

マイクロプログラミング言語 MPL200 と

その最適化について

重松保弘、有川薫、安在弘幸

(九州工業大学 情報工学科)

1. 序 論

MPL200 (Microprogramming Language for U-200L) は、ユーザマイクロプログラミング可能な計算機 FACOM U-200L のために開発されたハイレベルマイクロプログラミング言語である。同計算機は垂直方式であるが、asynchronous, poliphase, parallel 方式を採用しているため、ユーザマイクロプログラミングを極めて困難にしている。

これまでに開発された、垂直方式向きマシンオリエンテッドなハイレベルマイクロプログラミング言語には、MLP-900 のための GPM や Interdata 32/S のための MPL¹⁾ (Microdata Programming Language) などがある。いずれも、コンパイラは比較的簡単で、オブジェクトのマイクロルーチンも効率が良いと報告されている。しかし、これらのコンパイラでは、マイクロ命令の並列実行については考慮されていない。MPL では、最適化についても今後の課題とされている。これに対して、MPL200 は、マイクロ命令の並列実行に対する考慮を払っており、(1) プログラムは、プログラミング時にマイクロ命令の先取りや、並列実行によるハードウェアリソースの競合を考えないでよい、(2) 並列実行に対する最適化をコンパイラで処置しているので、オブジェクトのマイクロプログラムは、ハンドコーディングに近い効率が得られる、などの特徴をもっている。

Patterson は、マイクロプログラムの開発・検証システム Strum²⁾ に、自動最適化は取り入れていないものの、構造化プログラミングを取り入れ、マイクロプログラムの開発に効果をあげている。MPL200 においても、マイクロプログラムの作成とデバッグの容易さを考慮して、CASE 文、LOOP 文、REPEAT 文、WHILE 文等を取り入れ "構造化マイクロプログラミング" を可能にしている。そのため、効率の良さとあわせて、整ったドキュメントを得ることができる。

アプリケーション向きのマイクロプログラミングテクニックにおいては、ハイレベル言語より、むしろマイクロルーチンの自動チューニングの必要性が強調されている³⁾。しかし、マイクロプログラムの開発においては、依然として、ハイレベル言語が重要な道具であることは否めない。それは、プログラムの開発時間に占めるコーディング時間の比率が、機械語レベルでは約 20% にすぎないの⁴⁾に比べ、マイクロプログラムレベルでは約 50% にも達するとの報告⁵⁾からも裏付けられる。最近では、とくに実効的な最適化技法を取り入れた、ユーザマイクロプログラミング用ハイレベル言語の必要性が強調されており⁶⁾ MPL200 は、こうした要求に合う特徴をもち、ユーザマイクロプログラミングの有効な道具になると思われる。

2. 垂直方式マイクロプログラミングの特徴

垂直方式によるマイクロ命令は通常、(1) 命令語長が短い、(2) 命令のオペレーション部がエンコードされている、(3) 1 マイクロ命令で 1 種類のオペレーションしか

参照しない、などの特徴をもっている。

(1) の特徴は、制御記憶の容量が少なくすむという利点をもつが、同時に、(a) アドレスがページ化される、(b) フルワードのデータに対する直接参照命令がない、などの不便な点が出てくる。(a) に対する処置は、一般にハードウェアサポートがないため、プログラマが注意しなければならない。(b) については、定数を格納するためのメモリを付加する場合がある。

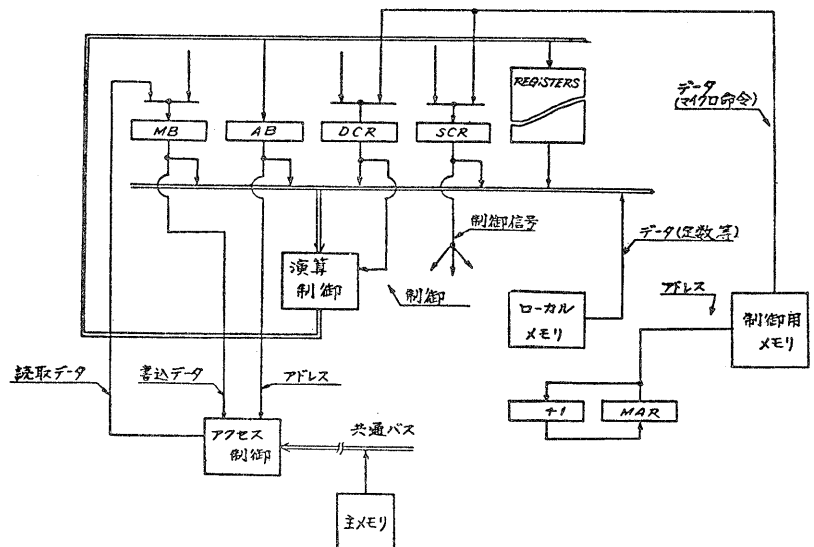
また、(2)、(3) の特徴から、複数のマイクロ命令を並列に実行することはあまり行わず、効率の低下を招くことが多い。このため、マイクロルーチンの実行速度を上げるための特殊なハードウェアを付加したり、主記憶からの入出力時間を利用したマイクロプログラミングで効率を上げる必要がある。

asynchronous, poliphase, parallel 方式では、マイクロ命令の先取り、および、複数のマイクロ命令の並列実行が起こる。そのため、(1) B (Branch) 命令または BC (Branch on Condition) 命令の使用時にプログラマは注意する必要がある、(2) ハードウェアリソースの競合が発生する場合がある、などの問題を生じる。これは、ユーザがマイクロプログラムをアセンブリ言語で作成するうえの困難な点である。一般に、タイミングコントロールが複雑であれば、ダミー命令の挿入によって、マイクロプログラミングを楽にできる。しかし、ダミー命令の挿入は制御記憶のロスとマイクロルーチンのスピードの低下をもたらす。このため、ダミー命令の挿入は必要最小限に限るべきである。Ramamoorthy らも、こうした問題について統括的な指摘を行なっている。⁵⁾

3. U-200L ハードウェアの特徴

U-200L のマイクロプログラムプロセッサ (マイクロプロセッサと呼ぶ) 制御部を図1に示す。U-200L のマイクロ命令は SCI (Sequence Control Instruction) と DCI (Data Path Control Instruction) の2種類に分かれている (詳細は付録参照)。

マイクロ命令は制御用メモリに格納され、MAR (Micro instruction Address Register) で示されるアドレスから、順次、読み出される。マイクロ命令は、すべて1語 (16 bit) から成り、制御記憶は4K語である。マイクロインストラクションレジスタは2個あり、SCIはSDRに、DCIはDCRに、それぞれ読み出される。読み出しは、2クロック (70 ns /



(図1) U-200L マイクロプロセッサ制御部

クロック) 単位に行われる。マイクロ命令の実行時間はSCIが2クロック、DCIでは大半が4クロックである。

U-200Lでは、DCIとSCIは独立に実行可能である。このため、4クロックのDCIとSCIが続いて読み出されたとき、この2つのマイクロ命令は並列に実行される。この例を図2に、そのタイミングチャートを図3に示す。そのため、

ハードウェアリソースの競合が発生するときがある。また、B命令、BC命令では、すぐ次のマイクロ命令を必ず実行するので、通常はNOP(No Operation)命令を置く必要がある。しかし、NOP命令の挿入は、マイクロルーチンを冗長にし、時間的な効率を低下させてしまう。また、NOP命令を有効なマイクロ命令で置きかえるためには、データフローの解析を必要とし、かえってプログラムミスの原因となりやすい。そのため、自動的にNOP命令の削除を行うソフトウェアサポートが必要となる。

主記憶とレジスタ間のデータの転送を行うためにAREQ(Access Request)命令が用意されている。しかし、通常、主記憶のアクセスには10クロック以上を必要とする。そこで、U-200Lでは、WAIT命令が、アクセス完了の同期をとるために用いられる。

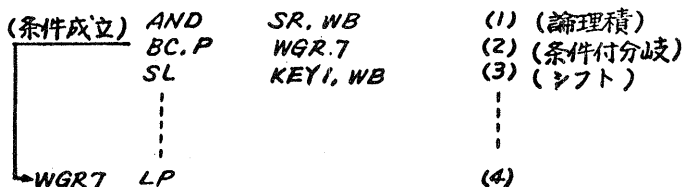
U-200Lのマイクロ命令には、直接数値を扱える命令がない。そのため、定数をあらかじめ格納できるメモリが用意されている。定数メモリ(ローカルメモリと呼ばれる)のLP(Load Pattern)領域が、このために用いられる。定数メモリには、LP領域の他にも、EX, SNI, SSAと呼ばれる領域があり、各々64語から成っている。

マイクロ命令のオペランドに指定できるレジスタ類を付録に示す。

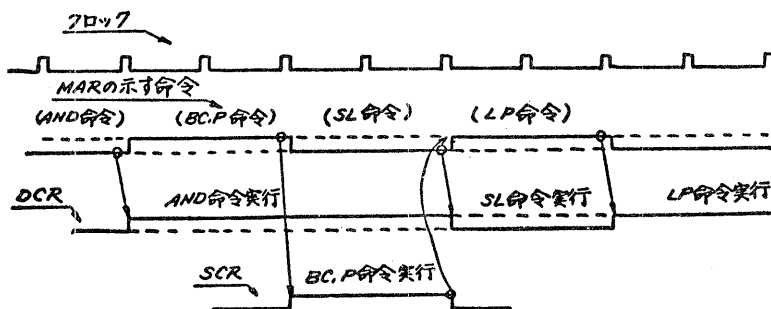
4. MPL200の特徴

MPL200は、PL/1 likeな形式をとっている(図4~図6参照)。それは、エミュレータや特殊目的向き計算機などを実現するためのシステム記述言語を目標としているためである。その結果、記述の自由度の高い言語の実現が可能になった。

U-200Lのマイクロ命令では、LP領域から値の読み出しはできるが、制御記憶から値の読み出しはできない。また、LP領域と制御記憶にマイクロ命令で、値を書き込むことはできるが、そのための時間が長い(6クロック)うえ、他のマイクロ命令と並列実行できない。こうした理由から、MPL200で使用できる変数



(図2) 命令の先取りと並列実行の例



(図3) 図2のマイクロルーチンのタイミングチャート

はレジスタに限られている。定数としては1語以内の2進数、10進数、16進数のいずれも指定可能である。

プログラムは、外部手続きと内部手続きから成り、外部手続きはCALL文によって内部手続きを呼び出す。内部手続きが、他の内部手続きを呼び出すことは禁止されている。その理由は、内部手続きの呼び出しのネスタイングにより、戻り番地を設定するためのレジスタの確保が必要になり、そのため、自由に使用できるレジスタが限定されてしまうからである。

手続きは、宣言文、注釈文、実行文および特殊な文から成る。宣言文は、変数名をレジスタに割りつけるために用いられる。外部手続きでの宣言は内部手続き内でも有効であるが、内部手続きでの宣言は、内部手続き内でのみ有効である。

実行文は(1)代入文、(2)制御文、(3)入出力文から成る。制御文には、GO TO文、IF文、DO文、CASE文、REPEAT文、WHILE文、LOOP文がある。図5、図6は、それぞれREPEAT文、CASE文の例である。入出力文にはREAD文、WRITE文、WCS文があり、主記憶に対する入出力および、制御記憶に対する出力が簡単に記述できる。図4はREAD文の例である。

特殊な文には、ORIGIN文、ASSEMBLER文、SET文がある。ORIGIN文は、マイクロルーチンの配置アドレスを指定する。

ASSEMBLER文は、アセンブリ言語形式で記述されたマイクロ命令のセ

ットを、MPL 200コンパイラのオブジェクトプログラム中に埋め込むために使用される。SET文は、CC(Condition Code)をSTR(Status Register)に格納するとき等に使用される。

MPL 200の特徴は、次のいくつかの点にまとめられる。

```

FETCH : PROC MAIN ;
/* INSTRUCTION FETCH */
DCL IC EQU GR7,
    ADDRESS EQU AB,
    WORK0 EQU FW0,
    WORK1 EQU FW1,
    WORK3 EQU FW3,
    MASK EQU ER,
    DISPLA EQU FL2 ;

ADDRESS=IC ;
READ INST ; /* INSTRUCTION
WORK3=0 ;
IC=IC+2 ;
ADDRESS=IC ;
MASK=X'FFF0' ;
WORK0=MASK ;
WORK1=X'E000' ;
SET OPB ;
READ INST ; /* OPERAND OR
DISPLA=DISP ;
SET SA EX ; /* DECODE AND
END FETCH ;

```

(図4) 命令フェッチルーチン

```

IPL: PROC MAIN;
DCL ENTKEY EQU GR1,
    PTR_AD EQU GR3,
    PARAM1 EQU GR2,
    PARAM2 EQU FRO,
    COMAND EQU MB,
    STATUS EQU MB,
    KEEP EQU FWO ;

IF ENTKEY <= 0
DO;
    PTR_AD = PTR_AD AND X'7FFF';
    PARAM1 = 0;
    PARAM2 = 2;
    CALL RD_DSR;
    COMAND = X'8800'; /* PTR START */
    CALL W-CMR;
    REPEAT DO;
        PARAM2 = 0;
        CALL RD_DSR;
        STATUS = STATUS SL 2;
        IF STATUS < 0
        THEN DO;
            KEEP = STATUS;
            GO TO RDATA;
        END;
        KEEP = STATUS;
        KEEP = KEEP AND X'2000';
    END;
    UNTIL KEEP > 0;
GO TO LAB ;

END;
FR1 = 0;
FR1 = 0;
RD_DSR : PROC ;
/* DUMMY */
FR1 = 0;
END RD_DSR ;
W-CMR : PROC ;
/* DUMMY */
FR1 = 0;
END W-CMR ;
RDATA : STATUS=0 ;
FR1 = 0;
END IPL;

```

(図5) IMPL ルーチン

- (1) PL/1 の派生語であり、マイクロプログラミングシステムの記述に適している。
- (2) 制御の流れに重点を置いた、構造化プログラミングが可能。
- (3) ローカルおよびグローバルな変数名の使用が可能。
- (4) アセンブリ言語形式で記述されたマイクロ命令の埋め込みが可能。

```

SELREG: PROC MAIN ;
DCL REGSEL EQU CT ; /* REGISTER SELECT */
      SOURCE EQU GRS ; /* GENERAL REGISTER SOURCE */
      INTCOD EQU IRC ; /* INTERRUPTION CODE REGISTER */
      OPEREG EQU OPR ; /* OPERATION REGISTER */
      ROTARY EQU KEY1 ; /* ROTARY SWITCH */
      PSW EQU STR ; /* STATUS REGISTER */
      REGSEL = ROTARY;
      IF REGSEL < 8
      THEN DO;
          CASE REGSEL OF
          0: OPEREG = AB;
          1: OPEREG = MB;
          2: OPEREG = PSW;
          3: OPEREG = INTCOD;
          4: OPEREG = FR0;
          5: OPEREG = FR1;
          6: OPEREG = FR2;
          7: OPEREG = FR3;
          END;
      ELSE DO;
          OPEREG = ROTARY;
          WB = SOURCE;
          OPEREG = WB;
      END;
      WB = X'0800';
END SELREG ;

```

(図6) レジスタセレクトルーチン

5. MPL200 コンパイラの最適化

一般に、マイクロプログラムの最適化は次の3種類がある。

- (1) 空間的最適化 : マイクロルーチンの占有する制御記憶の容量 (S) を減少させる。
- (2) 時間的最適化 : マイクロルーチンの実行に必要な時間 (T) を減少させる。
- (3) 空間的、時間的最適化 : $S \times T$ を最小にする。

この他にも、許容できる $S(T)$ のもとで $T(S)$ を最小にする方法などが考えられる。U-200L の制御記憶が 4K 語に制限されていることから、MPL200 コンパイラでは (1) に重点を置いた最適化を行なっている。

MPL200 コンパイラの最適化フェイズでは、次のような最適化を行なっている。

- (1) 冗長なマイクロ命令 (以下、命令と呼ぶ) の削除
 - (1-1) B 命令について
 - (1-2) C (Compare) 命令について
- (2) NOP 命令の削除
- (3) 主記憶に対する入出力タイミングの有効利用

(1) では、2つの場合の最適化を行なっている。(1-1) のタイプを図7に示す。このとき、冗長な B 命令と NOP 命令は削除できる。(1-2) のタイプを図8に示す。冗長な C 命令は削除される。(2) の場合、NOP 命令は通常、B、BC 命令に続いて

BC, P #0001	BC, P LAB	A FR1, MB	A FR1, ME
:		C C0000, MB	BC, P #0001
#0001 B LAB	⇒	BC, P #0001	
NOP			

(図8) B命令の削除

(図9) C命令の削除

生成されるので、次のような最適化が可能になる。

- (2-1) B, BC命令の直前の命令をNOP命令の位置に移す。
- (2-2) NOP命令を削除する。
- (2-3) B, BC命令の飛先の命令をNOP命令の位置にコピーし、飛先アドレスを+1する。
- (2-4) B, BC命令の飛先の命令をNOP命令の位置に移す。

このような最適化を行なうには、次の条件が必要になる。

- (2-1) B, BC命令およびNOP命令の位置に移した命令が、いずれも、他の命令の飛先にな、ていない。
- (2-2) NOP命令の次の命令が、B, BC命令の飛先以降で dead である。
ただし、命令が dead であるとは、その命令の sink (定義部のハードウェアリソース) の値、または carry flip flop の内容が、それ以降の制御の流れの中で使用されていない状態をいう。
- (2-3) BC命令のとき、分岐が成立しなかつた場合、NOP命令の位置に移された命令が dead である。
- (2-4) NOP命令の位置に移動する命令を飛先とするすべての命令について、(2-3) の条件が成立する。

ただし、いずれの場合も、NOP命令が他の命令の飛先にな、ていてはならない。

WAIT命令はAREQ命令と対で使用される。入力時は、WAIT命令の実行後、MB(Memory Buffer)の内容が入力データとして使用できる。出力時は、WAIT命令終了までMBの内容を保持しなければならない。この条件下で、入出力動作中、他のマイクロ命令を実行でき、効率の向上がはかれる。(3)の最適化では、WAIT命令の位置を移動させ、効率の向上をはか、ている。また、AREQ命令は、それに先行するAREQ命令によ、て入出力動作が開始されているとき、WAIT命令の機能を

```

FETCH      MV      GR7, AB
            (AREQ, INST)
            (WAIT)
            MV      C0000, FW3
            A        C0002, GR7
            MV      GR7, AB
            LP       000000, ER
            MV      ER, FW0
            LP       000001, FW1
            (SOPR)
            (AREQ, INST)
            (WAIT)
            MV      DISP, FL2
            (SSA, EX)
            NOP
            DC       LP, 11111111111110000
            DC       LP, 11100000000000000

```

(a) 最適化を行わないとき

```

FETCH      MV      GR7, AB
            (AREQ, INST)
            MV      C0000, FW3
            A        C0002, GR7
            MV      GR7, AB
            LP       000000, ER
            MV      ER, FW0
            LP       000001, FW1
            (WAIT)
            (SOPR)
            (AREQ, INST)
            (WAIT)
            (SSA, EX)
            MV      DISP, FL2
            LP, 111111111111110000
            DC       LP, 11100000000000000

```

(b) 最適化を行なったとき

(図10) 図4のプログラムのオブジェクトプログラム

U-200LのB命令のアドレスフィールドは12 bitであるが、BC命令のそれは8 bitである。このため、BC命令は256語単位のページ境界を越えて分岐することができない。そこで、コンパイラは、この制限にかかるBC命令を発見すると、同ページ内への分岐命令に変換し、新たに、目的の分岐先へのB命令を生成する。

(Step 数)	no optimization	automatic optimization	manual optimization
図4	15	14	12
図5	43	37	36
図6	63	38	35

(表1) 最適化の効果

8. 今後の課題

MPL200を用いることの利点は、まず、ドキュメントとしての見易さと理解の容易さである。それは、ひいてはプログラミング時間とデバッグ時間の短縮をもたらす。また、最適化によって、効率のよいオブジェクトプログラムを得ることができる。しかし、残された問題は多い。

マイクロプログラミングに対する1つの困難な問題は、デバッグである。MPL200は、各種の制御文を含め、制御の流れを構造的に記述できる。また、マイクロ命令の並列動作によるハードウェアリソースの競合や、入出力タイミングを考慮しなくてよいので、あらかじめ、マイクロプログラミング上のミスを防止できる。しかし、マイクロルーチンの論理的な誤りを見出すことはできない。そのためには、マイクロプロセッサのシミュレータを使用する方法が考えられる。しかし、シミュレータで、複雑なハードウェアタイミングをシミュレートすることは困難な場合が多い。こうしたことから、マイクロプログラム検証システムの作成が必要となってくると考えられる。

また、MPL200は、一般的な最適化の技法を取り入れていないので、この点についても検討の余地がある。また、限られた制御記憶容量の範囲内での、時間的最適化についても検討する必要がある。

さらに、マイクロプログラムの実行時に、動的に他のマイクロルーチンを制御記憶にロードして実行させる、いわゆるダイナミックマイクロプログラミングシステムの導入も必要になると思われる。⁶⁾

9. 謝辞

MPL200およびコンパイラの作成に協力していただいた古川正徳(九州工大、情報工学科)、江上照明(現在、日本電気ソフトウェア)の両氏に感謝します。また、日頃、御指導を仰ぐ九州大学吉田将教授に謝意を表します。

参考文献

- 1) A high-level microprogramming language [MPL], R.H. Eckhouse Jr., AFIPS Conference Proc. SJCC 1977
- 2) Strum: Structured Microprogram Development System for Correct Firmware, David A Patterson, IEEE Trans. on computers, vol. C-25, No. 10, Oct. 1976

- 3) Heuristic Synthesis of Microprogrammed Computer Architecture, A. M. Abd-Alla and David C. Karlgaard, IEEE Trans. on computers, vol. C-23, No. 8, Aug. 1974
- 4) Microprogram Optimization: A Survey, Tilak Agerwala, IEEE Trans. on computers, vol. C-25, No. 10, Oct. 1976
- 5) Optimization Strategies for Microprograms, R. L. Klein and C. V. Ramamoorthy, IEEE Trans. on computers, vol. C-20, No. 7, July 1971
- 6) オーバーレイ方式によるタイミツマイクロプログラムミツシステム, 重松, 安在, 九サ工大工学集報, 33号, 1976
- 7) Foundations of Microprogramming, Ashok K. Agrawala and Tomlinson G. Rauscher, Academic Press, Inc. New York San Francisco London 1976
- 8) The Identification of Maximal Parallelism in Straight-Line Microprograms, Subrata Dasgupta and John Jartar, IEEE Trans. on computers, vol. C-25, No. 10, Oct. 1976
- 9) FACOM U-200L マイクロプログラムプロセッサ動作取扱説明書, 富士通株式会社

(付録)

分類	命令コード	命令名称	略称	実行時間	命令 タイプ
転送命令	10001	MOVE	MT	2T	DCi
	10101	LOAD PATTERN	LP	4T	DCi
算術演算 命令	11000	ADD	A	4T	DCi
	11001	ADD AND MOVE TO AB	AMA	4T	DCi
	11010	ADD CARRY	AC	4T	DCi
	11011	SUBTRACT	S	4T	DCi
	11111	COMPARE	C	4T	DCi
論理演算 命令	11100	OR	OR	4T	DCi
	11101	AND	AND	4T	DCi
	11110	EXCLUSIVE OR	EOR	4T	DCi
シフト命令	10010	SHIFT RIGHT	SR	2T	DCi
	10011	SHIFT LEFT	SL	2T	DCi
	10110	SHIFT RIGHT DOUBLE	SRD	4T	DCi
	10111	SHIFT LEFT DOUBLE	SLD	4T	DCi
特殊命令	10100	EXECUTE	EX	4T	DCi
分岐命令	0000X	BRANCH	B	2T	Sci
	0001X	BRANCH AND LINK	BAL	2T	Sci
	00110	BRANCH ON CONDITION	BC	2T	Sci
制御命令	00101	SELECT NEXT INSTRUCTION	SNi	2T	Sci
	00100	CONTROL		2T	Sci

(付1) マイクロ命令一覧表

アドレス	オペランド名称	略称	READ WRITE	ビット数
000000	GENERAL REGISTER 0	GR0	R/W	16
000001	GENERAL REGISTER 1	GR1		
000010	GENERAL REGISTER 2	GR2		
000011	GENERAL REGISTER 3	GR3		
000100	GENERAL REGISTER 4	GR4		
000101	GENERAL REGISTER 5	GR5		
000110	GENERAL REGISTER 6	GR6		
000111	GENERAL REGISTER 7	GR7		
001000	GENERAL REGISTER DEST	GRD	R/W	16
001001	GENERAL REGISTER DEST ODD	GRDO		
001010	GENERAL REGISTER SOURCE	GRS		
001011	SCAL REGISTER	SCR	W	
001100	ADDRESS BUFFER REG	AB	R/W	16
001101	WORK BUFFER REG	WB		
001110	OPERATION REGISTER	OPR		
001111	COUNTER	CT		4
010000	MEMORY BUFFER REG	MB		16
010001	SYSTEM STATUS REG	SR		13
010010	STATUS REGISTER	STR		12
010011	LRC REGISTER	LRC		7
010100	FLOATING LINKAGE REG	FL2	R/W	16
010101	EXPONENT REGISTER	ER		
010110	FLOATING REGISTER DEST	FRD		
010111	FLOATING REGISTER DEST. ODD	FRDO		
011000	FLOATING REGISTER 0	FR0		
011001	FLOATING REGISTER 1	FR1		
011010	FLOATING REGISTER 2	FR2		
011011	FLOATING REGISTER 3	FR3		
011100	FLOATING WORK REGISTER 0	FW0		
011101	FLOATING WORK REGISTER 1	FW1		
011110	FLOATING WORK REGISTER 2	FW2		
011111	FLOATING WORK REGISTER 3	FW3		
100000	DISPLACEMENT	DSP	R	16
100001	DECODER	DEC	R	16
101000	FIXED CONSTANT #0000 (0000)	C0000	R	16
101001	FIXED CONSTANT #0001	C0001		
101010	FIXED CONSTANT #0002	C0002		
101011	FIXED CONSTANT #0004	C0004		
101100	FIXED CONSTANT #FFFF	CFFFF		
101101	FIXED CONSTANT #0007	C0007		
101110	FIXED CONSTANT #00F0	C00F0		
101111	FIXED CONSTANT #3000	C3000		
110000	ENTRY SWITCH	ENT	R	16
110001	KEY 1	KEY1		11
110010	KEY 2	KEY2		3

(付2) オペランド一覧表