이크로코드를 머신 독립적으로 자동생성하는 마이크로코드 자동생성 시스템을 제안한다. 즉, C언어 스타일의 고급언어 형태로 쉽게 기술되는 마이크로명령어 기술언어(microinstruction description language, MDL)를 개발 사용하여 마이크로프로그램으로 제어되는 아키텍쳐의 모든 하드웨어 요소 동작 및 오퍼랜드의 선택을 논리적으로 기술한다. 기술된 MDL은 각 마이크로명령어의 가능한 모든 필드형식을 고려하여 대상머신의 마이크로오퍼레이션 동작에 대한 정보를 제공함으로써 마이크로코드 자동생성시 HLML-C 중간언어와 각 대상머신의 마이크로코드 사이의 인터페이스 역할을 한다.

제안된 시스템은 SUN 워크스테이션(4.3BSD) 상에서 YACC와 C언어로 구현하여 생성된 마이크로코드를 검증함으로써 유효성을 입증한다.

Ⅱ. HLML-C 마이크로코드 자동생성 시스템

HLML-C 마이크로코드 자동생성 시스템의 전체 구성도는 그림1과 같다. HLML-C 언어는 C 언어와 유사한 구조를 갖는 머신 독립적인 고급 마이크로프로그래밍 언어로서 마이크로프로그램 작성 시 머신 독립성을 유지하면서 대상머신에 종속적인 특수한 오퍼레이션의 확장 적용과 각 대상머신의 하드웨어 자원(hardware resource)의 표현이 가능한 언어이다.[9][10]

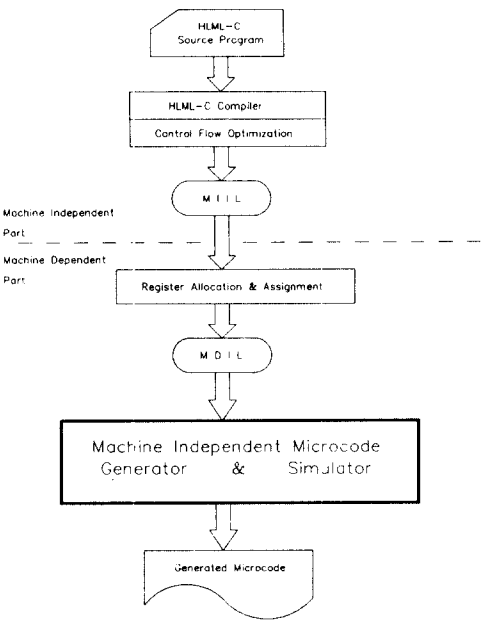


그림 1. HLML-C 마이크로코드 자동생성 시스템
Fig. 1. HLML-C automatic microcode generation system.

HLML-C 컴파일러는 HLML-C 언어로 작성된 원시 마이크로프로그램을 추상머신(abstract machine)상에서 동작하도록 정의된 3주소 코드(3-address code)의 머신 독립적인 중간언어 MIIL(machine indepent intermediate language)를 생성한다. MIIL 생성후 대상머신의 가용한 범용 레지스터 수와 ALU 갯수 등을 입력으로 받아, MIIL의 각 오퍼랜드의 변수들을 범용 레지스터에 할당하여 머신 종속적인 중간언어 MDIL(machine dependent intermediate language)를 생성한다. 그림2는 HLML-C 언어로 작성된 Fibonacci series의 마이크로프로그램의 예이고, 그림3은 6개의 가용한 범용 레지스터를 할당한 후 MDIL 생성 예이다.

```
mainmem 16 MM[1000]
memory 16 A:200,B:100:   /* A contains initial address
                            for Fibonacci, B contains series count */
microprogram Fibonacci_series;
{ register 16 x,y,current,addr,counter;

  addr = @A:    /* local address and counter */
  counter = B;

  x=1;  /* initialize */
  y=1;

  MM[addr] = x;
  MM[++addr] = y;

  for (counter -= 2; counter != 0; --counter) {
        current = x+y;
        MM[++addr] = current;
        x=y;
        y=current;
  }
}
```

그림 2. Fibonacci series HLML-C 마이크로프로그램의 예

Fig. 2. Example of Fibonacci series microprogram written by HLML-C.

```
                ENTRY       Fibonacci_series
                MOVE        AC      200
                MOVE        R3      AC
                LOAD        MDR     100
                MOVE        R4      MDR
                MOVE        R1      1
                MOVE        R2      1
                STORE       R1      R3
                INC         R3
                STORE       R2      R3
                SUB         R4      R4      2
L13 :
                CJMPNE      R4      0       L17
                JUMP                        L24
L15 :
                DEC         R4
                JUMP                        L13
L17 :
                ADD         AC      R1      R2
                MOVE        R5      AC
                INC         R3
                STORE       R5      R3
                MOVE        R1      R2
                MOVE        R2      R5
                JUMP                        L15
L24 :
                EXIT        Fibonacci_series
```

그림 3. Fibonacci series 마이크로프로그램에 대한 MDIL 생성 예

Fig. 3. Eaxample of MDIL generation for a Fibonacci series microprogram.

Ⅲ. 마이크로코드 생성기

머신 독립적인 마이크로코드 생성기는 주요부분으로서 매크로프로세서, MDL 구문분석기 및 마이크로코드 어셈블러, 마이크로오퍼레이션 시뮬레이터로 구성된다(그림4).

1. 매크로프로세서

매크로 정의는 MDIL 중간언어의 각 명령어들에 대하여 확장되는 심볼릭 마이크로오퍼레이션을 정의한다. 원시 HLML-C 마이크로프로그램으로 부터 생성된 MDIL의 각 명령어 문장이 정의된 각 매크로 정의를 호출하여 대상머신의 심볼릭 마이크로오퍼레이션으로 확장된다. 그림5(a), (b)는 간략화 된 매크로 정의 구문형식과 정의 예이다.

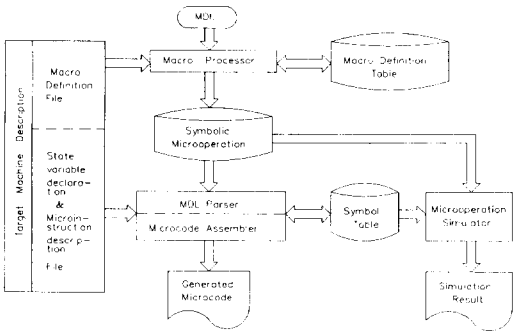


그림 4. 머신 독립적인 마이크로코드 자동생성기

Fig. 4. Machine independent automatic microcode generator.

2. MDL 구문분석기 및 어셈블러

MDL 구문분석기는 대상머신의 구조정보와 마이크로명령어 형식, 필드배치와 제어조건을 기술한 파일을 입력으로 받아 심볼 테이블을 구성한다. 심볼 테이블은 마이크로코드 어셈블러와 시뮬레이터에 대하여 대상머신의 구조정보를 제공한다. 마이크로코드 어셈블러는 심볼 테이블로 부터 마이크로명령어 형식 및 필드배치와 제어조건에 관한 정보를 제공받

```
<macro_definition> :  macro <identifier> <argument_list> <macro_body>
<argument_list> :  ( <argument> )*
<argument> :  '&' <identifier>
<macro_body> :  '{' ( <statement> )+ '}'
<statement> :  if '(' <condition> ')' <statement> [ else <statement> ]
             | <microoperations> ';'
             | <macro_call_statement> ';'
             | '{' ( <statement> )+ '}'
             | ';'
<microoperations> :  <microoperation> ( ',' <microoperation> )*
<microoperation> :  <operand> '=' <operand>
                  | <operand> '=' <unary_op> <operand>
                  | <operand> '=' <operand> <binary_op> <operand>
                  | read <operand> <operand>
                  | write <operand> <operand>
                  | <branch> <operand>
<branch> :  goto | cjmpz | cjmpnz | cjmpp | cjmpn | cjmpo
<operand> :  <identifier> | <argument> | <number>
```

(a)

```
macro LOAD &a &b                /* &a = MM[&b] */
{
        if ( &b == MAR )
                ;
        else {  BBUS = &b;  BLATCH = BBUS;
                MAR = BLATCH;
        }
        if ( &a == MDR )
                read MDR MAR;
        else {  read MDR MAR; MUX = MDR;
                ALU = MUX; SHIFTER = ALU; &a = SHIFTER;
        }
}
```

(b)

그림 5. (a)매크로 정의 구문형식
(b) 매크로 정의 예

Fig. 5. (a) Syntax of macro definition,
(b) Example of macro definition.

아 매크로프로세서에 의해 확장된 심볼릭 마이크로오퍼레이션을 대상머신에서 수행 가능한 마이크로코드로 변환한다.

3. 마이크로오퍼레이션 시뮬레이터

매크로프로세서에서 생성된 심볼릭 마이크로오퍼레이션은 시뮬레이터를 통해서 검증된다. 시뮬레이션의 수행결과는 대상머신의 각 하드웨어 자원(상태변수)에 저장되며, 이러한 상태변수들의 값을 추적함으로써 마이크로오퍼레이션의 수행 결과를 검증 할 수 있다. 즉, 각 대상머신에 대하여 생성된 마이크로오퍼레이션의 수행 동작을 시뮬레이션함으로써, 특정 대상머신의 마이크로레벨 동작을 확인 검증한다.

Ⅳ. 마이크로명령어 기술언어

마이크로명령어 기술언어인 MDL은 메모리, 레지스터, 서브레지스터, 버스, 버퍼 등 하드웨어 자원(resource)을 나타내는 상태변수를 선언하고 각 마이크로명령어 형식에 따른 필드 배치와 상태변수와의 관계 및 그 제어 조건을 기술한다.

1. 상태변수 선언

상태변수는 크게 레지스터형과 메모리 형으로 구분된다. 레지스터 형의 변수는 레지스터, 서브레지스터, 버스, 래치, 버퍼, 제어선과 같은 하드웨어 대상을 나타내고 메모리 형의 상태변수는 메모리를 선언한 변수로서 레지스터 형의 배열과 유사하다. 상태변수 선언 시 비트 폭, 워드 폭 등도 선언된다. 선언된 상태변수는 마이크로오퍼레이션의 오퍼랜드로서 사용된다. (그림6)

```
<resource_declararions> :  ( <declaration> ';' )+
<declaration> :  <register_declaration>  | <subregister_declaration>
               | <memory_declaration>    | <bus_declaration>
               | <input_declaration>     | <output_declaration>
<register_declaration> :  register <identifier> <bit_range>
                          { ',' <bit_range> )*
<subregister_declaration> :  subreg <sub_id> ( ',' <sub_id> )*
<sub_id> :  <identifier> <bit_range> '=' <identifier> <bit_range>
<memory_declaration> :  memory <mem_id> ( ',' <mem_id> )*
<mem_id> :  <identifier> <word_range> <bit_range>
<bus_declaration> :  bus <identifier> <bit_range>
                     { ',' <identifier> <bit_range> )*
<input_declaration> :  input <identifier> <bit_range>
                       { ',' <identifier> <bit_range> )*
<output_declaration> :  output <identifier> <bit_range>
                        { ',' <identifier> <bit_range> )*
<bit_range> :  { '<' <number> ':' <number> '>' }?
             | { '<' <number> '>' }?
<word_range> :  '[' <number> ':' <number> ']'
```

(a)

```
register        MAR<15:0>, HL<15:0>, FLAG<3:0>;
subreg          H<7:0> = HL<15:8>, L<7:0> = HL<7:0>,
                CA = FLAG<3>, N = FLAG<2>;
memory          MM[1023:0]<15:0>, CM[525:0]<68:0>;
bus             ABUS<15:0>, CBUS<15:0>, DBUS<15:0>;
input           IN;
output          OUT;
```

(b)

그림 6. (a)상태변수 선언 구문형식
(b) 상태변수 선언 예

Fig. 6. (a) Syntax of state variable declaration,
(b) Example of state variable declaration.

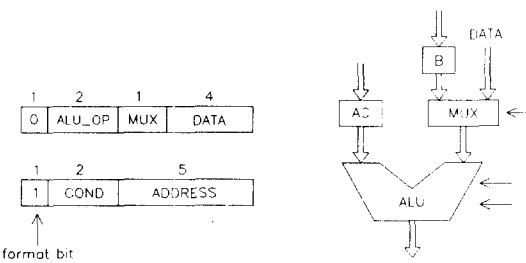
2. 마이크로명령어 기술

마이크로명령어 기술은 하드웨어 구조정보를 마이크로명령어의 형식, 필드값에 대응되어 동작하는 마이크로오퍼레이션을 기술한다. 기술되는 마이크로오퍼레이션은 매크로프로세서에서 확장되는 마이크로오퍼레이션을 표현한다. 그림7은 마이크로명령어 기술 구문형식을 나타내고, 그림8(b)는 (a)의 구조에 대하여 마이크로명령어 기술 예를 보여준다.

```
<microinstruction_definition> :  microinstruction '(' <bit_width> ')'
                                 '{' <mi_definitions> '}'
<mi_definitions> :  formats '{' format bit ':' <bit_range> ':'
                    { <format value> }+ '}'
                  | <field_definitions>
<format_value> :  value <number> ':' <field_definitions>
<field_definitions> :  <field_position> <field_value>
<field_position> :  field positions '{' ( <field_map> )+ '}'
<field_map> :  <field_name_id> ':' <bit_range> ';'
<field_value> :  field values '{' ( <value_map> )+ '}'
<value_map> :  <switch_statement>
             | <field_name_id> <timing> ':' <microoperations> ':'
<switch_statement> :  switch '(' <field_name_id> <timing> ')'
                      '{' ( <value_definition> )+ '}'
<timing> :  '[' phase <number> ']'
<value_definition> :  <number_type> ':' <microoperations> ':'
                    | <number_type> ':' <switch_statement>
<number_type> :  <binary> | <hexadecimal> | <octal> | <decimal>
```

그림 7. 마이크로명령어 기술 구문형식

Fig. 7. Syntax of microinstruction description.



(a)

```
microinstruction(8)   /* microinstruction bit width */
{ formats {
        format bit : <7>;  /* format bit position */
        value 0 : field positions {  /* field layout for format 0 */
                        ALU_OP : <6:5>;
                        MUX    : <4>;
                        DATA   : <3:0>;
                  }
                  field values {  /* field values for format 0 */
                        switch (MUX [phase 1]) { /* ALU MUX field */
                                b'0' : MUX = B;
                                b'1' : MUX = DATA;
                        }
                        switch (ALU_OP [phase 2]) {
                                        /* ALU operation field */
                                b'00' : ALU = AC;      /* move */
                                b'01' : ALU = AC + MUX; /* add */
                                b'10' : ALU = AC & MUX; /* and */
                                b'11' : ALU = ~AC;     /* complement */
                        }
                  }
        value 1 : field positions {  /* field layout for format 1 */
                        COND    : <6:5>;
                        ADDRESS : <4:0>;
                  }
                  field values { /* field values for format 1 */
                        switch (COND [phsae 1]) {
                                        /* conditional branch field */
                                b'00' : goto;
                                b'01' : cjmpz;  /* if zero */
                                b'10' : cjmpp;  /* if positive */
                                b'11' : cjmpn;  /* if negative */
                        }
                  }
    }
}
```

(b)

그림 8. (a) 마이크로명령어 형식과 ALU 예
(b) 마이크로명령어 기술 예

Fig. 8. (a) Example of microinstruction format and ALU,
(b) Example of description written by MDL.

마이크로명령어 기술 시 먼저 마이크로명령어의 비트 폭을 기술하고 마이크로명령어의 형식과 각 필드의 배치를 기술한다. 마이크로명령어 형식은 형식을 구분하는 필드의 위치와 값을 기술하고 이에 따른 각 필드의 배치를 기술한다. 필드의 배치 기술은 각 필드의 위치에 대응하는 필드값에 따라 동작하는 심볼릭 마이크로오퍼레이션의 선택을 switch 문을 사용하여 기술한다. 기술된 마이크로명령어는 MDL 구문분석기에 의하여 마이크로아키텍쳐 구조정보를 표현하는 각종 심볼테이블이 구성되며, 매크로프로세서에 의하여 확장된 심볼릭 마이크로오퍼레이션으로 표현된 마이크로프로그램이 구성된 심볼 테이블을 참조하여 최종적인 마이크로코드를 생성한다. 구성되는 심볼 테이블로서는 각 심볼릭 마이크로오퍼레이션과 코드 값을 저장하는 마이크로오퍼레이션 테이블과 마이크로명령어 형식과 필드 배치 정보를 저장하는 마이크로명령어 형식 테이블이 있다. 마이크로오퍼레이션 테이블은 마이크로프로그램의 동작이 기술되는 부분으로 매크로프로세서에 의해 확장되는 심볼릭 마이크로오퍼레이션이 저장된다. 심볼릭 마이크로오퍼레이션 저장 시 MDL 기술언어에 의해 기술된 마이크로명령어 형식, 필드형식, 필드값을 연결리스트 구조로 가리킨다. 또 각 심볼릭 마이크로오퍼레이션의 오퍼랜드는 상태변수 선언으로 기술되는 자원 테이블을 가리키는 포인터로 저장된다. 마이크로명령어 형식 테이블은 MDL 기술언어에 의해 기술된 각 마이크로오퍼레이션에 대응되는 마이크로명령어의 필드형식, 필드값 등이 저장된다. 자원 테이블은 상태변수 선언시 구성되는 테이블로써 선언된 각 하드웨어 자원의 속성이 저장된다. 그림9는 각 심볼 테이블 간의 자료 구조관계를 보여주는 그림이다. 마이크로코드 어셈블러는 매크로프로세서에 의해 생성된 심볼릭 마이크로오퍼레이션을 입력으로 받아 마이크로오퍼레이션 테이블을 검색한 후 연결리스트 구조로 연결된 자원 테이블과 마이크로명령어 형식 테이블을 참조하여 마이크로코드를 생성한다. 다중위상(polyphase)을 갖는 마이크로아키텍쳐의 경우에는 확장된 마이크로오퍼레이션의 위상(phase) 순서에 따라 마이크로오퍼레이션의 각 필드 값을 어셈블하여 마이크로코드를 생성한다.

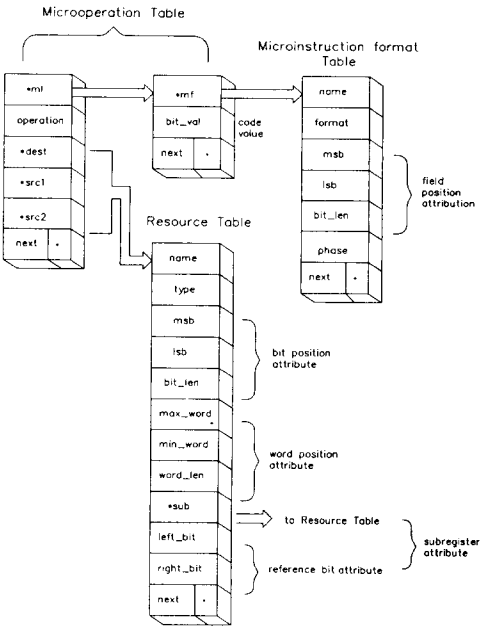


그림 9. 심볼 테이블의 자료구조
Fig. 9. Data structure of symbol table.

Ⅵ. 적용 예

제안된 시스템은 SUN4 SPARC workstation (UNIX 4.3 BSD)에서 구문 생성기(parser generator) YACC와 어휘 분석기(lexical analyzer) LEX, 그리고 C언어를 사용하여 구현하였다. 매크로프로세서와 시뮬레이터는 각각 1100, 1500 스텝의 YACC와 C언어로써 작성되었고, MDL 구문분석기와 마이크로어셈블러는 1200스텝의 YACC와 LEX, C언어로 프로그램하여 각 과정의 수행된 결과가 다른 과정의 입력이 되도록 처리하여 마이크로코드 자동생성 시스템을 도구화 하였다.

그림10은 적용된 예제 대상머신의 구성도를 나타낸 그림이다. 예제 대상머신은 16개의 범용 레지스터, AMD 2903 비트 슬라이스 칩과 유사한 구조의 ALU, 256워드 X 32비트의 제어기억장치등으로 구성된 16비트 머신으로서 4개의 다중위상(polyphase)를 갖고 마이크로명령어가 수행된다.

그림11은 예제 대상머신에 대한 매크로 정의와 MDL 기술이고, 그림12는 예제 대상머신에 대하여 그림3의 Fibonacci MDIL에 대한 매크로 확장된 심볼릭 마이크로오퍼레이션과 자동생성된 마이크로코드를 보인 그림이다.

Ⅶ. 결 론

본 논문에서는 MDL 기술언어를 사용하여 다양한 대상머신에 머신 독립적으로 적용이 가능한 마이크로코드 자동생성 시스템을 개발 제안하였다. 즉, H-LML-C 마이크로프로그램으로부터 생성된 각 MDIL

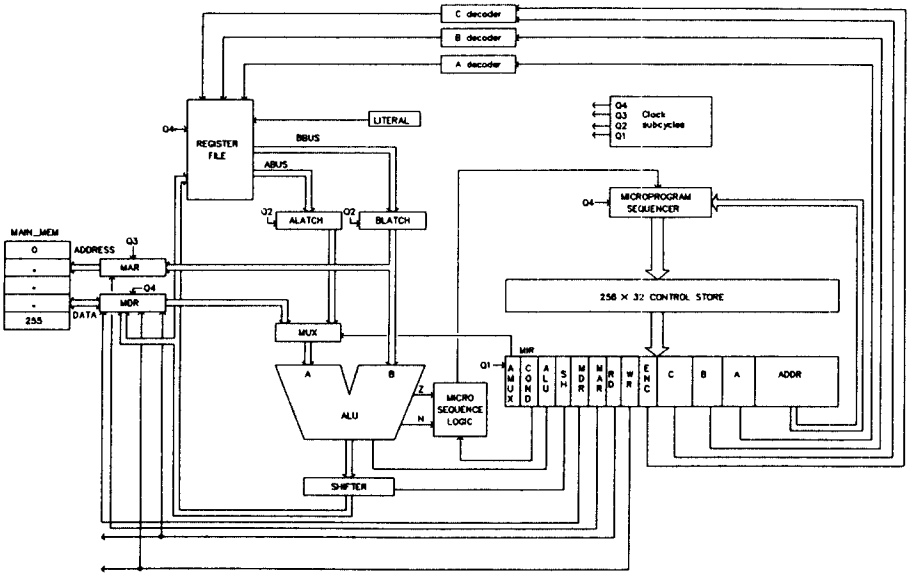


그림10. 예제 대상머신
Fig. 10. Example microarchitceture.

```
register  PC<15:0>, AC<15:0>, SP<15:0>, IR<15:0>, TIR<15:0>,
          TMP<15:0>, AMASK<15:0>, SMASK<15:0>, R1<15:0>, R2<15:0>,
          R3<15:0>, R4<15:0>, R5<15:0>, R6<15:0>, R7<15:0>, R8<15:0>,
          ALATCH<15:0>, BLATCH<15:0>, MAR<15:0>, MDR<15:0>, MPC<15:0>,
          MIR<31:0>, ALU<15:0>, SHIFTER<15:0>, MUX<15:0>, Z, N;
memory    CM[255:0]<31:0>, MM[4095:0]<15:0>;
bus       ABUS<15:0>, BBUS<15:0>, CBUS<15:0>;

/*  SUB MACRO definition  */
macro RML &x            /* register -> MUX */
{
        ALATCH = &x;  MUX = ALATCH;
}
macro AAR &x            /* ALU ADD output -> register */
{
        ALU = MUX + BLATCH;
        SHIFTER = ALU;  &x = SHIFTER;
}
macro MRL &x            /* MUX -> register */
{
        ALU = MUX;   SHIFTER = ALU;
        &x = SHIFTER;
}

/***** MACRO Instruction *****/
macro LOAD &a &b                /* &a = MM[&b] */
{
        if ( &b == MAR )
                ;
        else {
                BLATCH = &b;  MAR = BLATCH;
        }
        if ( &a == MDR )
        else {
                read MDR MAR; MUX = MDR; MRL &a;
        }
}
macro MOVE &a &b
{
        if ( &b == MDR )
                MUX = MDR;
        else
                RML &b;
        if ( &a == MDR ) {
                ALU = MUX; SHIFTER = ALU; &a = SHIFTER;
        }
        else
                MRL &a;
}
macro ADD  &a &b &c
{
        RML &b1;  BLATCH = &c1;  AAR &a1;
}
macro INC &a
{
        ADD  &a &a 1;
}
                .
                .
                .
```

(a)

```
microinstruction(32)
{       field positions {
                AMUX : <31>:     /* MAR,ALATCH selection field */
                COND : <30:29>:  /* conditional branch field */
                ALUC : <28:27>:  /* ALU operation field */
               SHIFT : <26:25>:  /* shift operation field */
                MDRC : <24>:     /* MDR selection field */
                MARC : <23>:     /* MAR selection field */
                  RD : <22>:     /* main memory read field */
                  WR : <21>:     /* main memory write field */
                 ENC : <20>:     /* destination selection field */
                   C : <19:16>:  /* destination register field */
                   B : <15:12>:  /* source register B port field */
                   A : <11:8>:   /* source register A port field */
                ADDR : <7:0>:    /* branch address/literal field */
        }
        field values {
                switch ( AMUX [phase 2] ) {
                        b'0' : MUX = ALATCH;
                        b'1' : MUX = MDR;
                }
                switch ( ALUC [phase 3] ) {
                        b'00' : ALU = MUX + BLATCH;
                        b'01' : ALU = MUX & BLATCH;
                        b'10' : ALU = MUX;
                        b'11' : ALU = ~MUX;
                }
                                .
                                .
                                .
                MARC [phase 3] : MAR = BLATCH;
                MDRC [phase 4] : MDR = SHIFTER;
                RD [phase 1] : read;
                WR [phase 4] : write;
                switch ( ENC [phase 4] ) {
                        b'1' :
                                switch ( C [phase 4] ) {
                                        b'0000' : PC = SHIFTER;
                                        b'0001' : AC = SHIFTER;
                                        b'0010' : SP = SHIFTER;
                                        b'0011' : IR = SHIFTER;
                                        b'0100' : TIR = SHIFTER;
                                                .
                                                .
                                }               .
                                                .
                                                .
                }
        }
}
```

(b)

그림11. 예제 대상머신에 대한
(a) 매크로 정의　(b) MDL 기술
Fig. 11. (a) Macro definition,
(b) MDL description for example microarchitecture.

```
microprogram Fibonacci
{
/*-------  MOVE  AC  200 expansion  -------*/
  ALATCH=0xc8;
  MUX=ALATCH;
  ALU=MUX;
  SHIFTER=ALU;
  AC=SHIFTER;
*--------  MOVE  R3  AC expansion  -------*/
  ALATCH=AC;
  MUX=ALATCH;
  ALU=MUX;
  SHIFTER=ALU;
  R3=SHIFTER;
/*-------  LOAD  MDR  100 expansion  -------*/
  BLATCH=0x64;
  MAR=BLATCH;
  read MDR MAR;
                    .
                    .
                    .
}
```

(a)

```
마이크로주소    필드명 제어값 필드위치          생성된 코드
0                                              0x 101105c8
                A :   5   <11: 8> ,
             ADDR :  c8   < 7: 0> ;
             AMUX :   0   <31:31> ;
             ALUC :   2   <28:27> ;
            SHIFT :   0   <26:25> ;
                C :   1   <19:16> ;
              ENC :   1   <20:20>
1                                              0x 101a0100
                A :   1   <11: 8> ,
             AMUX :   0   <31:31> ;
             ALUC :   2   <28:27> ;
            SHIFT :   0   <26:25> ;
                C :   a   <19:16> ;
              ENC :   1   <20:20>
2                                              0x 00805064
                B :   5   <15:12> ,
             ADDR :  64   < 7: 0> ;
             MARC :   1   <23:23>
3                                              0x 00400000
               RD :   1   <22:22>
                    .
                    .
                    .
```

(b)

그림12. Fibonacci MDIL 대한
(a) 매크로 확장
(b) 마이크로코드 생성

Fig. 12. (a) Macro expansion,
(b) microcode generation for Fibonacci MDIL.

중간언어의 명령어에 대하여 심볼릭 마이크로오퍼레이션을 정의하고, C 언어와 유사한 구조의 고급언어 형태를 갖는 MDL 기술언어로써 각 하드웨어 자원 및 마이크로명령어 형식에 따른 필드배치, 마이크로오퍼레이션 선택 제어조건 등을 기술하여 다양한 마이크로아키텍처에 대하여 머신 독립적으로 마이크로코드를 자동생성 하였다. 제안된 시스템은 마이크로코드 자동생성뿐 만 아니라 새로운 마이크로아키텍처의 설계와 기존 마이크로아키텍처의 변경 시 설계 자동화 도구로 응용될 수 있다. 앞으로의 연구 과제로서 기존에 개발된 마이크로코드 최적화 기법을 제안된 시스템에 적용한다면 효율적인 마이크로 코드 자동생성이 기대된다.

參 考 文 獻

[1] M. Sint, "MIDL - A Microinstruction Description Language," *IEEE 14th Annual Workshop on Microprogramming*, pp. 95-106, Oct. 1981.

[2] J.L. Giesser, "On Horizontally Microprogramed Microarchitecture Description Techniques," *IEEE Trans. on Software Eng.* vol. Se-8, no. 5, pp. 513-525, Sept. 1982.

[3] P. Hwang, "MDSS: A Design Automation System Including a Hardware Language, a Microcode Generator and a Functional Simulator," Ph.D. thesis, University of Cincinnati, 1984.

[4] M. Balakrishan, "An Efficient Retargetable Microcode Generator," *IEEE 19th Annual Workshop on Microprogramming*, pp. 44-53, 1986.

[5] J.F. Nixon et. al., "A Microarchitecture Description Language for Retargetable Firmware Tools," *IEEE 19th Annual Workshop on Microprogramming*, pp. 34-43, Oct. 1986.

[6] P. Marwedel, "Retargetable Compiler for a High-Level Microprograming Language," *IEEE 17th Annual Workshop on Microprogramming*, pp. 267-274, Nov. 1985.

[7] Robert A. Mueller, "Horizon: A Retargetable Compiler for Horizontal Microarchitectures," *IEEE Trans. on Soft eng.* vol. 14, no. 5, pp. 575-581, May 1988.

[8] E.S. T. Fernandes, "Microarchitecture Modeling through ADL," *IEEE 21th Annual Workshop on Microprograming*, pp. 100-104, Nov. 1988.

[9] 이상정, 임인칠, "머신 독립적인 고급 마이크로프로그래밍 언어의 설계," 대한전자공학회 논문지, 제25권 제 3 호, pp. 39－49, 1988, 3.

[10] 이상정, "고급 마이크로프로그래밍 언어의 설계와 마이크로코드 자동생성 시스템에 관한 연구," 한양대학교 대학원 박사학위논문, 1988. 6.

[11] 조영훈, 김광준, 고대경, 서선희, 김영우, 이상정, 임인칠, "마이크로명령어를 이용한 마이크로코드 자동 생성 시스템," 정보과학회 컴퓨터 그래픽스 및 설계자동화 연구회 학술발표 논문집, pp. 65－67, 1990. 6.

[12] 조영훈, "마이크로명령어 기술언어를 사용한 기계 독립적인 마이크로코드 자동생성기에 관한 연구," 한양대학교 대학원 석사학위논문, 1990. 12.

著 者 紹 介

李 相 靜(正會員)

1960年 6月 3日生. 1983年 2月 한양대학교 전자공학과 졸업(공학사). 1985年 2月 한양대학교 대학원 전자공학과 졸업 (공학석사). 1988年 8月 한양대학교 대학원 전자공학과 공학박사학위 취득. 1988年 9月～현재 순천향대학교 공과대학 전산학과 조교수. 주관심분야는 마이크로프로그래밍, 병렬처리 시스템, VLIW 아키텍처 등임.

●

林 寅 七 (正會員) 第28卷 B編 6號 參照
현재 한양대학교 전자공학과 교수

曺 永 勳(正會員)

1960年 7月 7日生. 1988年 2月 한양대학교 전자공학과 졸업(공학사). 1991年 2月 한양대학교 대학원 전자공학과 졸업(공학석사). 1991年 3月～현재 SONY International Korea(주) 근무중. 주관심분야는 마이크로프로그래밍, 컴퓨터 H/W 등임.