Microcode 생성을 위한 Regrister 할당

°최 영 희, 최 기 호
광운대학

The Register Allocation for Microcode Generation

Yeong Hee Choi, Ki Ho Choi
Kwang Woon Univ.

요 약 본 논문에서는 Microcode 생성 System 에 있어서 기계 독립적인 중간언어 명령을 기계 종속적인 Microoperation 으로 변환과 Register full 상태일때 register 상태와 이용빈도수에의한 대체 우선순위에 따라 deallocation 하므로서 Memory access swapping 회수를 감소시킬수 있음을 보였다.

ABSTRACT This paper shows the translation of a machine independent intermediate language into machine dependent microoperations and the reduction of memory access swapping by the allocation/deallocation scheme which is based on the replacement priority table which is given by the register status and the frequency of use of variable when all registers are full.

1. 서 론

1951년 Wilkes 의 microprogramming 개념 소개 이후 IC 기술의 급속한 발달에 힘입어 Microprogram 가능한 Computer 가 많이 등장하였고, 또한 Data communication processor, OS 등 특정 목적의 Digital system의 제어에 손쉽게 microprogramming 하기위한 tool 개발에 많은 연구가 진행되고있다. microcode 생성을 위한 Software tool의 부족은 microcode양이 증가할때 많은 비용이들고 신뢰도가 저하되는 결과를 가져온다. (1)

Horizontal microcode 생성비용을 줄이기 위하여 High Level Microprogramming Language (HLML) 로 의도하는 program 을 작성하고 HLML 을 target machine 의 microprogram 으로 변환 하기위한 portable microcode compiler를 개발하는것이 바람직하다. 여러 machine 에 대한 portable 한 microprogram 을 생성하는데 사용할 수 있는 보다 일반적인 microprogram 생성 system구성에 대하여 많은 연구가 진행되고 있다.(2-7)

이에 대한 일반적인 구조는 그림1과 같다.

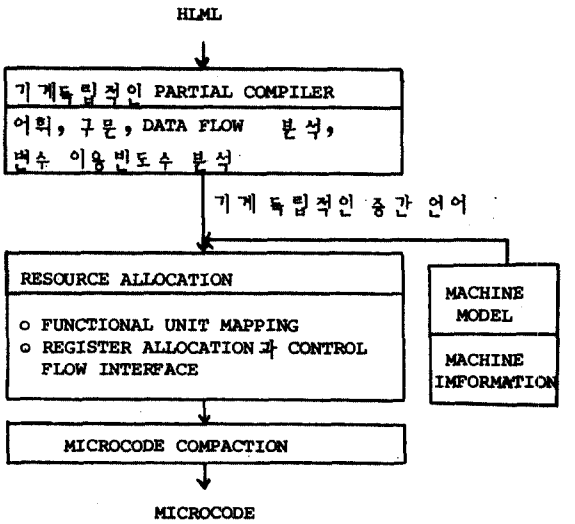


그림1. 마이크로코드 생성체제의 일반적인구조
General Structure of a microcode generation system

위의 구조중 register allocation문제는 최소수의 microinstruction 을 만들어내는데는microcode compaction 보다 큰 영향을 미치므로 아주 중요한 역활을 하고있다.(2)

Dewitt(3)의 변환 System 은 최적화와 register allocation 을 결합시킬수있도록 Code 생성을 Symbolic 변수level 로 끌어올리고 여러가지 microprogram 가능한 machine 을 설계할 수 있는 cantrol word model, code생성과 register allocation/deallocation의 결합 가능성을 제시했으나 너무 일반적이어서 실제 machine 에의적용이 어렵다.

Kim 과 Tan 은 (4) 광역적으로 변수의 track 을 이용하기 때문에 이에 대한 많은 정보를 요한다. 또한 여러종류의 register space 의 이용 가능성을 보이고 있으나 실제 어떻게 적용되는가는 분명치 않다.

Ma 와 Lewis 는 (2) 변수할당에 가중치를 주어 할당 우선 순위를 정하는 보다빠른 국소적인 방법을 이용하고 있다. 또한 MOP Scheduling 과 code 생성을 위해 machine model정보를 이용하여 PDP 11/40E 에 적용하였다. 그러나 기계 종속적인 MOP level 에서 resister를 할당하므로 algorithm 이 복잡하다.

Mueller (5,6,7)는 flow graph 를 이용한 microcode 생성기법과 register 할당의 결합 가능성에 대해 연구하고 있다.

본 논문은 기계 독립적인 중간 언어를 각 statement 단위로 Register allocation을 수행하고, 기계 독립적인 명령을 machine hardware unit 에 결합시킨다. 또한 register할당을 위한 deallocation 우선 순위 결정 방법을 논하였고 allocation/deallocation 수의 차이점을 비교한다.

2. Functional Unit 의 할당

Partial compiler에 의해 HLML 이 quadruple형태로 바뀌어진 IL(machine independent intermediate code)을 변환 table 을 이용하여 MOP list 형태로 변환한다.

IL 명령을 host machine 의 hardware unit 로 mapping 하는 MOP list 는 변환 table에 기억시킨다. 변환 table 을 이용한 예를 예1)에서 보였다.

예1) 뺄셈과 덧셈을 위한 명령

OP	I		O
SUB	SRC_1	SRC_2	DEST
ADD	SRC_1	SRC_2	DEST

microprogram 가능한 host machine의 MOP 변환 예를들면 아래와 같다.

o SUB의 경우

AC ← SRC1
AC ← AC-SRC2
DEST ← AC

o ADD의 경우

AC ← SRC1
AC ← AC+SRC2
DEST ← AC

3. Register할당 algorithm

Symbolic 변수 들을 register 에 할당하는 문제에대해 보다 작은 Step 으로 해를 구하기 위하여 변수 Weight mechanism 을 이용한다.

control flow interface 를 위해서 microoperation sequence 는 단일 입구와 단일출구를 갖는 straight line code(SLC) 의 set 로 나눈다.

register의 최종 상태 (FS) 는 각 SLC의 끝에서 register할당이 완료된 상태이고 SLC 의 초기상태 (IS) 는 그 SLC 에 comtrol flow가 있는 모든 SLC 의 FS 들의 합으로 정의된다.(8)

$$IS(i) = f(\bigcup_{k=1} FS(ik))$$

register 상태는 그 register 에 할당된 변수의 상태, 범위, type 을 나타낸다.

- 변수상태 : register 에 있는 변수 내용이 memory 의 내용과 틀리면 active, 같을 경우에는 passive.
- 변수범위 : 한 procedure 내에서만 access 가능한 변수는 local변수, 전체 program 에서 access 가능한 변수는 global 변수.
- 변수type : register 에 할당된후에 최소 한번 이용되고 다음에 다시 현 procedure 내에서 이용될 변수의 경우와, 현 procedure 내에서 다시 참조 되지 않는 변수의경우.

(1) SLC 내에서의 Register할당.

새로운 변수에 대해 free register를 이용할수 없을때까지 가능한 오랫동안 그 register 에 할당되어있는 변수를 그대로 유지시킨다.

새로운 한 변수를 할당하고자 할때free register 가 없는경우 register 에 이미 할당되어 있는 변수의 상태에 따라 우선순위를 표1과같이 정의하고 변수이용 빈도수에 따라 deallocate 한다. 대체 우선순위와 register상태를 이용한 register 할당 방법의 예를 예2)에서 보였다.

표1. 대체 우선 순위표

Replacement priority table

우선순위	상태		
1	현procedure에서 사용되지않는	passive	상태인 local변수
2	"	active	" local
3	"	passive	" global
4	"	active	" global
5	현procedure에서 사용될	passive	" local
6	"	passive	" gloval
7	"	active	" local
8	"	active	" global

예2) SLCi 에 할당 되어질 변수 VAR1,VAR2 가 있다고 ①과 같이 가정한다.

① iL 정의

```
VAR1 :  GLOBAL, USE
VAR2 :  GLOBAL, USE
  :

SLCi Starts
ADD VAR1 VAR2 DEST
SLCi  ends
```

② Register R1, R2 가 있다고 가정하고SLCi의

Register 할당이 시작되기전 Register상태

Register	변 수	상 태	범 위	TYPE
R1	VAR2	active	global	use
R2	VAR3	active	local	use

③ Register 할당과 변환후의 결과

```
MBR ← R2                    > memory write
MM(VAR3) ← MBR
MBR ← MM(VAR1)              > memory read
R2  ← MBR
AC  ← R2
AC  ← AC + R1
R2  ← AC
```

④ SLCi FS

Register	변 수	상 태	범 위	type
R1	VAR 2	active	global	use
R2	VAR 1	active	global	use

(2) SLC IS

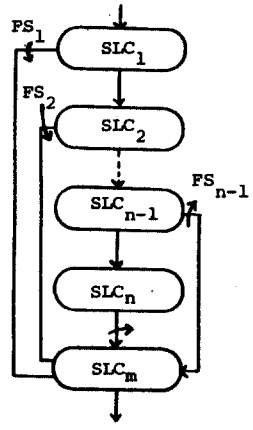


그림2. 전방분기의예
Example forward branch

SLC_m의 초기상태 IS_m은 그 SLC_m에 들어가기 바로 직전의 register 상태로서 정의된다.
그림2를보면 SLC_m 의 초기상태는 SLCi/i=1 to n의 FS에 의해 정의되며 현 SLC_m 에서의 register allocation/deallocation 을 수행하기 위한 근거가된다. n 개의 FS에서 Rj (j 번째 register) 가 갖고있는 모든변수가 같은경우Rj 에있는 IS_m의 변수와 범위는 FS에 따르고 상태는 모두 passive일때만 passive가되고 그 외에는 active 가된다. 반면에 만약 n 개의FS중 한변수이상 다른것과 틀리다면 IS_m에 있는Rj 는 free register 가 되어야한다.

예3) Register 는 R_1 과 R_2 를 사용한다고 가정하고 그림3에서 forward branch 에 대한 IS_4 를 결정한다 FS1, FS2, FS3는 다음과같이 주어진다.

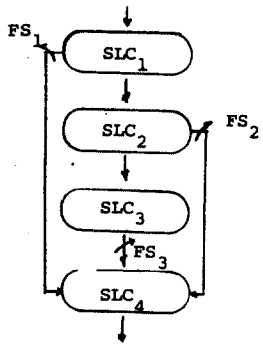


그림3. 예3의 전방분기
Forward branch of EX3

① FS1

Register	변 수	상 태
R1	VAR1	active
R2	VAR2	passive

② FS2

Register	변 수	상 태
R1	VAR1	passive
R2	VAR2	active

③ FS3

Register	변 수	상 태
R1	VAR1	passive
R2	VAR3	passive

④ FS4

Register	변 수	상 태
R1	VAR1	active
R2		

memory write 문은 MBR ← R2
MM(VAR2) ← MBR
와 같이 SLC2의 마지막에 삽입되어야하고 IS4 는 위표와 같다.

(3) Backward branch시 SLC의FS

그림4와같이 SLC_3 에서 SLC_2로 backward branch 할때 branch 문 전의 상태는 SLC_2 의 시작상태와 같아야한다.
결정되어야할 문제는 FS_3로 예4)와같이 IS2 NIS2,BS3 에의해서 결정된다. 이때 NIS 는 label 문이 실행시의 상태이고 BS 는 branch 문 바로직전의 상태이다.

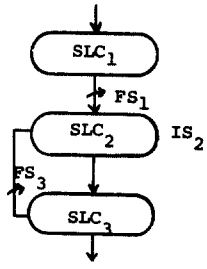
예4)① IS_2를 아래와 같이 가정한다.

Register	변 수	범 위	상 태
R1	VAR1	Global	active
R2	VAR2	Loacl	active

② SLC_2의 첫번째 문장이
Label1 MOVE VAR3 VAR1
일때의 NIS_2

Register	변 수	위 치
R1	VAR1	dest
R2	VAR3	source

```
MEMWRITE  R2  VAR2
MEMREAD   VAR3 R2
MOVE      R2   R1
```



③ SLC_2의 마지막 문장이 그림4. 예4의 후방분기
BRANCH Label1 Backward branch of EX4
일때의 branch 문 이전의 상태 BS_3는 다음과 같이 가정하면,

Register	변 수	상 태
R1	VAR4	active
R2	VAR3	active

④ SLC_3의 FS를 결정하기위해 NIS2와 BS3 에서 Register 에 있는 변수가 서로다르면 BS3 의 상태와 NIS2 의 변수위치를 고려하여 SLC2의 첫 문장이 실행될 수 있도록 FS3 를 결정한다. 본 예의 경우에는 R1의 변수가 서로다르고 상태와 위치가 active, dest 이므로 branch 문 이전에 memory write 가 삽입된다.

```
MEMWRITE  RI  VAR4
BRANCH    Lablel
```

Register	변 수	상 태
R1	VAR4	passive
R2	VAR3	active

(4) Register allocation algerithm

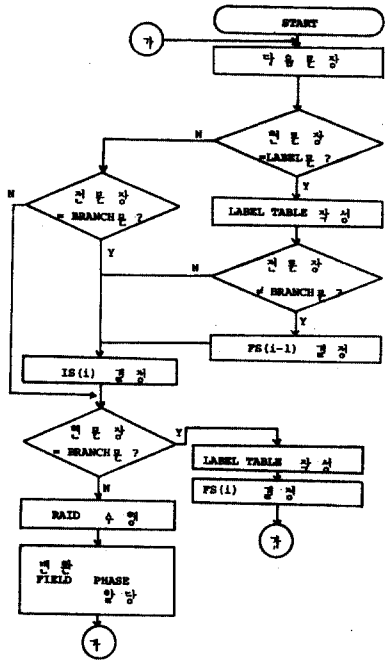


그림5. 레지스터 할당의 일반적인 구조
General structure of register allocation

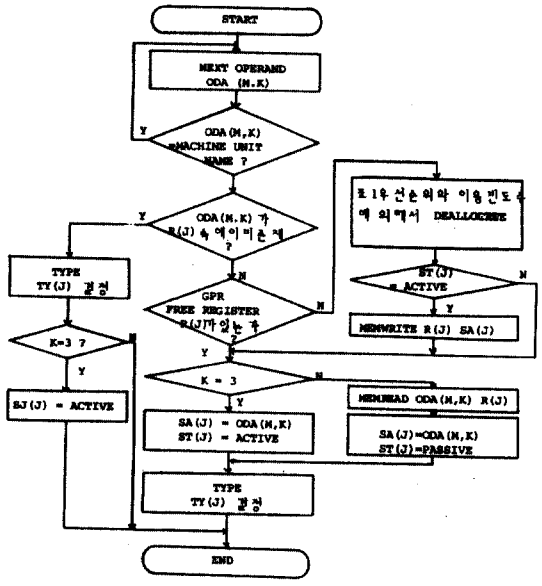


그림6. 레지스터 할당 알고리즘
Register A/D scheme algorithm

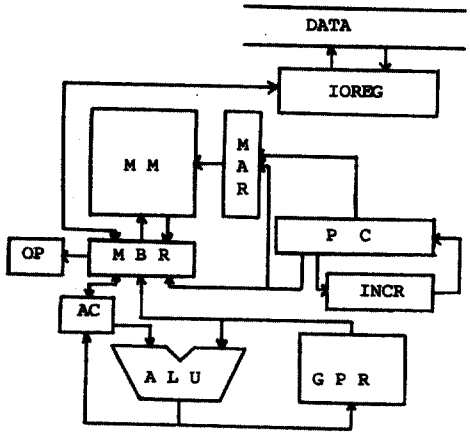


그림7. 가상의 마이크로 프로그램가능한기계
Virtual microprogrammable machine

4. 실험및 고찰

Host machine 으로는 가상의 microprogram가능한 machine 을 그림7과같이 구성했고 target machine 으로서 PDP8을 이용했다. (Host machine 의 Mop list 는 부록1참고) 한 HLML 로 쓰여진 program 이 partial compiler 에 의해서 IL 로 변환 되었다고 가정하고 Host machine 의 변환 tabel (부록 2참고)을 이용 MOP sequence로 변환과 register allocation algorithm 을 적용하였으며 이용가능한 register수의 변화와 다음과 같은 할당 방법의 변화에 따른 memory access swapping 회수를 비교해 보았다.

실험방법

방법1) 모든 변수들은 glabal 변수로 취급하며 deallocation 시 임의의 순서로 deallocation 한다. (대부분 현재의 compiler 에서 채택)

방법2) 대체 우선 순위표를 이용하고 우선 순위가 같을경우 임의의 순서로 dellocation 한다.

방법3) 방법2)에 변수의 이용빈도수를추가하여

우선순위가 같은경우 작은 빈도수를 갖는 변수를 deallocation 한다.

방법4) 방법2)에 우선순위가 같은경우 이후에 이용할 회수가 가장적은 빈도수를 갖는 변수를 deallocation 한다.

실험에 입력되는 test program (부록 표3 참고)은 13개의 SLC로 구성되어있고 7개의 global 변수, 6개의 임시 변수를 사용하였다. Register 의 수는 위의 4가지 방법에 의하여 3개의 register에서부터 10개의 register 까지 register 수를 증가 시켜가면서 memory access swapping 수를 비교하였다.

표2의 실험 결과와같이 7-8개의 register 면 emulation 하는데 충분하다는것을 알 수있고 4가지방법의 알고리즘에 대하여 load/store 명령수가 적은 순서대로 표시해본다.

방법3), 방법4), 방법2), 방법1)

방법1) 과 2)에 비해서 방법3)과 4)가 load/store 명령수가 감소를 보임을 PRIME 750 COMPUTER SYSTEM 을 이용한 표2의 실험결과를 통해서 입증할 수 있다.

표2. R A/D 수행후 Memory aceess swapping 수

Memory access swapping count after R A/D

Register수	방법1	방법2	방법3	방법4
3	28	22	17	20
4	26	20	15	19
5	21	20	14	18
6	17	15	13	17
7	15	13	11	11
8	10	10	10	10
9	10	10	10	10
10	10	10	10	10

5. 결 론

본 논문에서 가상의 microprogrm 가능한 machine 을 채택하여 기계독립적인 중간언어 명령을 변환 table에의해 기계종속적인 Microoperation 으로 변환하는 방법을 보였다. Host machine 의 변경시에는 새로운 machine에대한 table 의 재정의에 의해서 가능해지므로 portable 하다고 할 수있다. Register allocation시에 deallocation 의 결정에 있어서 실험결과와 같이 대체 우선순위가 같을 경우 빈도수를 추가한 방법이 좋은 성능을 얻을수 있음을 보였다.

참 고 문 헌

(1) S. Davidson and B.D. Shriver, "An overvicw of firmware engineering", camputer, Vol.11, pp. 21-33, May. 1978.

(2) P.Ma and T.G. Lewis, "Design of machine independent optimizing system for Emulator development", ACM TOPLAS, Vol.2, No.2, April 1980.

(3) DeWitt. D.J., "A machine independent approach to the production of horizontal microcode", Ph.D.thesis.University of Michigan. 1976.

(4) Kim.J. and Tan.C.J. "Register assignment algorithms for optimizing microcode compilers-Partl", Technical Report RC 7639, IBM T.J. Watson Research Center, May 1979.

(5) R.A.Mueller,J,Varghese,V.H Allan, "Global methods in the flow graph approach to retargetable microcode generation." Department of Computer Science Colorado state univ.

(6) R.A.Muller,M.R.Duda,S.M.O'Haire, "A survey of resource allocation methods in optimizing microcode compilers." Dapartment of Computer Science Colorado state univ.

(7) R.A.Muller,J.Varghese,"Flow graph machine models in microcode synthesis".

(8) R.P.Ma,T.G.Lewis,"The Design of a Resource Allocation Scheme for Microcode Generation". Euromicro Journal, Vol.11,No.5.PP. 277-286,1983.

부록1. Host machine microoperation list

```
      ---  MJP   LIST  ---
01.       GPR
02.       GPR
03.       GPR
04.       GPR
05.       GPR
06.       GPR
07.       GPR
08.       GPR
09.       M3R          <--  MM(MAR)
10.       MM(MAR)      <--  MBR
11.       M3R          <--  PROGCTR
12.       M3R          <--  PROGCTR
13.       M3R          <--  LITERAL
14.       M3R          <--  IDREG
15.       M3R          <--  GPR
16.       M3R          <--  AC
17.       GPR          <--  MBR
18.       GPR          <--  AC
19.       AC           <--  AC          +     GPR
20.       AC           <--  AC          -     GPR
21.       AC           <--  AC          .AND. GPR
22.       AC           <--  AC          .OR.  GPR
23.       AC           <--  AC          .XOR. GPR
24.       AC           <--  AC          -     LITERAL
25.       AC           <--  GPR
26.       AC           <--  .NOT.  GPR
27.       AC           <--  GPR         +     1
28.       AC           <--  GPR         -     1
29.       AC           <--  0
30.       AC           <--  -0
31.       AC           <--  1
32.       AC           <--  LITERAL
33.       AC           <--  MBR
34.       AC           <--  .NOT.  AC
35.       AC           <--  AC          +     1
36.       AC           <--  AC          -     1
37.       AC           <--  SHR(AC)
38.       AC           <--  SHL(AC)
39.       PROGCTR      <--  MBR(ADD)
40.       MAR          <--  ADDRESS
41.       AC 0: CARC   <--  ADDRESS
42.       AC 0: CARC   <--  ADDRESS
43.       AC           <--  ROR AC
44.       AC           <--  ROL AC
45.       IDREG        <--  LITERAL
46.       LITERAL      ADDRESS
47.       LITERAL      ADDRESS
48.       LITERAL      ADDRESS
49.       LITERAL      ADDRESS
50.       LITERAL      ADDRESS
51.       LITERAL      ADDRESS
52.       LITERAL      ADDRESS
53.       LITERAL      ADDRESS
```

부록2. 변환 table 의 일부

```
      ---  TRANSFORMATION  TABLE  ---
01.      ADD      SRC1     SRC2      DEST
         A        <--      SRC1
         A        <--      A         +        SRC2
         DEST     <--      A
02.      AND      SRC1     SRC2      DEST
         A        <--      SRC1
         A        <--      A         .AND.    SRC2
         DEST     <--      A
03.      SUB      SRC1     SRC2      DEST
         A        <--      SRC1
         A        <--      A         -        SRC2
         DEST     <--      A
04.      OR       SRC1     SRC2      DEST
         A        <--      SRC1
         A        <--      A         .OR.     SRC2
         DEST     <--      A
05.      RMOVE    MEM      SRC       DEST
         MBR      <--      SRC
         MAR      <--      MBR
         MBR      <--      MM(MAR)
         DEST     <--      MBR
06.      WMOVE    MEM      SRC       DEST
         MBR      <--      DEST
         MAR      <--      MBR
         MBR      <--      SRC
         MM(MAR)  <--      MBR
07.      INC      SRC
         A        <--      SRC       +        1
         SRC      <--      A
08.      DEC      SRC
         A        <--      SRC       -        1
         SRC      <--      A
09.      CLR      SRC
         A        <--      0
         SRC      <--      A
10.      MOVE     SRC                DEST
         A        <--      SRC
         DEST     <--      A
11.      COMP     SRC1     SRC2
         A        <--      SRC
         A        <--      A         -        SRC2
12.      COMP     SRC                LITERAL
```

부록3. PDP8 IL PROGRAM 의 일부

```
010101 INF        RMOVE     #1+PC                 #2+T.001
02                MOVE      #2-T.001              #1-IR
03                INC       #1+PC                 #1-PC
040202 INSTDC     EXTR      #1-IR                 #2+T.002
05                MOVE      #2-T.002              #1+OPCD
06                SLCT      #1-OPCD
07                                      C8
08                                      C0        SMRI
09                                      C1        SMRI
10                                      C2        SMRI
11                                      C3        SDCA
12                                      C4        SJMS
13                                      C5        SJMP
14                                      C6        SIO
15                                      C7        SOPT
160303 MRI        RMOVE     #1-MAR                #2+T.010
17                MOVE      #2-T.010              #1-MBR
18                SLCT      #1-OPCD
19                                      C3
20                                      C0        SAND
21                                      C1        STAD
22                                      C2        SISZ
230404 AND        AND       #1-MDR
24                          #1+ACCM               #1-ACCM
25                BRCH      BINF
260505 TAD        ADD       #1-MDR
27                          #1+ACCM               #1-ACCM
28
29
30                BRCH      BINF
310606 ISZ        RMOVE     #1+MAR                #2+T.011
32                ADD       #2-T.011              #2+T.012
33                WMOVE     #1+MAR
34                          #2-T.012
35                RMOVE     #1-MAR                #2+T.013
36                COMP      #2-T.013
37     07         BRCH      L.006
38                INC       #1+PC                 #1-PC
39     08 L.006   BRCH      BINF
400709 DCA        WMOVE     #1-MAR
41                          #1+ACCM               #1-ACCM
42                BRCH      BINF
430810 JMS        WMOVE     #1+MAR
44                          #1+PC
45                INC       #1+MAR                #1+MAR
46                          #1-MAR                #1-PC
47                BRCH      BINF
480911 JMP        MOVE      #1-MAR                #1-PC
49                BRCH      BINF
501012 IO         NOOP
51                BRCH      BINF
521113 OPT        NOOP1
53     PROGRAMEND
```

부록4. Register A/D 수행과정의 일부

```
MEMORY READ CNT=              1
                                                                                PAGE    1
{   1)  MEMORY READ CNT=            1
{   2)  0       INF          RMOVE
{   3)  0       INF          RMOVE        1    PC        #1+PC        PASSIVE    1
{   4)  0                    MOVE         2    T.001     #2+T.001     ACTIVE     2
{   5)  0                    MOVE         2
{   6)  0                    INC          2    IR        #1-IR        ACTIVE     NONE
{   7)  0                    INC          1    PC        #1+PC        PASSIVE    1
{   8)  0       INSTDC       EXTR         1    PC        #1-PC        ACTIVE     1
{   9)  0       INSTDC       EXTR         2    IR        #1-IR        ACTIVE     3
{  10)  0                    MOVE         3    T.002     #2+T.002     ACTIVE     3
{  11)  0                    MOVE         3                                      2
{  12)  0                    SLCT         3    OPCD      #1+OPCD      ACTIVE     NONE
{  13)                                    3    OPCD      #1-OPCD      ACTIVE     1
{  14)                              4                                            3
{  15)                              4
{  16)  D=        3                 6
{  17)  PR(D)=           6PR(I)=          6R(I)=PC
{  18)  RES(I)=PC
{  19)  VARTAB(J)=PC            VARCNT=         8.00000
{  20)  PR(D)=           6PR(I)=          6R(I)=IR
{  21)  RES(I)=IR
{  22)  VARTAB(J)=IR            VARCNT=         2.00000
{  23)  PR(D)=           6PR(I)=          6R(I)=OPCD
{  24)  RES(I)=OPCD
{  25)  VARTAB(J)=OPCD          VARCNT=         3.00000
{  26)  VARCOM=            1
{  27)  VARCOM=            2
{  28)  VARCOM=            3
{  29)  D=       2
{  30)  MEMORY WRITE COUNT=            1
{  31)  MEMORY READ CNT=          2
{  32)  0       MRI          RMOVE        2    MAR       #1-MAR       PASSIVE    3
{  33)
{  34)                             4
{  35)                             3
{  36)  D=        2                6
{  37)  PR(D)=           3PR(I)=          3R(I)=MAR
{  38)  RES(I)=MAR
{  39)  VARTAB(J)=MAR           VARCNT=        10.0000
{  40)  0       MRI          RMOVE
{  41)  0                    MOVE         2    T.010     #2+T.010     ACTIVE     2
{  42)  0                    MOVE         2                                      NONE
{  43)  0                    SLCT         2    MDR       #1-MDR       ACTIVE     3
{  44)  0       AND          AND          3    OPCD      #1-OPCD      ACTIVE     3
{  45)                                    2    MDR       #1-MDR       ACTIVE     3
{  46)                             4
{  47)                             6
{  48)  D=        3                6
{  49)  PR(D)=           6PR(I)=          6R(I)=PC
{  50)  RES(I)=PC
{  51)  VARTAB(J)=PC            VARCNT=         8.00000
{  52)  PR(D)=           6PR(I)=          6R(I)=MDR
{  53)  RES(I)=MDR
{  54)  VARTAB(J)=MDR           VARCNT=         3.00000
{  55)  PR(D)=           6PR(I)=          6R(I)=OPCD
```