論文 86-23-4-8

에뮬레이터 개발을 위한 중간 마이크로프로그래밍 언어의 변환

(A Translation of the Intermediate Microprogramming Language for Emulator Development)

崔 基 浩*, 林 寅 七**

(Ki Ho Choi and In Chil Lim)

要 約

본 논문에서는 마이크로프로그램 가능한 호스트 머신(host machine) 상에서 타겟트(target) 머신을 에뮬레이션하기 위한 기계독립적인 중간 마이크로프로그래밍 언어(Intermediate Microprogramming Language; IML)를 레지스터 할당하고 변환표와 MI(microinstruction) 형식 및 필드 정보 등을 이용하여 마이크로코드화하는 변환 시스템을 제안한다.

마이크로프로그램 가능한 가상의 16bit 호스트 머신상에서 PDP-8을 에뮬레이션하기 위한 IML을 제안된 변환시스템에 의해 마이크로코드화하고, 이 에뮬레이터상에서 타겟트 머신의 테스트 프로그램이 실행되는 것을 시뮬레이션함으로써 제안된 변환시스템에 대한 알고리즘의 타당성을 입증한다.

Abstract

This paper proposes a system that translates the machine independent intermediate microprogramming language (IML) into microcode, using the register allocation algorithm, the microinstruction format and the field information for the target machine emulation on a microprogrammable host machine. The IML, which is for PDP-8 emulation on a microprogrammable hypothetical 16 bit host machine, is microcoded by the proposed system, and the validity of the algorithm in the proposed system is verified by executing a test program of the target machine on the emulator.

I. 序 論

반도체기술의 급속한 발전에 힘입어 미니컴퓨터 수준 이상의 대부분의 컴퓨터들이 마이크로프로그램 제어방식을 이용하게 되었고, 이러한 컴퓨터 시스템에 대해 손쉽게 마이크로프로그램 하기 위한 도구개발에 최근 많은 연구가 진행되고 있다.[1,4]

새로운 컴퓨터 개발시나 마이크로프로그램 가능한 컴퓨터에서 기존의 컴퓨터가 갖고 있는 것과 같은 소프트웨어를 개발하려면 많은 시간과 비용이 들므로, 마이크로프로그램 가능한 호스트컴퓨터 상에서 마이크로코드로 기존 타겟트컴퓨터의 기계어 명령을 실행가능토록 하는 에뮬레이터를 개발하면 이를 쉽게 해결할 수 있다. 이러한 에뮬레이터 개발의 단가를 감소시키고 신뢰도를 증가시킬 수 있는 가장 바람직한 방법

*正會員, 光云大學校 電子計算機工學科
(Dept. of Computer Eng., Kwangwoon Univ.)
**正會員, 漢陽大學校 電子工學科
(Dept. of Electronic Eng., Hanyang Univ.)

接受日字 : 1986年 3月 24日
(※ 本 研究는 韓國電子通信研究所의 委託 研究課題의 一部로서 遂行된 것임.)

은 HLML(High Level Microprogramming Language)로 타겟트머신에 대해 의도하는 프로그램을 작성하고 이것을 호스트머신의 수평 마이크로코드(horizontal microcode)로 손쉽게 변환할 수 있는 이식가능한 마이크로코드 생성시스템 개발이다.

이러한 마이크로코드 생성시스템은 그림 1에서와 같이, HLML을 기계독립적인 중간 마이크로프로그래밍 언어(이하 IML : Intermediate Microprogramming Language)로 컴파일하는 부분과, 레지스터를 할당하고 각 기능부(function unit)들을 대응(mapping) 시켜 코드화된 MOP(microoperation) 시퀀스를 만드는 자원할당(resource allocation) 부분과, 이것을 수평형식(horizontal format)으로 코드를 최적화하는 마이크로코드 최적화(microcode compaction) 부분등 크게 세부분으로 나눈다. 효율적인 수평 마이크로코드를 생성하는데는 자원할당과 마이크로코드의 최적화가 문제가 되는데 이들중 자원할당 문제가 보다 더 큰 영향을 미친다.[5]

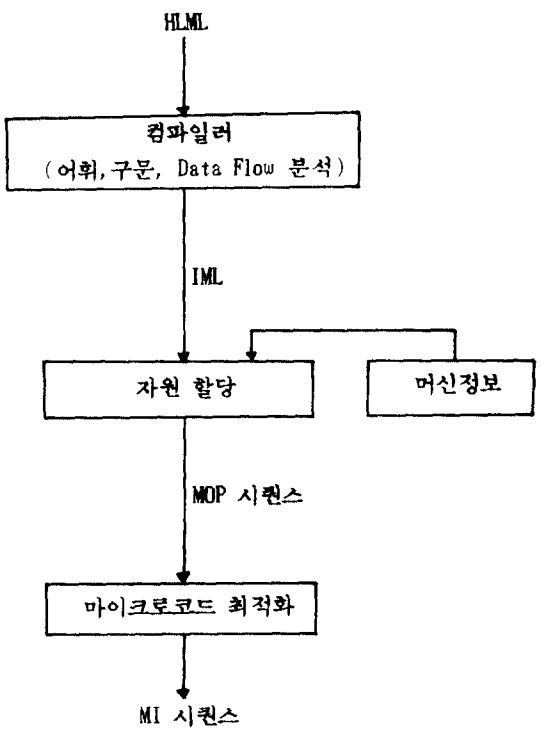


그림 1. 이식가능한 마이크로코드 생성 시스템
Fig. 1. A protable microcode generation system.

Dewitt[6]은 HLML로서 EMPL(Extensible Microprogramming Language)과 CWM(Control Word Model)을 제안하였고 최적화와 레지스터할당의 복합문제를 풀 수 있도록 코드생성에서 기호변수(symbol variable)를 이용하고 여러가지 마이크로프로그램 가능한 머신을 묘사할 수 있는 모델과 R A/D(Register Allocation/Deallocation)시 발생할 수 있는 문제점을 제시했으나 제안된 변환시스템은 너무 일반적이어서 실제 머신에 적용하지 못하였다.

Malik[7]은 HLML로 타겟트 머신의 모든 기억자원(storage resource)들을 변수형태로 나타낼 수 있는 VMPL(Virtual Micro Program Language)을 제안하였고, Ma와 Lewis[5,8,10]는 VMPL 컴파일러에 의한 기계독립적인 중간언어를 호스트머신의 기계종속적인 중간코드로 대응시킨후 R A/D를 수행하고 linear order 알고리즘을 이용해 최적화를 시도했다. 그러나 기계종속적인 중간언어로 레지스터할당을 하기 때문에 알고리즘이 복잡하고, 특히 루프에 대한 레지스터 할당이 비효율적이라는 문제점을 갖고 있다.[2]

Mueller[11,12]는 변수에 프로그램 흐름도(program flow graph)의 개념을 도입한 symbolic assertion 이용을 제안하였고 기본블럭(basik block)에서 흐름도를 이용한 마이크로코드 생성기법과 자원할당을 결합시켜 최적화 가능성을 제시하였으나 실제 머신에 적용되기에는 많은 문제점이 남아있다.

Hopkins[1] 등은 C언어를 이용한 Micro-C로 원시프로그램을 수직형의 기호 마이크로코드(vertical symbolic microcode)로 컴파일하고, 컴파일된 중간언어를 ISPS와 MICL(Microinstruction Constraint Language)에 의해 제공되는 마이크로구조 정보를 이용하여 packer와 어셈블러로 실행가능한 마이크로코드를 생성시키는 HLML 시스템을 연구 중이다.

국내에서는 미니컴퓨터 개발을 목표로 ETRI에서 IBM 370/138을 target으로한 32 bit VM머신 에뮬레이터를 마이크로 어셈블리어를 이용하여 연구 개발중에 있다.

본 논문에서는 HLML로 쓰여진 마이크로프로그램이 컴파일되어 IML로 주어졌을 때, 기계독립적인 IML 문에 대해 먼저 R A/D를 수행케함으로써, 기계종속적인 상태에서 레지스터를 할당하는 것보다 알고리즘 실행을 빠르게하고 보다 효율적인 MOP 시퀀스로 변환할 수 있는 시스템을 제안한다.

또한, 본 시스템의 적용가능성을 보이기 위하여 실존 머신으로서 비교적 간단한 기계어 명령세트를 갖는 PDP-8을 타겟트 머신으로 정하고, 마이크로프로그램 가능한 가상의 16 bit 호스트 머신에 대해 본 시스템을 적용한 예를 들었고, 출력된 MOP시퀀스 결과상에서 타겟트 머신의 테스트 프로그램이 실행되는 것을 시뮬

레이션함으로써 제안된 변환 시스템에 대한 알고리즘의 타당성을 입증한다.

Ⅱ. 變換 시스템의 構成

1. 시스템 구성도

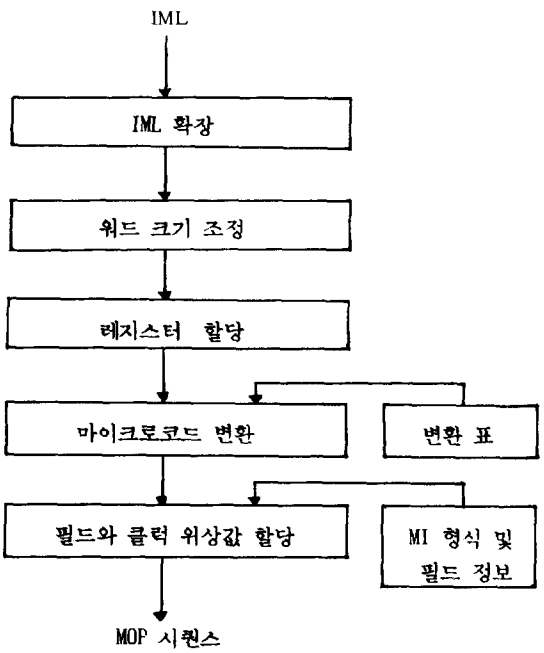


그림 2. 변환 시스템의 구성도
Fig. 2. Transformation system diagram.

그림 2는 제안되는 변환시스템의 구성도를 나타낸다. 먼저 복잡한 IML을 간단한 IML로 바꾸고 호스트와 타겟트 머신 간의 워드 크기(word size)와 산술연산 방식(mode) 차를 처리하며 타겟트 머신만 갖는 하드웨어 부분을 시뮬레이션하고, 기계독립적인 IML 형태에서 변수를 범용 레지스터(General Purpose Register : GPR, 또는 working register)에 할당하고 제어흐름을 인터페이스하며 또한 MI 형식의 각 필드와 제어기억 사이클(control store cycle)의 해당 위상값을 할당한다).

2. IML 설정

한 HLML로부터 컴파일러에 의해 블럭과 SLC(Straight Line Code)의 경계와 변수 상태 등을 참고 할 수 있는 부분과 quadruple 형태의 실행문장 부분을 갖는 타겟트 머신 에뮬레이션을 위한 IML이 아래 예와 같이 주어졌다고 가정한다.

이러한 가정은, Dewitt의 EMPL[6]이나 Malik의 VMPL[7]이 선언문과 실행문으로 구성되고, 가정된 IML과 비슷한 형태인 IISG(Intermediate Information Statement Group)와 IESG(Intermediate Executable Statement Group)로 컴파일되는데 근거를 두고 있다.

예) B373 LOOP ADD ACCM T.01 ACCM
0404 MOVE ACCM MDR

여기서
0 : 공백
A : 레이블
B : 블럭 레이블
1 : 메모리
2 : 상수
3 : 광역적 변수로서 사용형
4 : 광역적 변수로서 비사용형
5 : 국소적 변수로서 사용형
6 : 국소적 변수로서 비사용형
7 : 임시변수로서 사용형
8 : 임시변수로서 비사용형

단, 사용형은 해당 블럭내에서 이후 사용될 변수이고 비사용형은 이후 사용 안될 변수이다.

3. IML 확장

복잡한 IML 코드를 단순한 IML 코드로 바꾼다.

예) LOOP SRC1 SRC2 SRC3;Loop for SRC1 = SRC2 to SRC3 by 1을 간단한 IML 코드로 바꾸면 아래와 같다.

```
        MOVE   SRC2        SRC1
L.01    COMP   SRC3  SRC1
        CONDT  N           L.02
        INC                SRC1
          •
        BRCH               L.01
L.02    •
```

4. 워드 크기 조정

타겟트 머신과 호스트 머신간의 워드 크기 차이가 있으면 이를 조정한다. 호스트 머신의 워드 크기가 타겟트 머신의 워드 크기 보다 클 경우 호스트 머신 레지스터를 좌변정렬(left-justifying)시킴으로써 호스트 머신의 프래그(flag)를 이용할 수 있다. 또한 타겟트 머신의 워드 크기가 호스트 머신의 n배일 경우 IML변수를 n개의 세그멘트로 묶어서 대응시킨다.

예) 호스트 머신이 16 bit이고 타겟트 머신이 12 bit인 경우

```
INC          SRC1;SRC1에 1을 가산
NOT  SRC1    DEST;SRC1의 1의 보수
```

은

```
ADD  SRC1  16     SRC1
NOT  SRC1         DEST
AND  DEST  65520  DEST
```

와 같이 된다. 이때 12 bit 타겟트 머신의 프로그램과 데이타는 먼저 모두 왼쪽으로 4 bit씩 이동시켜서 (이때 0을 삽입) 16 bit 호스트 머신 메모리에 기억시킨다.

5. 레지스터 할당 알고리즘

기계독립적인 IML 문단위로 기호 변수들을 레지스터에 할당하는 것은 호스트 머신의 MOP 시퀀스 형태로 변환시킨 기계 종속적인 중간 코드에서 레지스터 할당하는 것보다 명령문 수가 훨씬 적기 때문에 방법이 간단하며 광역적이다. 또한 VMPL 컴파일러[7]에서와 같이 HLML을 IML로 컴파일시에 생기는 임시변수 중 이후 사용되지 않으면서 단지 다른 변수로 이동하는 경우는 그 destination되는 변수로 대치함으로써 한 IML을 제거할 수 있다.

예) B137 INF RMOVE MEM PC T.001
0804 MOVE T.001 IR

T.001은 컴파일러에 의한 임시변수이기 때문에 T.001에 할당된 레지스터의 변수명을 IR로 바꾸면 두번째 IML문은 제거할 수 있다. 만일 IML을 호스트 머신의 MOP 시퀀스로 먼저 변환시킨 후 R A/D를 수행한다면 MOVE명령을 위한 MOP 시퀀스를 제거할 수 없으므로 IR에 대해서도 R A/D를 수행해야 한다.

일반적으로 에뮬레이션 프로그램에서 기호변수로 된 오퍼런드 수는 호스트 머신의 GPR 수 보다 훨씬 크므로 레지스터는 한개 이상의 변수들에 의해 공유된다.

따라서 GPR이 충만시 새로운 한 변수를 할당하고자 할 때 자유 레지스터(free register)가 없으므로, 레지스터에 이미 할당되어 있는 변수의 상태를 고려하여 가능한 레지스터와 메모리간의 교체(swapping) 수를 줄이는 방향으로 대피(deallocation)한 후 자유레지스터를 만들고 새 변수를 할당하도록 한다.

레지스터 상태는 다음과 같이 그 레지스터에 할당된 변수의 명칭, 상태, 범위, 형, 위치로써 정의한다.

① 변수상태 : 레지스터에 있는 변수 내용이 메모리의 변수 내용과 틀리면 상이상태, 같을 경우에는 동일상태.

② 변수범위 : 한 블럭 내에서만 액세스 가능한 변수는 국소적(local) 변수, 전체 프로그램에서 액세스 가능한 변수는 광역적(global) 변수.

③ 변 수 형 : 레지스터에 할당된 후에 최소 한번 이용되고 다음에 다시 현 블럭 내에서 사용되면 사용형, 현 블럭 내에서 다시 사용되지 않으면 비사용형.

④ 변수위치 : 해당 IML 문에서의 오퍼런드 위치로서, source 오퍼런드이면 SRC, destination 오퍼런드이면 DEST.

새 변수 할당시 자유 레지스터가 없는 경우 레지스터 상태에 따라 표 1과 같이 대체우선순위를 정하고, 우선순위가 높은(번호가 작은) 쪽을 먼저 대피하며 우선순위가 같은 경우는 변수의 이용빈도수가 적은 쪽을 먼저 대피하도록 한다.

표 1. 대체 우선 순위표
Table 1. Replacement priority table.

우선순위	상 태
1	현 블럭에서 사용되지 않는 동일 상태인 국소적 변수
2	현 블럭에서 사용되지 않는 상이 상태인 국소적 변수
3	현 블럭에서 사용되지 않는 동일 상태인 광역적 변수
4	현 블럭에서 사용되지 않는 상이 상태인 광역적 변수
5	현 블럭에서 사용될 동일 상태인 국소적 변수
6	현 블럭에서 사용될 동일 상태인 광역적 변수
7	현 블럭에서 사용될 상이 상태인 국소적 변수
8	현 블럭에서 사용될 상이 상태인 광역적 변수

1) SLC내에서의 레지스터 할당

프로그램은 여러 블럭으로 구성되고, 제어흐름을 인터페이스 하기 위해 각 블럭은 분기문과 레이블문에 의해 나누어지는 SLC들의 세트로 구성한다. 한 SLC의 상태는 오퍼런드를 레지스터에 할당한 상태이고 SLC 시작점에서의 상태를 초기상태 IS라하고 레지스터 할당이 완료된 SLC의 끝점에서의 상태를 최종상태 FS라고 한다.

예) 아래 ①과 같이 SLCi에 할당되어질 변수 VAR1, VAR2, VAR3이 있고 이용할 수 있는 레지스터는 R1, R2, R3이고 SLCi 시작 전의 레지스터 상태가 ②와 같을 때.

① IML

```
        SLCi 시작
0333 ADD VAR1 VAR2 VAR3
        SLCi 끝
```

VAR1, VAR2, VAR3이 모두 광역적 변수이면서 뒤에 사용되는 변수이다.

② SLCi의 레지스터 할당이 시작되기전 레지스터 상태

레지스터	변수	상태	범위	형
R1	VAR2	상이	광역적	사용
R2	VAR4	상이	국소적	사용
R3	VAR3	상이	광역적	사용

③ 레지스터 할당 후의 결과

```
WMOVE  R2    VAR4  MEM ;R2→MEM
                        [VAR4]
RMOVE  MEM   VAR1  R2   ;MEM
                        [VAR1]→R2
ADD    R2    R1    R3
```

④ SLCi의 FS

레지스터	변수	상태	범위	형
R1	VAR2	상이	광역적	사용
R2	VAR1	동일	광역적	사용
R3	VAR3	상이	광역적	사용

⑤ 마이크로 코드로 변환후의 결과

```
MBR←VAR4
MAR←MBR
MBR←R2
MM[MAR]←MBR      } 메모리 write (WMOVE)

MBR←VAR1
MAR←MBR
MBR←MM[MAR]      } 메모리 read (RMOVE)
R2←MBR

AC←R2
AC←AC+R1         } ADD
R3←AC
```

2) 전방분기시 SLC의 IS

i번째 SLC의 초기상태 IS(i)는 그 SLC에 제어흐름이 있는 모든 SLC의 FS들의 함수로 정의한다.

$$IS(i) = f\left(\sum_{k=1}^{m} FS(i_k)\right)$$

또한 각 레지스터의 초기상태는

$$ISR(i, j) = f_j\left(\sum_{k=1}^{m} FSR(i_k, j)\right)$$

로 정의한다.

n개의 FS에서 Rj(j번째 레지스터)가 갖고 있는 IS의 변수와 범위는 FS에 따르고 상태는 모두 동일상태일 때만 동일상태가 되고 그 이외에는 상이상태가 된다. 반면에 만약 n개의 FS중 한변수 이상 다른 것과 틀린다면 IS에 있는 Rj는 자유 레지스터가 되어야 한다.

3) 후방 분기시의 SLC의 FS

그림 3과 같이 SLC3에서 SLC2로 후방분기할 때 분기문 전의 상태는 SLC2의 시작상태와 같아야 한다.

이러한 경우 IS(2), NIS(2), BS(3)에 의해서 FS(3)를 결정해야 한다. 이때 NIS는 레이블문이 실행시의 상태이고 BS는 분기문 바로 직전의 상태이다.

예) ① IS(2)가 아래와 같을 경우

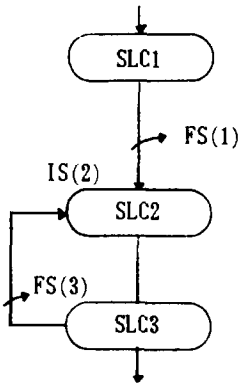


그림 3. 후방 분기의 예
Fig. 3. An example of backward branch.

레지스터	변수	범위	상태
R1	VAR1	광역적	상이
R2	VAR2	국소적	상이

② 이때 SLC2의 첫번째 문장이

```
LABEL1  MOVE  VAR3        VAR1
```

이라면 이 문장의 R A/D후의 NIS(2)와 출력은 다음과 같다.

NIS(2)

레지스터	변수	위치
R1	VAR1	DEST
R2	VAR3	SRC

```
WMOVE   R2    VAR2  MEM
RMOVE   MEM   VAR3  R2
LABEL1  MOVE  VAR3  VAR1
```

③ SLC3의 마지막 문장이

```
BRANCH          LABEL1
```

일 때의 분기문 이전의 상태 BS(3)는 다음과 같을 때

BS(3)

레지스터	변수	상태
R1	VAR4	상이
R2	VAR3	상이

④ SLC3의 FS를 결정하기 위해 NIS(2)와 BS(3)에서 레지스터에 있는 변수가 서로 다르면 BS(3)의 상태와 NIS(2)의 변수위치를 고려하여 SLC2의 첫 문장이 실행될 수 있도록 FS(3)를 결정한다. 본 예의 경우에는 R1의 변수가 서로 다르고 상태와 위치가 상이상태이고 DEST이므로 분기문 이전에 메모리 write문이 삽입된다.

```
WMOVE   R1  VAR4  MEM
BRANCH          LABEL1
```

FS(3) 레지스터 변수 상태
R1 VAR4 동일
R2 VAR3 상이
위 예의 결과를 보이면 다음과 같다.

```
        WMOVE   R2      VAR3    MEM
        RMOVE   MEM     VAR3    R2
LABEL1  MOVE    VAR3            VAR1
          •
          •
        -SLC2 끝, SLC3 시작-
          •
        WMOVE   R1      VAR4 MEM
        BRANCH                  LABEL1
```

레지스터 할당을 위한 알고리즘은 그림4와 같다. 그림5는 자유 레지스터 여부에 따른 변수의 R A/D 실행을 위한 흐름도이다.

6. 마이크로코드화

변환표를 이용하여 레지스터 할당된 각 IML 코드를 호스트 머신의 MOP 시퀀스로 대응(mapping)시킨다. (부록 변환 표 참조)

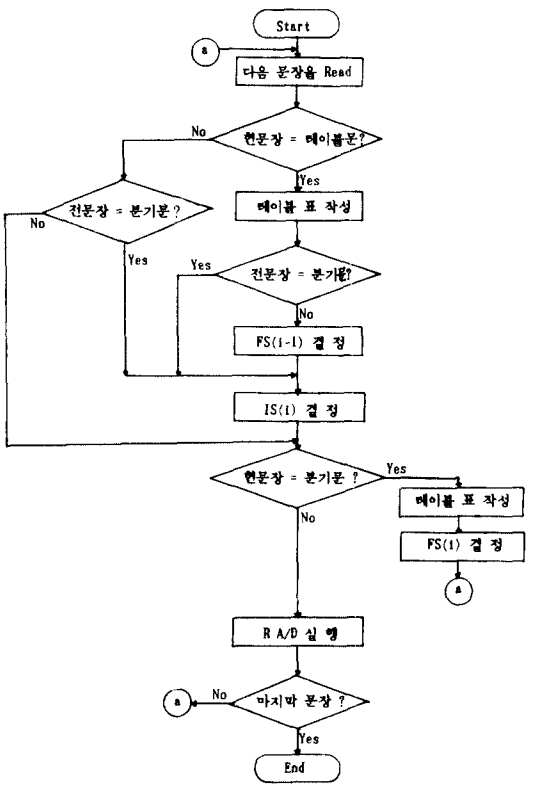


그림 4. 레지스터 할당 알고리즘
Fig. 4. Register allocation algorithm.

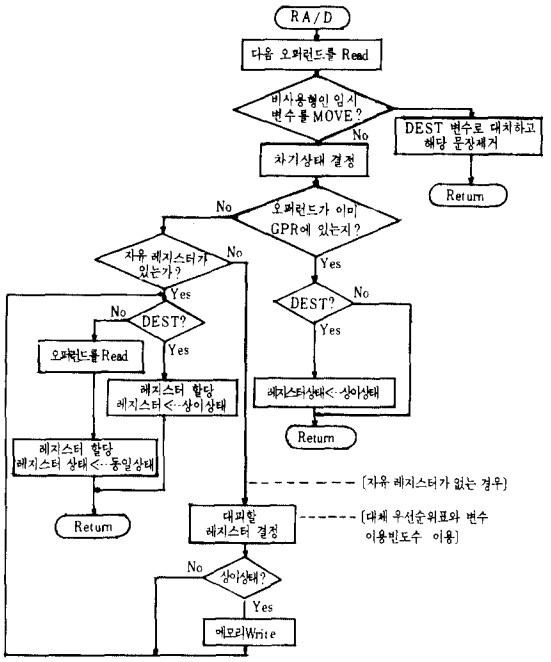


그림 5. R A/D 흐름도
Fig. 5. R A/D flow chart.

예) 레지스터 할당된 IML
ADD R1 R2 R1의 경우
AC←R1
AC←AC+R2
R1←AC

7. 필드 및 클럭 위상의 할당

호스트 머신의 MOP 세트를 M이라 할 때
$M = \{MOPi | i = 1, \cdots\cdots, n\}$
각 MOP는 5-tuple 세트로 나타낸다.
MOPi= {OP, I, O, F, P}
OP : 기본 연산을 지정
I : OP의 입력으로 사용되는 자원
O : OP의 출력으로 사용되는 자원
F : ⟨OP, I, O⟩ 실행시 MI 형식의 필드 세트
P : ⟨OP, I, O⟩ 실행시의 클럭 위상(phase)

본 논문의 가상 머신에서는 모든 MOP에 대해 위상이 같다고 가정한다(부록 MOP 리스트 참조). 이러한 경우 P는 생략될 수 있다.

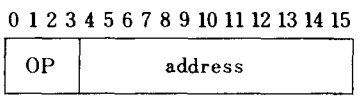
예 : AC← AC .AND. GPR인 경우
MOPi=⟨AND, I, O, F⟩ 형태로 나타낼 수 있으며
Bit 1-4 : GPR값에 의해 결정

Bit 24 : 1로 세트
그외 Bit : 0(zero)
가 된다.

Ⅲ. 適用 例 및 컴퓨터 시뮬레이션

호스트 머신으로는 부록의 MOP 리스트와 같은 마이크로 연산을 수행하는 가상의 마이크로프로그램 가능한 16bit 머신으로 구성하고, 타겟트 머신으로는 실존 컴퓨터로서 비교적 간단한 기계어명령 세트를 갖는 PDP-8으로 정하고 제안된 시스템을 보다 쉽게 적용시키며 그 타당성을 보이기 위해 다음과 같이 일부 수정하여 에뮬레이션 하였다.

* 연산처리 bit수를 16bit로 가정하였고
* 기계에 명령 형식은 아래와 같이 정하고



직접번지 지정(direct addressing) 하는 메모리 참조 명령만 가능하게 하였다.

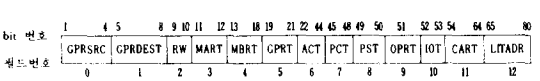
* 기계어 명령과 그 OP 코드는

0000 AND
0001 TAD
0010 ISZ
0011 DCA
0100 JMS
0101 JMP
0110 IOT
0111 OPT

로 정하고 IOT와 OPT의 세부 명령은 실행작동이 없는(no operation) 것으로 처리했다.

부록 MOP 리스트의 9-64번까지가 호스트머신의 MOP들이고 호스트머신의 MI형식은 MOP 리스트와 같이 80열의 수평형식(horizontal format)이다. 1-4번까지는 source 레지스터를 지정하고 5-8번까지는 destination 레지스터를 지정하는 필드이며, 65-80번까지는 상수값이나 번지를 지정하는데 이용된다.

Destination을 기준으로 하여 서로 배타적인(mutually exclusive) 관계에 있는 operation이나 function을 하나로 그룹지으면 아래와 같이 13개의 필드로 나눌 수 있다. 이것을 부호화 된 MI형식(encoded MI format)을 취한다면 전체 bit수를 줄일 수도 있다.



HLML로 쓰여진 에뮬레이션 프로그램이 컴파일되어 IML문으로 주어졌을 때, 기계독립적인 IML문 단위로 대체 우선순위와 변수 이용 빈도수를 이용한 레지스터 할당 알고리즘을 적용하고 호스트머신의 변환표를 이용하여 MOP 시퀀스로 변환하는 시스템을 VAX 11/750 상에서 FORTRAN으로 프로그램하여 구성하였다.

본 시스템에 대한 알고리즘의 타당성을 입증하기 위하여 부록의 PDP-8 PARTIAL IML을 본 시스템에 적용하였다. 기계종적인 IML 상태에서 레지스터 할당할 경우[8]보다 알고리즘 실행이 빠르고 이 경우 8개의 MOP를 줄일 수 있었다. 출력된 결과(부록 MOP sequence) 상에서 테스트 프로그램이 실행되는 것을 C 언어를 이용하여 시뮬레이션하였다. 시뮬레이션의 흐름도는 그림 7과 같다.

시뮬레이션시 MI형식을 앞에서와 같은 13개 필드로 나누어 MOP 시퀀스를 코드화하였다. 테스트 하기 위한 타겟트 머신의 기계어 프로그램은 1부터 100까지 합하는 프로그램으로 정하였고 그림6과 같다. 이때 END는 의사명령어(pseudo instruction)이다.

각 상수는 16진수이고, 테스트 프로그램은 8번지부터 시작하는 것으로 하여 호스트 머신의 메모리 1번지에 기호 변수 pc의 초기값 8을 넣었다.

시뮬레이션 결과 15번지에 $13BA_{16}$ ($=5050_{10}$)가 저장됨으로써 본 시스템의 알고리즘이 타당함을 확인할 수 있었다. (부록 MAIN MEMORY 참조)

		번지	명령
DCA	12	0	0 0 0 0
LOOP; TAD	13	1	0 0 0 8
ISZ	13	•	•
ISZ	14	•	•
JMP	LOOP	8	3 0 1 2
DCA	15	9	1 0 1 3
END		A	2 0 1 3
		B	2 0 1 4
		C	5 0 0 9
		D	3 0 1 5
		E	F 0 0 0
		•	•
		•	•
		12	0 0 0 0
		13	0 0 0 1
		14	F F 9 C
		15	0 0 0 0
		•	•

그림 6. 호스트 머신의 메모리 내용
Fig. 6. Main memory contents in host machine.

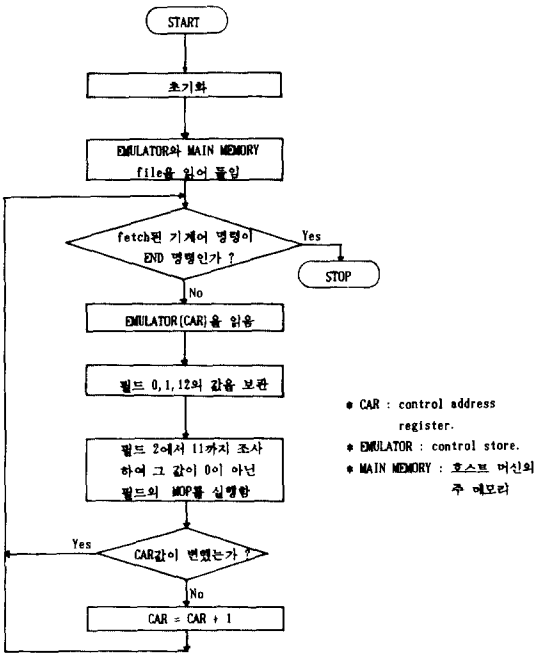


그림 7. 시뮬레이션의 흐름도.
Fig. 7. Flow chart for the simulation.

Ⅳ. 結 論

본 논문에서는 HLML로 쓰여진 에뮬레이터 프로그램이 컴파일러에 의해 IML로 주어졌을 때 기계독립적인 IML문 단위로 대체 우선순위와 변수의 이용 빈도수를 고려한 R A/D를 수행하고 변환표 등에 의해 마이크로코드화하는 변환 시스템을 구성하였다.

기계 독립적인 IML문 단위로 R A/D를 수행하는 것이 호스트 머신의 MOP 시퀀스 형태의 기계종속적인 중간 코드에서 수행하는 것보다 알고리즘 실행이 빠르며 컴파일과정에서 생긴 임시 변수로 인한 R A/D 를 줄일 수 있었다. 마이크로프로그램 가능한 가상 16bit 호스트 머신으로 PDP-8을 타겟트 머신으로한 IML을 변환 시스템에 적용하여 마이크로코드화하였고, 이 에뮬레이터 상에서 타겟트 머신의 간단한 테스트 프로그램이 실행되는 것을 C언어를 이용하여 시뮬레이션함으로써 제안된 시스템의 알고리즘이 타당함을 입증하였다.

변환표와 MI형식 및 필드 정보 등을 바꿈으로써 마이크로프로그램 가능한 다른 호스트 머신에서도 본 변환시스템을 간단히 이식가능하다.

보다 광역적인 R A/D, 마이크로코드의 최적화, HLML과 그 컴파일러의 설계등 마이크로프로그래밍 도구의 개발은 연구되어야 할 과제이다.

附 錄

MOP LIST

```
01. GPR
02. GPR
03. GPR
04. GPR
05. GPR
06. GPR
07. GPR
08. GPR
09. MBR          ←   MM(MAR)
10. MM(MAR)      ←   MBR
11. MAR          ←   PC
12. MAR          ←   MBR(AD)
13. MBR(AD)      ←   PC
14. MBR          ←   LITERAL
15. MBR          ←   IOREG
16. MBR          ←   GPR
17. MBR          ←   AC
18. MBR          ←   OPR
19. GPR          ←   MBR
20. GPR          ←   AC
21. GPR          ←   PS
22. AC           ←   AC          +       GPR
23. AC           ←   AC          −       GPR
24. AC           ←   AC          .AND.   GPR
25. AC           ←   AC          .OR.    GPR
26. AC           ←   AC          .XOR.   GPR
27. AC           ←   AC          .AND.   LITERAL
28. AC           ←   AC          +       LITERAL
29. AC           ←   GPR
30. AC           ←   PS
31. AC           ←   GPR         +       1
32. AC           ←   GPR         −       1
33. AC           ←   0
34. AC           ←   −1
35. AC           ←   1
36. AC           ←   LITERAL
37. AC           ←   MBR
38. AC           ←   .NOT.       AC
39. AC           ←   AC          +       1
40. AC           ←   AC          −       1
41. AC           ←   SHR(AC)
42. AC           ←   SHL(AC)
43. AC           ←   ROR(AC)
44. AC           ←   ROL(AC)
45. PC           ←   MBR(AD)
46. PC           ←   PC          +       1
47. PC           ←   PC          −       1
48. PC           ←   0
49. PS           ←   GPR
50. PS           ←   0
51. OPR          ←   MBR(OP)
52. IOREG        ←   DATA
53. IOREG        ←   MBR         ←
54. C=0          ⇨   CAR         ←       ADDRESS
55. C=1          ⇨   CAR         ←       ADDRESS
56. V=1          ⇨   CAR         ←       ADDRESS
57. N=0          ⇨   CAR         ←       ADDRESS
58. N=1          ⇨   CAR         ←       ADDRESS
59. Z=0          ⇨   CAR         ←       ADDRESS
60. Z=1          ⇨   CAR         ←       ADDRESS
61. CAR          ←   ADDRESS
62. CAR          ←   SBR
63. CAR          ←   MAPPER(OPR)
64. SBR          ←   CAR+1,      CAR     ←   ADDRESS
65. LITERAL/ ADDRESS
66. LITERAL/ ADDRESS
67. LITERAL/ ADDRESS
68. LITERAL/ ADDRESS
69. LITERAL/ ADDRESS
70. LITERAL/ ADDRESS
71. LITERAL/ ADDRESS
```

```
72. LITERAL/  ADDRESS
73. LITERAL/  ADDRESS
74. LITERAL/  ADDRESS
75. LITERAL/  ADDRESS
76. LITERAL/  ADDRESS
77. LITERAL/  ADDRESS
78. LITERAL/  ADDRESS
79. LITERAL/  ADDRESS
80. LITERAL/  ADDRESS
```

```
>>>>>>>><<<<<<<<<
> PDP 8 PARTIAL IML <
>>>>>>>><<<<<<<<<
```

```
B137  INF     RMOVE   MEM      PC      T. 001
0804          MOVE    T. 001           IR
0004          INC                      PC
B247  INSTDC  EXTR    OPCODE   IR      T. 002
0803          MOVE    T. 002           OPCD
0247          EXTR    PGEADR   IR      T. 003
0803          MOVE    T. 003           MAR
0147          RMOVE   MEM      MAR     T. 004
0804          MOVE    T. 004           MDR
0420          SLCT    OPCD     8
002B                           0       AND
002B                           1       TAD
002B                           2       ISZ
002B                           3       DCA
002B                           4       JMS
002B                           5       JMP
002B                           6       IOT
002B                           7       OPT
B344  AND     AND     ACCM     MDR     ACCM
000B          BRCH                     INF
B344  TAD     ADD     ACCM     MDR     ACCM
000B          BRCH                     INF
B137  ISZ     RMOVE   MEM      MAR     T. 005
0827          ADD     T. 005   1       T. 006
0831          WMOVE   T. 006   MAR     MEM
0147          RMOVE   MEM      MAR     T. 007
0820          COMP    T. 007   0
040A          CONDF   Z                L. 001
0004          INC                      PC
A00B  L. 001  BRCH                     INF
B342  DCA     WMOVE   ACCM     MAR     MEM
0004          CLR                      ACCM
000B          BRCH                     INF
B332  JMS     WMOVE   PC       MAR     MEM
 003          INC                      MAR
0404          MOVE    MAR              PC
000B          BRCH                     INF
B404  JMP     MOVE    MAR              PC
000B          BRCH                     INF
B000  IOT     NOOP
000B          BRCH                     INF
B000  OPT     NOOP
000B          BRCH                     INF
```

```
>>>>>>>>>><<<<<<<<<
> TRANSFOMATION TABLE <
>>>>>>>>>><<<<<<<<<
```

```
1   ADD     SRC 1   SRC 2    DEST
    AC      ←       SRC 1
    AC      ←       AC       +       SRC 2
    DEST    ←       AC
2   AND     SRC 1   SRC 2    DEST
    AC      ←       SRC 1
    AC      ←       AC       .AND.   SRC 2
    DEST    ←       AC
3   SUB     SRC 1   SRC 2    DEST
    AC      ←       SRC 1
    AC      ←       AC       -       SRC 2
    DEST    ←       AC
4   OR      SCR 1   SCR 2    DEST
    AC      ←       SRC 1
    AC      ←       AC       .OR.    SRC 2
    DEST    ←       AC
5   RMOVE   MEM     SRC      DEST
    MBR     ←       SRC
    MAR     ←       MBR
    MBR     ←       MM(MAR)
    DEST    ←       MBR
6   WMOVE   SRC     DEST     MEM
    MBR     ←       DEST
    MAR     ←       MBR
    MBR     ←       SRC
    MM(MAR) ←       MBR
7   INC                      SRC
    AC      ←       SRC      +       1
    SRC     ←       AC
8   DEC                      SRC
    AC      ←       SRC      -       1
    SRC     ←       AC
9   CLR                      SRC
    AC      ←       0
    SRC     ←       AC
10  MOVE    SRC             DEST
    AC      ←       SRC
    DEST    ←       AC
11  COMP    SRC 1   SRC 2
    AC      ←       SRC 1
    AC      ←       AC       -       SRC 2
12  COMP    SRC     LITERAL
    AC      ←       SRC
    AC      ←       AC       -       LITERAL
13  BRCH                     LABEL
    CAR     ←       LABEL
14  NEXT
    PC      ←       PC       +       1
15  NOT     SRC 1            DEST
    AC      ←       .NOT.    SRC 1
    DEST    ←       AC
16  XOR     SRC 1   SRC 2    DEST
    AC      ←       SRC 1
    AC      ←       AC       .XOR.   SRC 2
    DEST    ←       AC
17  SHL     SRC              DEST
    AC      ←       SRC
    DEST    ←       AC
18  SHR     SRC              DEST
    AC      ←       SRC
    AC      ←       SHR(AC)
    DEST    ←       AC
19  SET                      DEST
    AC      ←       -1
    DEST    ←       AC
20  MEMWREAD MEM    ADDR     DEST
    MBR     ←       ADDR
    MAR     ←       MBR
    MBR     ←       MM(MAR)
    DEST    ←       MBR
21  MEMWRITE SRC    ADDR     MEM
    MBR     ←       ADDR
    MAR     ←       MBR
    MBR     ←       SRC
    MM(MAR) ←       MBR
22  EXTR    OPCODE  SRC      DEST
    AC      ←       OPCODE
    AC      ←       AC       • AMD •  SRC
    DEST    ←       AC
23  SLCT    SRC     N
                    0        SUBR 1
                    1        SUBR 2
                    2        SUBR 3
    MBR     ←       SRC
    OPR     ←       MBR
    CAR     ←       MAPPER(OPR)
24  CONDF   Z                LABEL
    AC      ←       PS
    AC      ←       AC       .AND.   1
    Z=1     ⇨       CAR      ←       LABEL
25  CONDT   Z                LABLE
    AC      ←       PS
    AC      ←       AC       .AND.   1
    Z=0     ⇨       CAR      ←       LABEL
```

```
26      NOOP
        NOOP
27      XEQ                                     SUBROUTINE
        SBR             ←       CAR             +       1
        CAR             ←       SUBROUTINE
28      RET
        CAR             ←       SBR
29      CNDF    SRC     MASK    LABLE
        AC              ←       AC              .AND.   SRC
        Z=1             ⇨       CAR             ←       LABEL
30      CNDT    SRC     MASK    LABEL
        AC              ←       MASK
        AC              ←       AC              .AND.   SRC
        Z=0             ⇨       CAR             ←       LABEL
```

```
        ************
        * MOP SEQUENCE *
        ************
        MBR     ←       PC
        MAR     ←       MBR
        MBR     ←       MM(MAR)
        R 1     ←       MBR
INF     MBR     ←       R 1
        MAR     ←       MBR
        MBR     ←       MM(MAR)
        R 3     ←       MBR
        AC      ←       R 1             +       1
        R 1     ←       AC
INSTDC  AC      ←       OPCODE
        AC      ←       AC              .AND.   R 3
        R 4     ←       AC
        AC      ←       PGEADR
        AC      ←       AC              .AND.   R 3
        R 5     ←       AC
        MBR     ←       R 5
        MAR     ←       MBR
        MBR     ←       MM(MAR)
        R 6     ←       MBR
        MBR     ←       R 4
        OPR     ←       MBR
        CAR     ←       MAPPER(OPR)
AND     AC      ←       R 7
        AC      ←       AC              .AND.   R 6
        R 7     ←       AC
        CAR     ←       INF
TAD     AC      ←       R 7
        AC      ←       AC              +       R 6
        R 7     ←       AC
        CAR     ←       INF
ISZ     MBR     ←       R 5
        MAR     ←       MBR
        MBR     ←       MM(MAR)
        R 2     ←       MBR
        AC      ←       R 2
        AC      ←       AC              +       1
        R 2     ←       AC
        MBR     ←       R 5
        MAR     ←       MBR
        MBR     ←       R 2
        MM(MAR) ←       MBR
        MBR     ←       R 5
        MAR     ←       MBR
        MBR     ←       MM(MAR)
        R 2     ←       MBR
        AC      ←       R 2
        AC      ←       AC              −       0
        AC      ←       PS
        AC      ←       AC              .AND.   1
        Z=1     ⇨       CAR             ←       L. 001
        AC      ←       R 1             +       1
        R1      ←       AC
L. 001  CAR     ←       INF
DCA     MBR     ←       R 5
        MAR     ←       MBR
        MBR     ←       R 7
        MM(MAR) ←       MBR
        AC      ←       0
        R 7     ←       AC
        CAR     ←       INF
JMS     MBR     ←       R 5
        MAR     ←       MBR
        MBR     ←       R 1
        MM(MAR) ←       MBR
        AC      ←       R 5             +       1
        R 5     ←       AC
        AC      ←       R 5
        R 1     ←       AC
        CAR     ←       INF
JMP     AC      ←       R 5
        R 1     ←       AC
        CAR     ←       INF
IOT     NOOP
        CAR     ←       INF
OPT     NOOP
        CAR     ←       INF
```

```
        ********************************
        * OUTPUT AFTER MACHINE PROGRAM EXECUTION *
        ********************************
        ***MAIN MEMORY***
        0000    0000
        0001    0008
        0002    0000
        0003    0000
        0004    0000
        0005    0000
        0006    0000
        0007    0000
        0008    3012
        0009    1013
        000a    2013
        000b    2014
        000c    5009
        000d    3015
        000e    f000
        000f    0000
        0010    0000
        0011    0000
        0012    0000
        0013    0065
        0014    0000
        0015    13ba
        0016    0000
        0017    0000
        0018    0000
        0019    0000
        001a    0000
        001b    0000
        001c    0000
        001d    0000

***RESULT of SUMMATION***
    SUM= 13ba;hexadecimal number

***REGISTERS***
PC=0000, MAR=0000, AC=0000, PS=0001, OPR=000f
R1=000f, R2=0000, R3=f000, R4=f000, R5=0000
R6=0000, R7=0000, R8=0000, R9=0000, R10=0000
R11=0000, R12=0000, R13=0000, R14=0000, R15=0000
```

參 考 文 獻

[1] W.C. Hopkins et al., "Target-independent high-level microprogramming", *IEEE 18th Annual Microprogramming Workshop*, pp. 137-143, Dec. 1985.

[2] R.A. Muller et al., "A survey of resource

allocation methods in optimizing microcode compiliers", *IEEE 17th Annual Microprogramming Workshop*, pp. 285-295, Dec. 1984.

[3] 서대화, 윤석환, 방승량, "Microprogramming을 위한 tool개발에 관한 연구(microassembler)", 한국정보과학회 84춘계 학술발표집, 제11권 11호, pp. 291-298, 4월 1984년.

[4] 박장석, 윤석환, 박승규, "Target machine 에뮬레이션을 위한 마이크로 인스트럭션 설계에 관한연구", 한국정보과학회 84년 춘계학술발표집, 제11권 11호, pp. 271-280, 4월 1984년.

[5] P.Y. Ma and T.G. Lewis, "The design of a resource allocation scheme for microcode generation", *Euromicro Journal*, vol. 1, no. 5, pp. 277-286, 1983.

[6] D.J. Dewitt, "A machine independent approach to the production of horizontal micrrocode", *Ph. D. thesis, University of Michigan*, 1976.

[7] K. Malik, "Designing a high level microprogramming language", *Ph. D. thesis, Oregon Univ.*, 1979.

[8] P.Y. Ma, "Optimizing the mcirocode produced by a high level microprogramming language", *Ph. D. thesis, Oregon Univ.*, 1979.

[9] P.Y. Ma and T.G. Lewis, "Design of a machine independent optimizing system for emulator development", *ACM TOPLAS*, vol. 2, no. 2, pp. 239-262, April 1980.

[10] P.Y. Ma and T.G. Lewis, "On the design of a microcode compiler for a machine-independent high-level language", *IEEE Trans. on Software Eng.* vol. S-7, no. 3, pp. 261-274, May 1981.

[11] R.A. Muller et al., "Flow graph machine models in microcode synthesis," *IEEE 16th Annual Microprogramming Workshop*, pp. 159-167, Oct. 1983.

[12] R.A. Muller et al., "Global methods in the flow graph approach to retargetable microcode generation," *IEEE 17th Annual Microprogramming Workshop*, pp. 275-284, Dec. 1984.

[13] 최기호, 정하중, 문성일, 임인칠, "Emulator 개발을 위한 중간 microprogramming언어의 변환", 대한전자공학회 추계종합학술대회 논문집, vol. 8, no. 2, pp. 735-738, 11월 1985년.