머신 독립적인 마이크로코드 자동 생성

°박병수, 민경철, 김의정, 이상정, 임인칠
한 양 대 학 교

A Machine Independent Automatic Microcode Generation

° B.S.PARK, K.C.MIN, Y.J.KIM, S.J.LEE, I.C.LIM
Hanyang University

Abstract

This paper proposes a microcode generating system which automatically generates the microcode of various target machine by inputing the intermediate language (MDIL) from the machine independent HLML-C (High Level Microprograming Language C) language. The MOP's (Microoperations) which is modeled 7-tuples generate to extend MDIL by table driven method with the information of translation table for each target machine. As compaction being considered and the hardware resource of target machine used, the conflicts of hardware elements are removed possibly.

This proposed system is implemented with C language and yacc on VAX-11/750 (UNIX 4.3 BSD).

Ⅰ. 서 론

마이크로프로그램을 이용한 컴퓨터 시스템의 제어방식은 프로그램을 이용한 소프트웨어적인 제어가 가능하고 규칙적인 제어구조로 인하여 수정(flexibility) 및 확장(extensibility)이 용이하면서 저가격으로 다기능의 함수를 수행(funtionality)할 수 있다는 장점이 있다. 최근 대규모 계산의 처리를 요하는 응용 프로그램 등에 마이크로프로그램이 이용됨에 따라 마이크로프로그램이 복잡해지고 규모가 커지게 되어 고급 마이크로프로그램을 사용하여 마이크로코드를 자동생성하는 마이크로코드 자동생성 시스템에 대한 요구가 증대되고 있는 실정이다. [1] - [3]

본 논문에서는 머신 독립적인 고급 마이크로프로그래밍 언어인 HLML-C 언어를 입력으로 받아 컴파일하여 최종적으로 대상머신의 마이크로코드를 생성하는 마이크로코드 자동생성 시스템을 제안한다. HLML-C 언어[6]는 C언어와 유사한 구조를 갖는 머신 독립적인 고급 마이크로프로그래밍 언어로써 다양한 머신의 특성을 고려하여 일반화시킨 추상머신(abstract machine)상에서 그 오퍼레이션을 정의한다. HLML-C 언어로 작성된 마이크로프로그램은 각 변수에 대한 레지스터 할당 후 머신 종속적인 중간언어(Machine Dependent Intermediate Language, MDIL)와 각 대상머신의 정보를 입력으로 받아 7-tuple로 모델링한 MOP's를 생성한다. MOP's 생성시 MDIL의 OP코드, 각 입출력 오퍼랜드 및 대상머신의 하드웨어 요소를 고려하여 변환표를 작성한 후, 컴팩션을 고려하여 대상 머신의 하드웨어 자원을 고루 이용함으로써 하드웨어 요소의 상충관계를 최대한 제거한다.

이상에서 제안한 마이크로코드 자동생성 시스템은 VAX 11/750 (UNIX 4.3 BSD) 상에서 yacc 와 C언어로 실현하고 다양한 예의 마이크로프로그램들에 대하여 대상머신에 적용시켰다.

Ⅱ. 마이크로코드 자동 생성 시스템의 구성

HLML-C 시스템은 그림 1과 같이 구성되고, 그림 2는 HLML-C 언어로 작성된 곱셈 마이크로프로그램의 예이다.

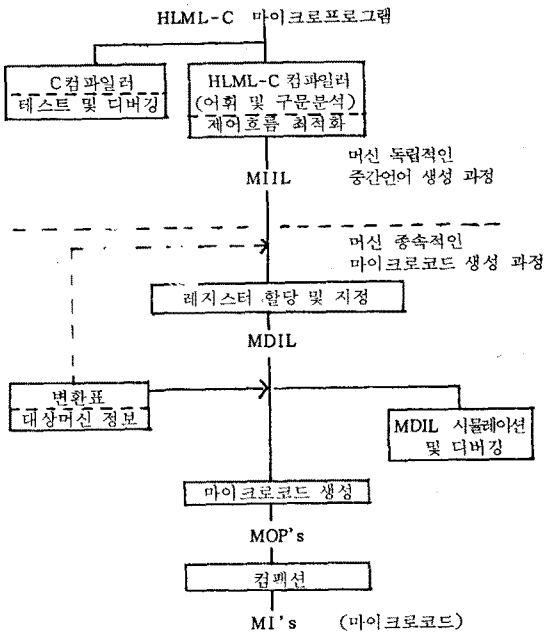

그림 1. HLML-C 시스템 구성도

```
/* 8 bit multiplication of sign magnitude */
#define SIGN 0X8000  /* for sign extraction */
#define DIGIT 0X7FFF /* for digit extraction */
memory 16 A:100, /* multiplicand */
          B:101, /* multiplier */
          C:102;  /* result */
microprogram multiplication:
{ register 16 mier, mcand, result;
  mier = B; /* get multiplier in the register */
  result = (mier & SIGN + A) & SIGN; /* get sign result */
  mier &= DIGIT; /* get digit of multiplier */
  mcand = A & DIGIT; /* get digit of multiplicand */
  while (mier != 0) { /* iterate until mier is zero */
        if ((mier & 1) == 1) /* if LSB of mier is one */
              result += mcand; /* add mcand to result */
        mier >>= 1; /* shift right one mier */
        mcand <<= 1; /* shift left one mcand */
  }
  C = result;
}
```

그림 2. HLML-C 언어로 작성된 곱셈 마이크로프로그램의 예

그림 3은 그림 2의 곱셈 마이크로프로그램 예에서 MIIL 생성후에 가용한 범용 레지스터의 수를 2, ALU, 메모리 레지스터가 각각 1개로 가정했을 때 레지스터 할당 후의 MDIL 생성 예이다. 여기서 %, #은 ALU, 메모리 레지스터를, $는 범용 레지스터를 나타내고, STORE #, LOAD # 은 레지스터 할당후에 삽입된 명령이다.

```
0     ENTRY    multiplication
1     LOAD     *SP      &TOP
2     ADD      *TOP     *SP      3
3     LOAD     #1       101
4     MOVE     $1       #1
5     AND      %1       $1       0X8000
6     LOAD     #1       100
7     ADD      %1       %1       #1
8     AND      %1       %1       0X8000
9     MOVE     $2       %1
10    STORE#   $2       2
11    AND      $1       $1       0X7FFF
12    LOAD     #1       100
13    AND      %1       #1       0X7FFF
14    MOVE     $2       %1
15    CJMPEQ   $1       0        26
16    AND      %1       $1       1
17    STORE#   $1       0
18    CJMPNE   %1       1        22
19    LOAD#    $1       2
20    ADD      $1       $1       $2
21    STORE#   $1       2
22    LOAD#    $1       0
23    SHRO     $1       $1       1
24    SHLO     $2       $2       1
25    JUMP                       15
26    LOAD#    $1       2
27    STORE    $1       102
28    EXIT     multiplication
```

그림 3. 곱셈 마이크로프로그램에 대한 MDIL 생성 예

Ⅲ. MOP 모델링

MDIL이 각 대상머신의 변환표를 입력으로 받아 마이크로코드를 생성할 때 7-tuple로 모델링된 다음과 같이 MOP를 생성한다.

MOP = ⟨ N, OP, I, O, E, F, T ⟩

여기서, N : MOP를 지칭하는 번호
OP : MOP가 수행하는 함수의 니모닉
I : MOP가 수행하는 함수의 입력이 되는 레지스터
O : MOP의 함수 결과가 출력되는 레지스터 집합
E : MOP의 함수에 이용되는 하드웨어 요소의 집합
F : 대상머신의 마이크로명령어 중에 MOP가 수행하는 함수가 위치한 필드 집합
T : MOP의 수행에 요구되는 타이밍 위상의 집합

Ⅳ. 마이크로코드 생성

3주소 코드의 MDIL로부터 7-tuple의 MOP는 MDIL의 OP코드, 각 입출력 오퍼랜드 및 대상머신의 하드웨어 요소를 고려하여 변환표를 작성한 후 작성된 변환표로부터 생성된다.

Ⅳ-1. 변환표 작성

변환표 작성은 먼저 MDIL의 OP코드가 수행될 대상머신의 하드웨어 요소를 활성화시키는 MOP의 tuple을 정의하고, MDIL의 입력 오퍼랜드에서 하드웨어 요소에 직접 연결된 레지스터로의 데이타 이동을 나타내는 MOP의 tuple을 정의한다. 또 수행 결과를 MDIL의 출력 오퍼랜드에 저장하는 오퍼랜드를 정의한다. 변환표의 작성은 다음과 같다.

단계 1) 각 MDIL의 오퍼레이션과 이를 수행하는 대상머신의 하드웨어 요소, MDIL의 오퍼랜드 갯수 및 종류에 따라 공통적으로 기술될 수 있는 MOP tuple의 집합을 고려하여 MDIL의 연산자 군을 분류한다.

단계 2) 분류된 연산자 별로 입력모드 변환표를 구성한다. 입력모드 변환표는 MDIL의 오퍼랜드로부터 하드웨어 요소에 직접 연결된 레지스터로의 데이타 이동을 나타내는 MOP의 tuple의 집합을 갖는 변환표이다.

단계 3) 분류된 연산자 별로 수행모드 변환표를 구성한다. 수행모드 변환표는 각 연산자를 수행시키는 하드웨어 요소를 활성화시키는 MOP의 tuple 집합을 갖는 변환표이다.

단계 4) 분류된 연산자 별로 출력모드 변환표를 구성한다. 출력모드 변환표는 수행 후의 결과를 MDIL의 출력 오퍼랜드에 저장하는 MOP tuple 집합을 갖는 변환표이다.

위와 같이 변환표 작성시에 각 MOP의 병렬처리를 위해 동일한 함수를 수행하는 2개 이상의 ALU를 갖는 대상머신의 경우를 고려하여 작성한다. 먼저 각 ALU를 활성화시키는 MOP tuple의 집합을 구하고 변환표 작성시에는 1개의 ALU를 갖는 머신으로 간주하고 MOP 생성시에 특정 ALU로 결합한다.

(예) Lah의 머신[4](그림 4)의 경우 다음과 같이 MDIL의 연산자군이 분류된다.

(1) ALU 2항 연산자 : ADD, SUB, ADDC, SUBC, AND, OR, NAND, NOR, NXOR, CAND
(2) 단항 연산자 : INC, DEC, CMP
(3) 쉬프트 연산자 : SHRO, SHLO, ROTL, ROTR
(4) 이동 연산자 : MOVE
(5) 메모리 연산자 : LOAD, STORE, RMOVE, WMOVE

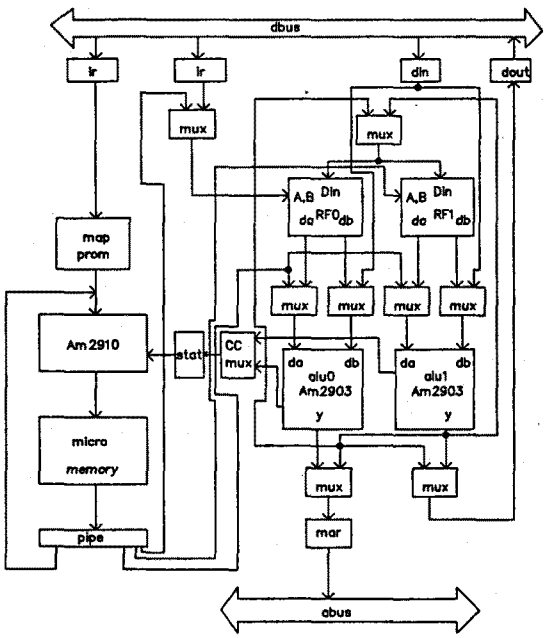

그림 4. Lah 머신의 시스템 구성도

(6) 제어흐름 연산자 : JUMP, CJMPGE, CJMPGT, CJMPLE, CJMPLT, CJMPEQ, CJMPNE, CJMP

그림5는 ALU 2항 연산자의 입력 오퍼랜드 중 첫번째 입력 오퍼랜드에 대한 입력모드 변환표 작성 예이다.

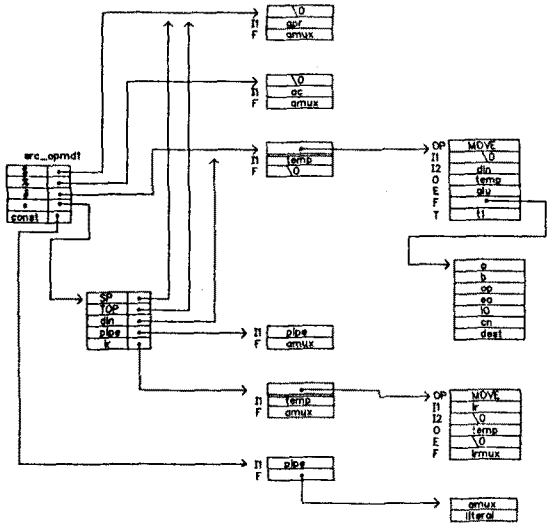

그림5. Lah 머신에 대한 ALU 2항 연산자군의 1항 오퍼랜드의 입력모드 변환표

Ⅳ-2. MOP 생성

MDIL과 대상머신의 하드웨어를 고려하여 변환표를 작성한 후 MOP를 생성한다. MOP 생성시 대상머신이 동일함수를 이용하는 ALU가 2개 이상인 경우에는 컴팩션을 고려하여 MOP를 생성한다. 즉, 대상머신의 ALU를 고루 이용하여 하드웨어 요소의 상충관계를 최대한 제거하기 위하여 각 MDIL에 번갈아 가며 ALU를 할당한다. MOP 생성 단계는 다음과 같다. 여기서 MOP를 생성할 MDIL을 MDILi라 하고 동일함수를 수행하는 대상머신의 갯수를 n이라 한다. 그리고 MDILi가 ALU를 이용하는 경우 MDILi가 결합될 ALU를 지정하는 변수를 alu_flag 이라 할 때 다음과 같이 MOP를 생성한다.

단계1) alu_flag의 값을 n으로 초기화시키고, 첫번째 MDIL 문을 읽는다.

단계2) MDILi가 ALU 함수인 경우 alu_flag을 다음과 같이 계산한다.

alu_flag = alu_flag mod n

단계3) MDILi의 입력모드 변환표에 따라 MOP tuple을 생성한다. 이때 MDILi가 ALU를 이용하는 함수인 경우에는 alu_flag이 지정하는 ALU를 나타내도록 MOP tuple을 생성한다.

단계4) MDILi의 수행모드 변환표에 따라 MOP tuple을 생성한다. 이때 MDILi가 ALU를 이용하는 함수인 경우에는 alu_flag이 지정하는 ALU를 나타내도록 MOP tuple을 생성한다.

단계5) MDILi의 출력모드 변환표에 따라 MOP tuple을 생성한다. 이때 MDILi가 ALU를 이용하는 함수인 경우에는 alu_flag이 지정하는 ALU를 나타내도록 MOP tuple을 생성한다.

단계6) MDILi가 ALU를 이용하는 경우 alu_flag을 하나 증가시킨다.

단계7) 다음 MDIL문을 읽고 마이크로프로그램의 끝이 아니면 단계2로 가고, 끝이면 MOP 생성을 마친다.

그림6은 동일함수를 수행하는 ALU가 2개(n=2)인 Lah의 머신상에서 그림3의 곱셈 마이크로프로그램 MDIL 중 3번째 기본블럭에 속하는 16, 17, 18번지의 MDIL에 대해 7-tuple로 변환된 MOP 생성 예이다.

```
0       <OP> : ENTRY :: multiplication                    <T> :
        <I> :  ..       <O> :
        <E> :
        <F> :
                              :
                              :
32      <OP> :  MOVE             <T> :  t1
        <I> :  pipe  ..          <O> :  ql
        <E> :  alu1
        <F> :  dest1  cn1  101  ea1  op1  b1  a1
               literal(1)  amux1
33      <OP> :  AND              <T> :  t1
        <I> :  gpr1  ..  ql      <O> :  stat  ac
        <E> :  alu1
        <F> :  ccmux  lstat  we1  oey1  rfmux
               dest1  cn1  101  ea1
               op1  b1  a1  amux1
34      <OP> :  MOVE             <T> :  t1
        <I> :  pipe  ..          <O> :  mar
        <E> :  alu0
        <F> :  marmux  oey0  dest0  cn0  100  ea0
               op0  b0  a0  literal(0)  amux0
35      <OP> :  MOVE             <T> :  t1
        <I> :  gpr1  ..          <O> :  dout
        <E> :  alu0
        <F> :  ldout  oey0  doutmux  dest0  cn0
               100  ea0  op0  b0  a0  amux0
36      <OP> :  WRITE            <T> :  t1
        <I> :  mar  ..  dout     <O> :
        <E> :  abus  dbus
        <F> :  write
37      <OP> :  MOVE             <T> :  t1
        <I> :  pipe  ..          <O> :  ql
        <E> :  alu1
        <F> :  dest1  cn1  101  ea1  op1  b1  a1
               literal(1)  amux1
38      <OP> :  SUB              <T> :  t1
        <I> :  ac  ..  ql        <O> :  stat
        <E> :  alu1
        <F> :  lstat  ccmux  dest1  cn1  101  ea1
               op1  b1  a1  amux1
39      <OP> :  CJMPF            <T> :  t1
        <I> :  ZERO  stat  ..  pipe    <O> :
        <E> :
        <F> :  addr(48)  seq
                              :
                              :
70      <OP> :  EXIT :: multiplication                    <T> :
        <I> :  ..       <O> :
        <E> :
        <F> :
```

그림6. Lah의 머신에 대한 곱셈 마이크로프로그램 MOP 생성 예

Ⅴ. 코드 확장의 생성

표1은 HLML-C언어로 작성된 6개의 테스트 마이크로프로그램이 중간언어로 확장 생성되는 과정을 나타내는 표이다. 표2는 Lah와 Cal data머신[5]에 대한 각 테스트 마이크로 프로그램의 MOP's의 확장 생성 비율을 나타낸 표이다.

표1. 중간언어 확장 생성비율

마이크로프로그램 / 결과		prime	fbnc	bubble	cvrt	fmpy	fadd
HLML-C 실행문수		16	13	12	21	54	89
범용레지스터변수 수		5	5	4	7	12	13
MIIL 문 수		46	22	22	34	94	169
MDIL 문 수	**R=7	46	22	22	34	112	185
	R=5	46	22	22	42	122	217
	R=3	56	29	25	51	161	233

** R : 가용한 범용 레지스터 수

표 2. MOP's 확장 생성비율

마이크로프로그램 / 결과		prime		fbnc		bubble		cvrt		fmpy		fadd	
		Lah	Cal	Lah	Cal	Lah	Cal	Lah	Cal	Lah	Cal	Lah	Cal
MOP 수	R = 7	79	97	37	42	40	49	76	82	267	290	415	443
	R = 5	79	97	37	42	40	49	100	114	297	330	509	569
	R = 3	109	137	58	70	49	61	125	147	410	480	557	633
MOP/MDIL	R = 7	1.72	2.11	1.68	1.91	1.82	2.23	2.24	2.41	2.38	2.59	2.24	2.39
	R = 5	1.72	2.11	1.68	1.91	1.82	2.23	2.38	2.71	2.43	2.70	2.35	2.62
	R = 3	1.95	2.45	2.64	2.41	1.96	2.44	2.45	2.88	2.55	2.98	2.39	2.72
MOP/HLML-C	R = 7	4.94	6.06	2.85	3.23	3.33	4.08	3.62	3.90	4.94	5.37	4.66	4.98
	R = 5	4.94	6.06	2.85	3.23	3.33	4.08	4.76	5.43	5.50	6.11	5.72	6.39
	R = 3	6.81	8.56	4.46	5.38	4.08	5.08	5.95	7.00	7.59	8.89	6.26	7.11

Ⅵ. 결 론

본 논문에서는 머신 독립적인 고급 마이크로프로그래밍 언어인 HLML-C 언어를 사용하여 각 대상머신의 마이크로코드를 자동 생성하는 마이크로코드 자동생성 시스템을 제안하였다. 생성된 MDIL은 각 대상머신의 변환표 정보를 입력으로 받아 7-tuple로 모델링된 MOP's로 생성된다. 변환표에 의하여 분류 구성된 각 MDIL의 오퍼레이션 및 오퍼랜드들은 변환표 구동방식에 의하여 MOP의 각 tuple로 확장 생성된다. 이때 각 MOPS's의 병렬처리 가능성을 최대한 보장하기 위해서 대상머신의 각 하드웨어 자원을 고루 이용하여 병렬처리에 장애가 되는 하드웨어 자원 상충관계를 최대한 제거하였다.

본 마이크로코드 자동 생성 시스템은 VAX 11/750(UNIX 4.3 BSD) 상에서 구문 자동생성기인 yacc와 C언어를 사용하여 도구화하였고, 수평 마이크로명령어 형식을 갖는 Lah 머신을 대상머신으로 하여 모두 6개의 테스트 마이크로프로그램을 작성하여 적용 비교 검토하여 제안된 고급 마이크로프로그래밍 언어의 마이크로코드 자동생성 시스템의 타당성 및 실용성을 입증하였다.

참 고 문 헌

1. S.Davidson, "*Progress in High-Level Microprogramming*", IEEE Software Vol.3, No.4, pp.18 ~ 26, July 1986.
2. R.J.Sheraga and J.L.Gieser, "Automatic Microcode Generation for Horizontally Microprogrammed Processors", IEEE 14th Annual Workshop on Microprogramming, pp.154 ~ 168, Oct. 1981.
3. S.R.Vegdahl, "Local Code Generation and Compaction in Optimizing Microcode Compilers", Ph.D. thesis, Carnegie-Mell Univ., 1982.
4. J.Lah, "Development and Analysis of a Global Compaction Technique of Microprograms", Ph.D.thesis, Univ. of Michigan, 1984.
5. A.K.Agrawala, T.G.Rauscher, "Foundations of Microprogramming", Academic Press, 1976.
6. 이상정, 임인철, "머신 독립적인 고급 마이크로프로그래밍 언어의 설계", 대한전자공학회 논문지, 제 25 권 제 3 호, pp.39 ~ 49, 1988.