論文 85-22-6-19

# MC68000μP의 命令語디코오딩 機能에 관한 試驗알고리즘

# (A Test Algorithm for Instruction Decoding Function of MC 68000 μP)

金 鍾 勳*, 安 光 善**

(Jong Hoon Kim and Gwang Seon Ahn)

要 約

LSI/VLSI의 製造技術이 발달함에 따라 마이크로프로세서(μP)의 機能試驗에 많은 시간이 소요되고 있다. 論文에서는 MC 68000μP를 對象으로 機能試驗이 複雜하게 되는 要因인 命令語디코오딩 機能에 대한 効率的인 試驗方法을 제안하였다. 이를 위하여 使用者측에 제공된 命令語디코오딩의 情報인 operation word를 分析하고 그 bit形態에 따라 命令語를 代表命令語와 所屬命令語로 구분하여 命令語디코오딩 故障檢出에 필요한 最少의 試驗명령어雙을 選定하였으며 이들에 대한 試驗處理方法을 論하였다. 本 試驗알고리즘을 69個의 基本命令語에 대해 수행시킨 결과 332個의 試驗命令語雙을 구할 수 있었다.

**Abstract**

The functional testing of microprocessor comes to be time - consuming task with the progress of technology of LSI/VLSI. In this paper, we present an efficient method to test instruction decoding function of MC 68000 that is the reason of complicated functional testing. This method is based on the analysis of operation word that is instruction decoding information available to user with microprocessor's manual. The instruction is partitioned into representative instructions and party instructions. Then 332 minimum test instruction pairs are chosen from 69 basic instructions for detecting of instruction decoding function faults and test procedure for these is discussed.

## I. 序 論

最近 LSI/VLSI에 대한 製造技術의 발달로 因하여 chip의 機能과 內部回路가 複雜하고 多樣해졌다. 그 대표적인 例가 μP인데 이것의 正常動作을 보장하기 위해서는 故障의 有無를 試驗해야 할 必要가 있다. 試驗에는 構造試驗과 機能試驗의 두가지가 있으나 産業上의 秘密로 生産者측에서 제공되는 情報가 극히 制限되어 있으므로 概念的 次元의 機能試驗을 主로 하고 있다[1,2,3,4].

1978年 Thatte등[1]은 μP의 機能을 S-graph모델로써 記述하고 체계적인 機能試驗 알고리즘을 제안하였다. 그러나 이 알고리즘中 命令語디코오딩 機能試驗은 가능한 모든 命令語雙을 對象으로 하였기 때문에 試驗의 複雜度가 매우 높다. 例를 들어 命令語 数가 n인 μP의 경우 Thatte 알고리즘을 적용하면 試驗對象이 되는 명령어雙의 次数가 $n^2$정도 되며 이것은 16-bit, 32-bit μP의 出現과 함께 심각한 問題가 되고 있다.

1983年 Saluja등[6]은 試驗의 複雜度를 낮추기 위하여

*正會員, 東義工業專門大學 電子科
(Dept. of Electron. Dong Eui Tech. Junior College)
**正會員, 慶北大學校 電子工學科
(Dept. of Electron. Eng., Kyung Pook National Univ.)
接受日字 : 1985年 6月 15日

命令語의 實行時間에 대한 情報를 근거로 令命語디코오딩 機能故障을 檢出하는 알고리즘을 제안하였으나 이 方法은 $\mu$P의 制御信号를 측정하기 위한 hardware가 불가피하게 요구된다.以後 1984年 Brahme등[7]은 命令語의 $\mu$-order 構成을 分析하여 simple fault를 定義하고 이를 바탕으로 code word를 이용하여 効率的인·命令語디코오딩 機能試驗 알고리즘을 제안하였는데 이것은 $\mu$-order의 基本的 構成이 동일한 命令語 상호간의 디코오딩 機能故障은 檢出하기 곤란하다.

本 論文에서는 MC68000 $\mu$P의 命令語디코오더의 stuck-at 故障을 對象으로 命令語디코오딩의 기본情報가 되는 operation word를 分析하여 試驗對象 명령어雙의 数를 줄임으로써 複雜度가 낮은 効率的인 命令語 디코오딩 機能試驗을 제안한다.

## Ⅱ. 命令語디코오딩 機能故障

$\mu$P의 機能은 命令語디코오딩, 레지스터디코오딩, 데이터貯藏, 데이터操作, 데이터 傳送등으로 구분되며 試驗도 이를 各各에 대해 개별적으로 遂行된다.[5] 한편 機能試驗에는 試驗의 効率性이 강조되고 있는데 이들 중 특히 命令語디코오딩 機能故障에 대한 試驗이 가장 複雜하고 試驗時間이 많이 소요되고 있다.[5,6,7]

$\mu$P의 命令語디코오딩 機能을 $f_D$라고 하면 모든 命令語集合 $I=\{i_1, i_2, \cdots, i_m\}$에 대해 $f_D(i_j)=i_j\ (1\leq j\leq n)$으로 표현된다. 그러나 만일 實行시키려는 命令語 $i_j$를 디코오딩할 때 命令語디코오더에 stuck-at 故障이 존재한다면 이 디코오더의 動作은 그 세부적 遂行事項과는 관계없이 다음 세가지 故障効果로 나타낼 수 있다.

i) $f(i_j/\phi)$ : 命令語 $i_j$가 實行되지 않는 故障

ii) $f(i_j/i_k)$ : 命令語 $i_j$대신에 다른 命令語 $i_k$가 實行되는 故障

iii) $f(i_j/i_k^1+i_k^2+\cdots+i_k^m)$ : 命令語 $i_j$의 命令語形式이 m個 word일 때 $i_j$대신에 다른 命令語 sequence 가 實行되는 故障

上記 故障중에서 $f(i_j/\phi)$는 operation word에 할당되지 않은 bit값때문에 나타나는 것으로 이것을 檢出하기 위해서는 命令語 $i_j$의 destination register 속에 적당한 operand값을 초기설정한 후 $i_j$를 實行시켜 그 結果가 올바른가를 확인하면 된다. 그리고 $f(i_j/i_k)$는 source register속에 命令語 $i_j$와 $i_k$의 實行結果를 구별할 수 있는 적당한 operand를 넣어두고 $i_j$를 실행시킨 후 그 結果를 확인하면 檢出할 수 있다.

$f(i_j/i_k^1+i_k^2+\cdots+i_k^3)$도 $f(i_j/i_k)$와 비슷한 방식으로 檢出해야 되지만 이들 命令語를 모두 고려한 operand의 결정이 곤란하므로 별도로 試驗하지는 않는다. 그러나 $i_j$와 命令語 sequence의 實行結果가 일치할 可能性은 거의 없으므로 이 故障은 $f(i_j/i_k)$의 試驗時에 대부분 檢出된다. 그러므로 命令語 $i_j$에 대한 命令語 디코오딩 機能故障은 $f(i_j/i_k)$이 主가 된다.

$f(i_j/i_k)$를 檢出하기 위하여 $\mu$P의 모든 가능한 命令語의 雙을 試驗對象으로 고려해야 한다. 그러나 MC68000 $\mu$P의 경우 命令語의 operation word 形態가 일정한 규칙을 가지고 있으므로 이것을 分析하여 그 bit 형태에 따라 命令語를 분류함으로써 試驗對象 명령어雙의 数를 훨씬 줄일 수 있다.

定義 1 : 命令語 $i_j$의 命令語디코오딩 機能故障을 檢出하기 위하여 選定된 명령어雙 $i_j/i_k$를 試驗명령어雙 그리고 $i_k$를 點檢命令語라고 한다.

## Ⅲ. 故障檢出을 위한 試驗명령어 雙

MC68000 $\mu$P는 69個의 基本命令語들로 構成되어 있으며 이들의 命令語形式은 1個 word에서 5個 word의 길이를 갖는다. 그러나 實行되는 操作은 命令의 最初의 word 즉 operation word로 지정되고 그 나머지의 word는 operand를 나타내고 있다. 따라서 命令語 디코오딩 機能故障에 對한 試驗對象은 operation word로 국한시킬 수 있으므로 이것을 분석함으로써 試驗명령어쌍의 数를 줄인다.

### 1. 命令語의 operation word 分析

MC68000 $\mu$P의 operation word는 operation code와 address mode로 構成되어 있다.[8] 그리고 各 operation code에는 operation의 크기, 形態 및 條件을 규정하는 field들이 포함되어 있으며 그 길이도 命令語에 따라 다르다. 그러나 모든 命令語는 operation word의 bit번호 12~15에 해당하는 operation code 즉 代表 code에 의해 분류할 수 있으며 이들은 다시 나머지 operation code와 여러 field를 포함하고 있는 bit 번호 0~11의 bit 즉 所屬 code에 의해 특징적으로 구별된다. 表 1은 MC68000 $\mu$P의 基本命令語들을 代表 code에 따라 분류한 것이며 試驗을 위하여 다음의 定義들이 필요하다.

定義 2 : MC68000 $\mu$P의 全 命令語 集合 I는 다음과 같은 성질을 갖는 部分命令語 集合 $I_s\ (0\leq S\leq 15)$으로 분할할 수 있으며 各 $I_s$에 속해있는 命令語들을 $I_s$의 所屬命令語라고 한다 :

i) $\bigcup_{s=0}^{15} I_s = I$

ii) $I_s=\{i_s^w | i_s^w (\in I)$는 동일한 代表 code, $0\leq w\leq k\}$. 여기서 $k=\max\{||I_s||\}$이며 $||I_s||$는 部分命令語 集合 $I_s$의 원소数.

定義 3 : 各 部分命令語 集合 $I_s\ (0\leq S\leq 15)$으로 부

표 1. MC68000μP의 오퍼레이션 코오드 맵
Table 1. Operation code map for MC68000μP.

| set number | representative code (bit 15~12) | operation | basic Instruction |
|---|---|---|---|
| $I_0$ | 0 0 0 0 | bit manipulation/immediate /move peripheral data | $BTST_{ST}$, $BTST_{DY}$, $BSET_{ST}$, $BCHG_{ST}$, $BCHG_{DY}$, $BCLY_{ST}$, $BCLR_{DY}$, MOVEP, ORI, ANDI, SUBI, ADDI, EORI, CMPI, $BSET_{DY}$ |
| $I_1$ | 0 0 0 1 | move byte | $MOVE_B$ |
| $I_2$ | 0 0 1 0 | move long | $MOVE_L$ |
| $I_3$ | 0 0 1 1 | move word | $MOVE_W$ |
| $I_4$ | 0 1 0 0 | mi scellaneous | NEGX, MOVE form SR, CLR, NEG, MOVE to CCR, NOT, MOVE to SR, NBCD, PEA, SWAP, EXT, MOVEM, TST, TAS, TRAP, LINK, UNLK, RESET, MOVE, USP, NOP, STOP, RTE, RTS, TRAPV, RTR, JSR, JMP, CHK, LEA |
| $I_5$ | 0 1 0 1 | add quick/subtract quick/ set conditionally/decrement conditionally | ADDQ, SUBQ, $S_{cc}$, $DB_{cc}$ |
| $I_6$ | 0 1 1 0 | branch conditionally | $B_{cc}$ |
| $I_7$ | 0 1 1 1 | move quick | MOVEQ |
| $I_8$ | 1 0 0 0 | OR/divide/subtract decimal | OR, DIVU, DIVS, SBCD |
| $I_9$ | 1 0 0 1 | subtract/subtract extended | SUB, SUBX |
| $I_{10}$ | 1 0 1 0 | (unassigned) | $\phi$ |
| $I_{11}$ | 1 0 1 1 | compare/exclusive OR | CMP, CMPM, EOR |
| $I_{12}$ | 1 1 0 0 | AND/multiply/add decimal/ exchange | AND, MULU, MULS, ABCD, EXG |
| $I_{13}$ | 1 1 0 1 | add/add extended | ADD, ADDX |
| $I_{14}$ | 1 1 1 0 | shift/rotate | $ASd_{RS}$, $ASd_{MS}$, $LSd_{RS}$, $LSd_{MS}$ $ROd_{RS}$, $ROd_{MS}$, $ROXd_{RS}$, $ROXd_{MS}$ |
| $I_{15}$ | 1 1 1 1 | (unassigned) | $\phi$ |

註) ST : static, DY : dynamic, B : byte, L : long, W : word, d : right or left
RS : yegister shift, MS : memory shift, RR : register rotate, MR : memory rotate

터 代表 code의 디코오딩 機能故障을 點檢하기 위하여 선택된 1個의 命令語 $i_s$를 그 $I_s$의 代表命令語라 하고 이들로 이루어진 集合을 I의 選擇集合 $I_c$라고 한다.

定義 3에 의해 구해진 選擇集合은 그 원소인 代表命令語 상호간의 命令語디코오딩 機能試驗을 통하여 operation word의 bit번호 12~15에 해당하는 代表 code에 대한 故障 여부의 판단에 이용한다.

2. 試驗명령어雙의 選定

MC68000 μP의 命令語의 operation word에는 各 命令語를 특정지우기 위한 bit 값이 존재한다. 만일 命令語디코오딩 機能故障이 존재한다면 그 原因은 bit값이 틀리게 디코오딩되었기 때문이다. 그러므로 命令語 디코오더의 stuck-at故障하에서는 各 命令語 operation word의 모든 가능한 1-bit差 명령어雙의 集合은 모든 가능한 命令語디코오딩 故障을 포함하고 있다. 한편 이러한 1-bit差 명령어雙의 数는 많으나 命令語디코오딩의 順序를 고려한다면 命令語를 代表code에 따라 분할하여 代表命令語와 所屬命令語에 대한 試驗명령어雙을 구분하여 選定함으로써 命令語디코오딩 機能試驗에 필요한 試驗명령어雙의 数를 최소로 할 수 있다.

定義 4 : 命令語 $i_j$에 대한 命令語디코오딩 機能故障을 檢出하기 위하여 $i_j$의 operation word 중에서 試驗對象으로 정한 bit를 試驗bit라고 한다.

定義 5 : 命令語 $i_j$와 $i_k$의 operation word를 비교하여 各 대응되는 bit들 중에서 r個가 서로 다른 경우 이 命令語의 雙 $i_j/i_k$를 r-bit差 명령어雙이라고 한다.

定理 1 : 命令語디코오더의 stuck-at故障하에서 모든

1－bit差 명령어雙들로 構成된 試驗명령어雙 $i_j/i_k$ 의 集合은 2－bit差 以上의 명령어雙 상호간의 디코오딩 機能故障을 포함한다.

證明：命令語 形式이 16－bit인 $\mu$P의 경우 各 bit 번호를 $C_{15}$, $C_{14}$, …, $C_0$ 라고 하면 命令語디코오딩 機能 $f_D$는 다음과 같이 Boolean 函數로 표현된다.

$$f_D(C_{15}, C_{14}, \cdots, C_0) = \sum_{n=0}^{2^{15}} \tau(\alpha_{15}, \alpha_{14}, \cdots, \alpha_0) \cdot C_{15}^{\alpha_{15}} \cdot C_{14}^{\alpha_{14}} \cdots\cdots C_0^{\alpha_0}$$

여기서 $\alpha_i (0 \leq i \leq 15)$는 bit Ci의 값이고 $\tau(\alpha_{15}, \alpha_{14}, \cdots, \alpha_0)$는 boolean expression이다. 만일 code가 $\underbrace{0\ 0 \cdots 0}_{15\text{bit}}$인 命令語 $i_0$를 디코오딩하고자 한다면 正常動作일 때 다음 式에서 첫번째 項 $C_{15}, C_{14}, \cdots, C_0$ 에 해당하는 出力이 active 된다.

$$i_0 = (\overline{0} \cdot \overline{0} \cdots \overline{0}) \cdot \overline{C}_{15} \cdot \overline{C}_{14} \cdots \overline{C}_0 + (\overline{0} \cdot \overline{0} \cdots \overline{0} \cdot \overline{1}) \cdot \overline{C}_{15} \cdot \overline{C}_{14} \cdots \overline{C}_1 \cdot C_0 + (\overline{0} \cdot \overline{0} \cdots \overline{0} \cdot \overline{1} \cdot \overline{0}) \cdot \overline{C}_{15} \cdot \overline{C}_{14} \cdots \overline{C}_2 \cdot C_1 \cdot \overline{C}_0 + (\overline{0} \cdot \overline{0} \cdots \overline{0} \cdot \overline{1} \cdot \overline{1}) \cdot \overline{C}_{15} \cdot \overline{C}_{14} \cdots \overline{C}_2 \cdot C_1 \cdot C_0 + \cdots + (\overline{1} \cdot \overline{1} \cdots \overline{1}) \cdot C_{15} \cdot C_{14} \cdots C_0$$

이 경우 2－bit差 명령어雙 $i_0/i_3$에 대한 디코오딩 機能故障 즉 $f(i_0/i_3)$가 존재한다면 윗 式에서 첫번째 項대신에 네번째 項의 出力이 active된다는 것을 의미하고 이것은 bit $C_1$ 및 $C_0$가 各各 stuck-at 1 되었기 때문이다. 따라서 이 故障은 1－bit差 명령어雙인 $i_0/i_1$ 혹은 $i_0/i_2$에도 반드시 포함되어 있다. 이러한 性質은 모든 가능한 bit조합에 대해 적용되므로 모든 1－bit差 명령어雙은 모든 2－bit差 以上의 명령어雙의 디코오딩 機能故障을 포함하고 있다. Q. E. D.

定理 1에 의하여 모든 가능한 1－bit差 명령어雙을 選定하고 이것들에 대해 디코오딩 機能試驗을 행하면 r－bit差 명령어雙의 機能故障도 檢出된다.

MC68000 $\mu$P의 operation word에는 各 命令語의 動作을 더욱 세분하고 또한addressing mode를 결정하기 위해 다음의 field들이 포함되어 있으며 이 field의 값은 試驗명령어쌍의 選定過程에서 결정될 수 있다.

size field, effective address field, register field, op-mode field, data field, condition field, register / memory field, immediate/register field

定義 6：命令語 $i_j$에 대한 試驗명령어雙 $i_j/i_k$의 選定過程에서 1－bit差인 條件을 만족하기 위해 결정된 field의 값을 $i_j/i_k$의 試驗條件이라 한다.

1) 代表命令語에 대한 試驗명령어雙

MC68000 $\mu$P의 operation word중 代表code에 대한 디코오딩 故障의 여부는 選擇集合 $I_c$의 원소인 代表命令語 상호간에 대한 試驗명령어雙을 選定하여 이것을 試驗함으로써 판단할 수 있다.

試驗에 적합한 選擇集合을 構成하기 위하여 다음 事項을 고려한다.

i) 部分命令語 集合 $I_S (0 \leq S \leq 15)$의 所屬命令語가 1個인 경우 代表命令語로 選擇한다.

ii) 이미 選擇된 代表命令語와의 사이에 所屬code에 대한 star product[9] 연산時 $\phi$이 되지 않는 所屬命令語를 선택한다. 단, 연산時 所屬code중 값이 결정되어 있지 않은 bit는 don't care X로 취급된다.

部分命令語 集合 $I_S$중 $I_{10}$, $I_{15}$는 할당되어있지 않으므로 選擇集合 構成시에 원소 $i_0$, $i_{15}$는 없다. 그러므로 命令語 $i_j$의 各 試驗bit를 試驗하기 위한 1－bit差 點檢命令語가 없을 수도 있으며 이런 경우는 $f(i_j/\phi)$에 해당한다.

構成한 選擇集合 $I_c$는 $I_c = \{BTST_{static}, MOVE_{byte}, MOVE_{long}, MOVE_{word}, NBCD, ADDQ, B_{cc}, MOVEQ, OR, SUB, CMP, AND, ADD, ASL_{register\ shift}\}$ 이며 代表命令語에 대한 試驗命令語雙의 選定은 選擇集合 $I_c$의 원소 상호간에 이루어지고 各 代表命令語의 代表code가 한번씩 試驗bit로 정해진다.

選定알고리즘을 SPARKS語[11]로 표현하면 다음과 같다.

Algorithm 1

//choose test instruction pairs for chosen set $I_c$//

```
procedure Test Instruction Pairs of Representative
Instruction declare
  *      →operator of star product
  QINST  →result of star product
  TB     →test bit
  TC     →test condition
1  begin
2    for j=0 to 15 do
3      take ij : //test instruction//
4      for TB=12 to 15 do
5        for k=0 to 15 do
6          take ik : //detect instruction//
7          if ij=ik then go to 10 :
8          QINST←ij * ik :
9          if only TB of QINST is φ then go to 12 :
10       end :
11       print ij/φ : go to 14 :
12       print ij/ ik : //test instruction pair//
13       print TC : //test condition//
14     end :
15   end :
16 end :
```

알고리즘 1의 수행결과 表 2에 보인바와 같이 14個

표 2. 代表命令語에 대한 試驗명령어雙
Table 2. Test instruction pairs for representative instructions.

| representative code | test instruction pair ($i_j/i_k$) | test condition of field | |
|---|---|---|---|
| 0 0 0 0 | $BTST_{ST}$/OR | register | 1 0 0 |
| | | OP-mode | 0 0 1 |
| | $BTST_{ST}$/NBCD | $\phi$ | |
| | $BTST_{ST}$/$MOVE_{L}$ | register | 1 0 0 |
| | | OP-mode | 0 0 0 |
| | $BTST_{ST}$/$MOVE_{B}$ | register | 1 0 0 |
| | | OP-mode | 0 0 0 |
| 0 0 0 1 | $MOVE_{B}$/SUB | $\phi$ | |
| | $MOVE_{B}$/ADDQ | D-mode | 0 X X |
| | $MOVE_{B}$/$MOVE_{W}$ | $\phi$ | |
| | $MOVE_{B}$/$BTST_{ST}$ | D-register | 1 0 0 |
| | | D-mode | 0 0 0 |
| 0 0 1 0 | $MOVE_{L}$/$B_{CC}$ | $\phi$ | |
| | $MOVE_{L}$/$BTST_{ST}$ | D-register | 1 0 0 |
| | | D-mode | 0 0 0 |
| | $MOVE_{L}$/$MOVE_{W}$ | $\phi$ | |
| | $MOVE_{L}$/ $\phi$ | $\phi$ | |
| 0 0 1 1 | $MOVE_{W}$/CMP | $\phi$ | |
| | MOVEW/MOVEQ | D-mode | X X 0 |
| | $MOVE_{W}$/$MOVE_{B}$ | $\phi$ | |
| | $MOVE_{W}$/$MOVE_{L}$ | $\phi$ | |
| 0 1 0 0 | NBCD/AND | register | 1 0 0 |
| | | OP-mode | 0 0 0 |
| | NBCD/$BTST_{ST}$ | data | 0 0 0 |
| | NBCD/$B_{CC}$ | Condition | 1 0 0 0 |
| | | DIS | 00XXXXXX |
| | NBCD/ADDQ | data | 1 0 0 |
| | | size | 0 0 0 |
| 0 1 0 1 | ADDQ/ADD | OP-mode | 0 X X |
| | ADDQ/$MOVE_{B}$ | D -mode | 0 X X |
| | ADDQ/MOVEQ | $\phi$ | |
| | ADDQ/NBCD | register | 1 0 0 |
| | | data | 00XXXXXX |
| 0 1 1 0 | $B_{CC}$/$ASL_{RS}$ | Condition | XXX 1 |
| | $B_{CC}$/$MOVE_{L}$ | $\phi$ | |
| | $B_{CC}$/NBCD | Condition | 1 0 0 0 |
| | | DIS | 00XXXXXX |
| | $B_{CC}$/MOVEQ | Condition | XXX 0 |

| representative code | test instruction pair ($i_j/i_k$) | test condition of field | |
|---|---|---|---|
| 0 1 1 1 | MOVEQ/$MOVE_{W}$ | D-mode | 0 X X |
| | MOVEQ/ADDQ | $\phi$ | |
| | MOVEQ/$B_{CC}$ | Condition | X X X |
| | MOVEQ/$B_{CC}$ | Condition | X X X |
| | MOVEQ/$\phi$ | $\phi$ | |
| 1 0 0 0 | OR/$BTST_{ST}$ | register | 1 0 0 |
| | | OP-mode | 0 X X |
| | OR/AND | $\phi$ | |
| | OR/SUB | $\phi$ | |
| | OR/$\phi$ | $\phi$ | |
| 1 0 0 1 | SUB/$MOVE_{B}$ | $\phi$ | |
| | SUB/ADD | $\phi$ | |
| | SUB/CMP | $\phi$ | |
| | SUB/OR | $\phi$ | |
| 1 0 1 1 | CMP/$MOVE_{W}$ | $\phi$ | |
| | CMP/SUB | $\phi$ | |
| | CMP/$\phi$ | $\phi$ | |
| 1 1 0 0 | AND/NBCD | register | 1 0 0 |
| | | OP-mode | 0 0 0 |
| | AND/OR | $\phi$ | |
| | AND/$ASL_{RS}$ | OP-mode | 1 X X |
| | | EA-mode | X 0 0 |
| | AND/ADD | $\phi$ | |
| 1 1 0 1 | ADD/ADDQ | OP-mode | 0 X X |
| | ADD/SUB | $\phi$ | |
| | ADD/AND | $\phi$ | |
| | ADD/$\phi$ | $\phi$ | |
| 1 1 1 0 | $ASL_{RS}$/$B_{CC}$ | Condition | X X X |
| | $ASL_{RS}$/AND | OP-mode | 1 X X |
| | | EA-mode | X 0 0 |
| | $ASL_{RS}$/$\phi$ | $\phi$ | |

註) ST : static, EA : effective address, D : destination, DIS : displacement, B : byte, L : long, W : word, RS : register shift

의 代表命令語에 대해 53個의 試驗命令語雙을 선정하였다. 試驗命令語雙 중에서 24個는 試驗條件이 부가되어 있으므로 이들은 試驗處理를 위한 operation word의 構成時 별도로 고려해야 한다.

2) 所屬命令語에 대한 試驗명령어雙

代表命令語 상호간의 機能試驗을 통하여 代表code의 정상적인 디코오딩이 확인되면 各 代表code가 소속되어 있는 部分命令語 集合의 所屬命令語들의 代表code

에 대한 디코오딩도 보장할 수 있다. 따라서 동일한 部分命令語 集合 $I_s(0 \leq S \leq 15)$에 속하는 所屬命令語 상호간의 機能試驗만 필요하게 되며 이를 위해 所屬 code를 바탕으로 所屬命令語 상호간의 試驗명령어雙을 選定한다.

같은 部分命令語 集合에 속하는 所屬命令語들은 상호간에 구별되는 특징적인 bit값을 갖고 있으므로 이들 各各을 試驗bit로 정하고 命令語 $i_j$의 試驗bit에 대한 點檢命令語 $i_k$를 결정한다.

定義 7 : operation word의 所屬code에는 各 所屬命令語들을 구별할 수 있는 bit값이 있으며 이 값을 그 命令語의 特性値, 그 해당 bit들을 特性bit $\xi$라고 한다.

定義 8 : 만일 命令語 $i_j$의 特性bit $\xi_k$에 대해 $\{\xi_j\} \subset \{\xi_k\}$이면 $i_j$의 $\bar{\xi}_j$에는 $i_k$의 해당 特性値의 입력이 금지되며 이 bit들을 $i_j$의 $i_k$에 대한 擬似特性bit $\xi_j'$라고 한다.

所屬命令語에 대한 試驗명령어雙의 選定은 命令語 $i_j$의 特性bit $\xi_j$ 모두에 대해 개별적으로 수행된다. 그러므로 이들 bit는 한번씩 試驗bit로 정해지며 各 試驗bit마다 1-bit差인 點檢命令語가 1個씩 選定된다. 그리고 해당 試驗bit에 대한 1-bit差 命令語가 없으면 $f(i_j/\phi)$가 되며 만일 命令語 $i_j$에 擬似特性bit $\xi_j'$가 존재하면 역시 試驗bit로 한다. 따라서 P個의 特性bit와 8個의 擬似特性bit를 가진 所屬命令語는 P+8個의 試驗명령어雙이 選定된다.

各 部分命令語 集合의 所屬命令語에 대한 選定알고리즘은 다음과 같다.

Algorithm 2

//choose test instruction pairs for partial instruction set $I$ $(0 \leq S \leq 15)$//

```
procedure Test Instruction Pairs of Party Instruction
  declare MAX  →number of element of Is(0 ≤≤15)
          ξ    →characteristic bit of I
          ξ'   →ps edo characteristic bit of I
          α    →bit number of LSB of ξ
          β    →bit number of MSB of ξ
          α'   →bit number of LSB of ξ'
          β'   →bit number of MSB of ξ'
          *    →operator of star product
          QINST→result of star product
          TB   →test bit
          TC   →test condition
1  begin
2    for all partial instruction set Is do
3      for j=0 to MAX do
4        take ij : //test instruction//
5        determine ξj of ij :
6        for m=α to β do
7          for k=0 to MAX do
8            take ik : //detect instruction//
9            if ij=ik then go to 18 :
10           if ξj ∉ ξk then go to 16 :
11           determine ξj' of ij :
12           for n=α' to β' do
13             print ij/ik : //test instruction pair for
               characteristic bit//
14             print TC : //test condition//
15           end :
16           QINST ij * ik :
17           if only bit m of QINST is φ then go to 20 :
18         end :
19         print ij/φ : go to 22 :
20         print ij/ik//test instruction pair for pseudo
           characteristic bit//
21         print TC : //test condition//
22       end :
23     end :
24   end :
25 end :
```

알고리즘 2의 수행결과 表 3에 보인 바와 같이 69個의 所屬命令語에 대해 332個의 試驗명령어雙을 구하였다. 部分命令語 集合중 $I_{10}$, $I_{15}$에는 命令語가 할당되어 있지 않으며 $I_1$, $I_2$, $I_3$, $I_7$은 원소가 1個뿐이므로 所屬명령어 상호간의 디스코오딩故障은 시험할 필요가 없다. 選定된 試驗명령어雙중 部分命令語 集合 $I_0$의 所屬命令語들에 대한 試驗명령어雙을 表 4에 예시하였다.

## Ⅳ. 試驗명령어雙에 대한 試驗

選定된 試驗명령어雙 $i_j/i_k$에 대한 命令語디코오딩 機能試驗은 $f(i_j/\phi)$와 $f(i_j/i_k)$를 모두 檢出할 수 있도록 수행되며 operation word의 構成, 試驗벡터의 결정및 試驗處理의 過程을 거친다.

operation word의 構成時 MC68000 $\mu$P의 effective address mode범주를 고려하여 共通의 mode을 선택하여 命令語디코오딩 機能故障은 addressing mode 內의 故障과 무관하다고 가정한다.

試驗알고리즘은 다음과 같다.

Algorithm 3

//processing of test for test instruction pairs//

```
Procedure Test Processing
  declare TC   →test condition
          OPW  →operat'on word
          CM   →common mode of test instruction pair ij/ik
          S(i) →source of instruction i
          D(i) →destination of instruction i
```

```
1  begin
2    for all test instruction pair do
3      take i_j/i_k : //test instruction pair//
4      load OPW with TC of i_j/i_k :
5      choose CM : //determine value of addressing
         mode field//
6      building OPW of i_j/i_k : //assembler of i_j/i_k//
7      load S(i_j) with test vector X :
8      load S(i_k) with test vector Y such that i_j(X) ≠ i_k(Y) :
9      read S(i_j) : //expected value X//
10     read S(i_k) : //expected value Y//
11     load D(i_j) with test vector W such that W ≠ i_j(X) :
12     read D(i_j) : //expected value W//
13     execute i_j :
14     read D(i_j) : //expected value i_j(X)//
15   end :
16 end :
```

例 1 : 試驗명령어雙 $BTST_{static}$/NEGX에 대한 試驗 다음과 같이 알고리즘 3 의 各 단계를 적용해본다.

단계 1 – 3 試驗명령어雙 $BTST_{static}$/NEGX를 취한다.

단계 4 operation word에 試驗條件을 대입한다.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $BTST_{static}$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | × | × | × | × | × | × |
| NBCD | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | × | × | × | × | × | × |

表 3. 各 部分命令語 集合에 대한 試驗명령어雙의 数
Table 3. Number of test instruction pairs for each partial instruction set.

| set number \ number | basic instruction | test instruction pair |
|---|---|---|
| $I_0$ | 15 | 58 |
| $I_1$ | 1 | ϕ |
| $I_2$ | 1 | ϕ |
| $I_3$ | 1 | ϕ |
| $I_4$ | 29 | 175 |
| $I_5$ | 4 | 16 |
| $I_6$ | 1 | ϕ |
| $I_7$ | 1 | ϕ |
| $I_8$ | 4 | 15 |
| $I_9$ | 2 | 6 |
| $I_{10}$ | ϕ | ϕ |
| $I_{11}$ | 3 | 11 |
| $I_{12}$ | 5 | 25 |
| $I_{13}$ | 2 | 6 |
| $I_{14}$ | 8 | 20 |
| $I_{15}$ | ϕ | ϕ |

단계 5 $BTST_{static}$의 effective address field는 data addressing mode이고 NBCD는 data alterable addressing mode이다. 따라서 이들 중 data register direct mode D1 을 선택한다.

단계 6 operation word를 구성하면 다음과같이 되므로 試驗對象인 試驗명령어雙의 assembler는 BTST #data, D1/NBCD D1가 된다.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $BTST_{static}$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| NBCD | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

단계 7 – 8 BTST #data, D1과 NBCD D1 의 source인 D1 및 data에는 試驗벡터로서 各各 $D1 \leftarrow \underbrace{AA\cdots A}_{8\,digit}$, data←01하고 condition code Zero인 CC(Z)에는 CC(Z)←0 한다.

단계 9 – 10 D1및 CC(Z)를 read한다. // 기대 값, D1 = AA…A 및 **CC**(Z) = 0 //

단계11 – 12 BTST의 Source와 destination이 같으므로 단계13으로 향한다.

단계13 命令語 BTST # S01, D1 을 실행한다.

단계14 data register D1및 condition code register의 CC(Z)를 read한다. //기대값, D1 = AA…A및 CC(Z) = 1//

알고리즘 3 을 총 332個의 試驗명령어雙에 대해 수행하였다. 表 5 는 이들 중 各 部分命令語 集合에서 한개씩을 선택하여 試驗명령어雙의 assembler, 試驗벡터 및 기대값을 예시한 것이다. 알고리즘 수행과정에서 RESET, AND등의 一部 試驗命令語는 여러개의 試驗명령어雙에 대한 試驗벡터가 동일하여 試驗의 複雜度가 더욱 감소됨을 알 수 있었다.

定理 2 : 알고리즘 3 은 命令語디코오더의 stuck-at 故障下에서 모든 命令語디코오딩 機能故障 $f(i_j/\phi)$ 와 $f(i_j/i_k)$ 을 檢出할 수 있다.

證明 : 만일 命令語디코오더에 stuck-at 故障이 존재한다면 알고리즘 3 의 단계 1 – 3 에서 취해진 試驗명령어雙의 集合속에는 定理 1 에 의해 반드시 그 故障効果가 포함되어 있다. 그리고 단계 4 – 6 에서 構成된 命令語 $i_j$ 및 $i_k$의 operation word는 試驗bit를 제외하고는 동일하므로 만일 그 해당 試驗bit에서 故障効果가 발생하여 命令語디코오딩 機能故障 $f(i_j/i_k)$가 존재하는 경우 $i_j$ 대신에 $i_k$가 실행될 것이므로 단계 7 – 8 에서 $S(i_j)$와 $S(i_k)$ 속에 $i_j(X) \neq i_j(Y)$되는 試驗벡터 X및 Y를 넣어두면 $i_j(X)$대신에 $i_k(Y)$가 $D(i_j)$속에 들어있을 것이다. 단계 9 – 10은 $S(i_j)$ 및 $S(i_k)$ 속에

표 4. $I_0$의 所屬命令語에 대한 試驗명령어雙
Table 4. Test instruction pairs for party instructions of $I_0$.

| test instruction pair ($i_J/i_K$) | test condition of field |
|---|---|
| $BCHG_{DY}/BCHG_{ST}$ | register : 1 0 0 |
| $BCHG_{DY}/BCET_{DY}$ | $\phi$ |
| $BCHG_{DY}/BTST_{DY}$ | $\phi$ |
| $BCHG_{ST}$/ORI | size : 0 1 |
| $BCHG_{ST}$/CMPI | size : 0 1 |
| $BCHG_{ST}$/EORI | size : 0 1 |
| $BCHG_{ST}$/MOVEP | EA-mode : 0 0 1 |
| | register : 0 0 0 |
| | OP-mode : 1 0 0 |
| $BCHG_{ST}/BSET_{ST}$ | $\phi$ |
| $BCHG_{ST}/BTST_{ST}$ | $\phi$ |
| $BTST_{DY}/BTST_{ST}$ | $\phi$ |
| $BTST_{DY}/BCLR_{DY}$ | $\phi$ |
| $BTST_{DY}/BCHG_{DY}$ | $\phi$ |
| $BTST_{ST}$/ORI | size : 0 0 |
| $BTST_{ST}$/CMPI | size : 0 0 |
| $BTST_{ST}$/EORI | size : 0 0 |
| $BTST_{ST}$/MOVEP | EA-mode : 0 0 1 |
| | register : 1 0 0 |
| | OP-mode : 1 0 1 |
| $BTST_{ST}/BCLR_{ST}$ | $\phi$ |
| $BTST_{ST/ST}/BCHG_{ST}$ | $\phi$ |
| $BSET_{DY}/BSET_{ST}$ | register : 1 0 0 |
| $BSET_{DY}/BCHG_{DY}$ | $\phi$ |
| $BSET_{DY}/BCLR_{DY}$ | $\phi$ |

| test instruction par($i_J/i_K$) | test condition of field |
|---|---|
| $BSET_{ST}/BSET_{DY}$ | register : 1 0 0 |
| $BSET_{ST}/BCHG_{ST}$ | $\phi$ |
| $BSET_{ST}/BCLR_{SS}$ | $\phi$ |
| $BSET_{ST}/\phi$ | $\phi$ |
| $BCLR_{DY}/BCLR_{ST}$ | register : 1 0 0 |
| $BCLR_{DY}/BTST_{DY}$ | $\phi$ |
| $BCLR_{DY}/BSET_{DY}$ | $\phi$ |
| $BCLR_{ST}$/ORI | size : 1 0 |
| $BCLR_{ST}$/CMPI | size : 1 0 |
| $BCLR_{ST}$/EORI | size : 1 0 |
| $BCLR_{ST}$/MOVEP | EA-mode : 0 0 1 |
| | OP-mode : 1 1 0 |
| $BCLR_{ST}/BTST_{ST}$ | $\phi$ |
| $BCLR_{ST}/BCHG_{ST}$ | $\phi$ |
| MOVEP/$BCHG_{DY}$ | OP-mode : 1 0 1 |
| | EA-mode : 0 0 1 |
| MOVEP/$BSET_{ST}$ | register : 1 0 0 |
| | EA-mode : 0 0 1 |
| MOVEP/$BTST_{ST}$ | registes : 1 0 0 |
| | EA-mode : 0 0 1 |
| ORI/$BCHG_{ST}$ | size : 0 1 |
| ORI/SUCI | $\phi$ |
| ORI/ANDI | $\phi$ |
| ORI/MOVEP | EA-mode : 0 0 1 |
| | register : 0 0 0 |

| test instructio pair($i_J/i_K$) | test condition of field |
|---|---|
| ANDI/EORI | $\phi$ |
| ANDI/ADDI | $\phi$ |
| ANDI/ORI | $\phi$ |
| ANDI/MOVEP | EA-mode : 0 0 1 |
| | register : 0 0 1 |
| SUBI/CMPI | $\phi$ |
| SUBI/ORI | $\phi$ |
| SUBI/ADDI | $\phi$ |
| AUBI/MOVEP | EA-mode : 0 0 1 |
| | register : 0 1 0 |
| ADDI/ANDI | $\phi$ |
| ADDI/SUBI | $\phi$ |
| ADDI/MOVEP | EA-mode : 0 0 1 |
| | register : 0 1 1 |
| ADDI/$\phi$ | $\phi$ |
| EDRI/ANDI | $\phi$ |
| EORI/$BCHG_{ST}$ | size : 0 1 |
| EORI/MOVEP | EA-mode : 0 0 1 |
| | register : 1 0 1 |
| EORI/$\phi$ | $\phi$ |
| CMPI/SUBI | $\phi$ |
| CMPI/$BCHG_{ST}$ | size : 0 1 |
| CMPI/MOVEP | EA-mode : 0 0 1 |
| CMPI/$\phi$ MPI | $\phi$ |

註) DY : dynamic, ST : static, EA : effective address

試驗벡터 X및 Y가 들어있는가를 확인한다. 단계11은 $f(i_J/\phi)$를 고려하여 $D(i_J)$를 $W \neq i_J(X)$인 試驗벡터 W로써 초기화시켰다. 따라서 이 故障이 존재하는 경우 $D(I_J)$속의 W가 변하지 않고 그대로 있을 것이다. 단계12는 $D(I_J)$속의 試驗벡터 Z를 확인한다. 단계13에서 命令語 $i_J$를 실행시키고 단계14에서 $D(i_J)$를 read하면 $f(i_J/\phi)$, $f(i_J/i_K)$가 존재하는 경우 기대값 $i_J(X)$와 다를 것이므로 故障이 檢出된다. Q. E. D

## V. 結 論

$\mu$P의 機能試驗을 複雜하게 하는 要因이 되는 命令語디코오딩 機能試驗을 効率的으로 遂行하기 위해 生産者측에서 제공된 情報중 operation word를 分析함으로써 命令語를 代表命令語와 所屬命令語로 분류하여 最少의 試驗명령어雙을 구하였다. 알고리즘 1 및 2는 이를 위한 것인데 MC68000 $\mu$P의 경우 基本命令語 数가 69個이므로 Thatte의 알고리즘을 적용하면 필요한 試驗명령어雙의 数가 4761個 以上되나 本 알고리즘으로는 332個이므로 試驗의 複雜度가 훨씬 감소되었다.

알고리즘 3은 선정된 試驗명령어雙에 대한 試驗을 위한 것이며 本 研究에 의하여 選定된 試驗명령어雙은 그 数가 적기 때문에 짧은 時間동안에 chip에 대한 命令語디코오딩 機能試驗을 할 수 있다. 따라서 데이터 처리부의 機能試驗을 위한 最適試驗命令語의 選定方法[10]과 함께 製作중의 chip에 대한 試驗은 물론 본 chip을 사용하는 SSM16등의 시스템의 試驗도 시스템의 試驗도 시스템의 여유시간(idle time)에 할 수 있다.

本 論文에서는 試驗명령어雙에 대한 試驗順序를 고

표 5. 試驗명령어雙에 대한 試驗處理 例
Table 5. Example of test processing for test instruction pairs.

| set number | test instruction pair ($i_j$ / $i_k$) | assembler | test vector | expected value |
|---|---|---|---|---|
| $I_0$ | BTST/NBCD | BTST#data, D1/NBCD D1 | D1←AAAAAAAA; CC(Z)←0;data←0001 | D1=AAAAAAAA;CC(Z)=1 |
| $I_1$ | MOVE. B/SUB | MOVE. B D0, D1/SUB. B D0, D1 | D0←AAAAAAAA; D1←FFFFFFFF | D1=FFFFFFAA |
| $I_2$ | MOVE. L/MOVE. W | MOVE. B D0, D1/MOVE. W D0, D1 | D0←AAAAAAAA; D1←FFFFFFFF | D1=AAAAAAAA |
| $I_3$ | MOVE. W/CMP | MOVE. W D0, D1/CMP. B D0, D1 | D0←AAAAAAAA; D1←FFFFFFFF | D1=FFFFAAAA |
| $I_4$ | CLR/NEGX | CLR. B D0/NEGX. B D0 | D0←55555555; CCCX>←0 | D0=55555500 |
| $I_5$ | $S_{cc}$/ADDQ | $S_{cc}$ D0/BDDQ. L#<data>, D0 | CC(C)←1 ;D0←33333333; data←#010 | D0=FFFFFFFF |
| $I_6$ | $B_{cc}$/$ASL_{reg}$ | BVS#S04/ASL. B#100, D4 | $\phi$ | PC=PC+04 |
| $I_7$ | MOVE Q/ADDQ | MOVEQ#S00, D1/ADDQ. B#001, D1 | D1←AAAAAAAA | D1=00000000 |
| $I_8$ | OR/DIVU | OR. L D0, D1/DIUU D0, D1 | D0←33333333; D1←66666666 | D1=77777777 |
| $I_9$ | SUB/SUBX | SUB. B D1, A0 @/SUBX. B D0, D1 | D0←33333333; D1←33333333<br>A0 @←66666666; CC(X)←0 | A0 @=66666633 |
| $I_{10}$ | $\phi$ | $\phi$ | $\phi$ | $\phi$ |
| $I_{11}$ | EOR/CMP | EOR. B D1, D0/CMP. B D0, D1 | D1←33333333; D0←55555555 | D0=55555566 |
| $I_{12}$ | ABCD/EXG | ABCD D0, D1/EXG D1, D0 | D0←33333333; D1←55555555 | D1=55555588 |
| $I_{13}$ | ADD/ADDX | ADD. B D1, A0 @/ADDX. B D0, D1 | D1←33333333; A0 @←55555555 | A0 @=55555588 |
| $I_{14}$ | ASR/LSR | ASR. L #<data>, D0/LSRL#<data>, D0 | D0←AAAAAAAA; data←#001 | D0=D5555555 |
| $I_{15}$ | $\phi$ | $\phi$ | $\phi$ | $\phi$ |

려하지 않았으나 fault coverage 및 test ambiguity를 개선하기 위해 기존의 알고리즘[4,5,6,7]과 조합하는 방안에 대해서도 硏究가 필요하리라 생각된다.

## 參 考 文 獻


[1] S.M. Thatte and J.A. Abraham, "A Methodology for Functional Level Testing of Microprocessors," *Proc. 8th Int. Conf. Fault-Tolerant Computing.* Toulouse, France IEEE Comput. Soc., pp. 90-95, June 1978.

[2] Y. Min and S.Y.H. Su, "Testing Functional Faults in VLSI," *Proc. 19th Design Automation Conf,* Las Vegas, Nevada, pp. 384-392, 1982.

[3] R. Chantal and S. Gabriele, "Microprocessor Functional Testing," *Test Conference,* IEEE Comput. Soc., pp. 433-443, 1980.

[4] M.A. Annaratone and M.G. Sami, "An Approach to Functional Testing of Microprocessors," *Proc. 1982 Fault-Tolerant Computing Symp* Santa Monica,' Ca, pp. 158-164, June 1982.

[5] S.M. Thatte and J.A. Abraham, "Test Generation for Microprocessors," *IEEE Trans. Comput.,* vol C-29, pp. 429-441, June 1980.

[6] K.K. Saluja and L. Shen, S.Y.H. Su, "A Simplified Algorithm for Testing Microprocessors," *1983 Int. Test Conference,* pp. 668-675, 1983.

[7] D. Brahme and J.A. Abraham, "Functional Testing of Microprocessors," *IEEE Trans. Comput.,* vol C-33, pp. 475-485, June, 1984.

[8] *MC68000 16-Bit Microprocessor User's Manual,* Motovola Inc., 1980.

[9] D. Lewin, *Computer-Aided Design of Digital System,* Crame, Russak & Company, Inc., pp. 136-138, 1977.

[10] 安光善, 마이크로프로세서 데이터 처리시험을 위한 最適試驗命令語, 대한전자공학회, vol.21, no. 1, pp. 57-61, 1984.

[11] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms,* Computer Science Press Inc., pp. 4-24, 1978.*