論　文

マイクロプログラミング言語 MPL 200 とその最適化技法*

重松保弘**　有川　薫**　安在弘幸**

Abstract

This paper describes the characteristics and the optimization techniques of a high level microprogramming language, MPL 200, designed and developed for FACOM U-200 L. U-200 L has two microinstruction registers and asynchronous poliphase parallel architecture, which enables parallel execution of microinstructions.

Making these features work effectively, it is necessary for microprogrammer to know about microparallelism, resource conflict, hardware timing and optimization technique. This makes user-microprogramming defficult and troublesome.

MPL 200 releases microprogrammers from these problems, and is characterized by (1) it's employing structured programming concerned with the control flow, (2) it's implementing several automatic optimization techniques by means of checking resource conflict and information flow.

1. まえがき

MPL 200 は，ユーザマイクロプログラマブルな計算機 FACOM U-200 L (以後，U-200 L と記す) 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．

一般に，垂直型のマイクロ命令 (μI) をもつ計算機ではマイクロプログラミング (μPing) は比較的容易である．しかし，U-200 L は，垂直型の μI をもちながら，非同期 (asynchronous)，多相 (poliphase)，並列 (parallel) 方式を採用し，2個のマイクロ命令レジスタ (μIR) をもっているので，μI の先取りや並列実行が可能になっており，このことが，μPing を極めて困難なものにしている．

垂直型の μI をもつ計算機に対して開発されたハイレベル μPing 言語には GPM や MPL[1] 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プログラム (μP) も効率が良いと報告されている[7]．しかし，これらのコンパイラでは，μI の先取りや並列実行に対する検討はなされていない．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，MPL 200 では，μI の先取りや並列実行に対する考慮を払っており，次のような特徴をもっている．

(1)　μPing 時に，μI の実行順序や，並列実行によるハードウェアリソース (以後，リソースと記す) の競合 (resource conflict) を考えなくてよい．(2)最適化をコンパイラで処置しているので，効率の良いオブジェクト μP が得られる．

μP の最適化に関する初期の研究は，Kleir と Ramamoorthy によってなされている[2]．しかし，そこで述べられた最適化技法の実効性については明らかにされていない．本論文では，(1)タイミングの同期用に挿入した無効 μI の削除，(2)冗長な μI の削除，(3)主記憶のアクセス時間の有効利用，に関する最適化を行い，オブジェクト μP のステップ数の比較を行っている．

Patterson は，μP の開発・検証システム Strum[5] に構造化プログラミングを取り入れ，効果をあげている***．MPL 200 においても，構造化プログラミングを取り入れており，効率の良さとともに，整ったド

---

* A Microprogramming Language MPL 200 and It's Optimization Techniques by Yasuhiro SHIGEMATSU, Kaoru ARIKAWA and Hiroyuki ANZAI (Department of Computer Science, Kyushu Institute of Technology).

** 九州工業大学情報工学科

*** Strum では自動最適化を取り入れていない．

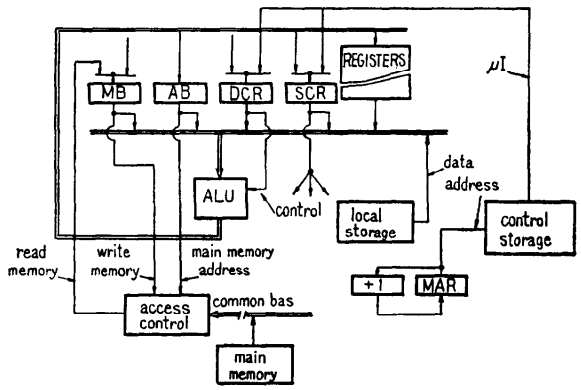


Fig. 1 Microprogramable processor of U-200 L.

キュメントリストを得ることができる.

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 μPing 用ハイレベル言語の必要性が強調されており[4]，MPL 200 は，ユーザ μPing のための有効な道具になると思われる.

2. U-200 L の特徴

ここでは，U-200 L のマイクロプログラマブルプロセッサの特徴について簡単に述べる. **Fig. 1** に，その制御部を示す.

μI は，SCI (Sequence Control Instruction) と DCI (Data path Control Instruction) の 2 種類に分かれている. μIR は，SCR と DCR の 2 個から成り，SCI は SCR に，DCI は DCR に，それぞれ読み出される. 読み出しは 2 T 単位 (70 ns/T) に行われる. μI の実行時間は，SCI は 2 T，DCI は大半が 4 T である. DCI と SCI は，並列に実行が可能である. このため，4 T の DCI に続いて SCI が読み出されたとき，この 2 つの μI は並列に実行される. **Fig. 2**, **Fig. 3** に，その例とタイミングチャートを示す.

```
        AND   SR, WB
        BC, P WGR 7
        SL    KEY 1, WB
        ⋮
→WGR 7  LP
```

Fig. 2 An example of microprogram for U-200 L.

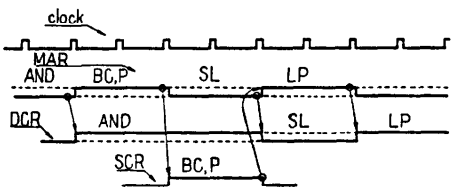


Fig. 3 Timing chart of Fig. 2.

U-200 L では，immediate タイプの μI がないため，定数をあらかじめ格納しておく記憶領域が用意されている. 定数メモリ (ローカルメモリ) の LP (Load Pattern) 領域がそれにあたる. 定数メモリには，このほか，機能分岐のための SSA (Set Start Address) 領域などがある.

3. MPL 200 の概要

MPL 200 は，エミュレータや特殊目的向き計算機を実現するためのシステム記述言語を目標としている.

プログラムは，1 つの外部手続きと，複数個の内部手続きから構成される (内部手続きのネスティングは許さない). 各手続きは，宣言文，注釈文，実行文および特殊な文から構成される. **Table 1**(次頁参照)に，言語仕様の概要を示す.

変数名には，μI のオペランドに指定可能なリソース名 (予約語となっている) が使用できる. また，宣言文によって，新たな変数名をリソースに割り付けることができ，以後，これをリソース名として使用することができる. このとき，外部手続きで宣言された変数名は，内部手続き内で新たに他のリソース名に割り付けられない限り，その内部手続き内でも，もとのリソース名として使用できる.

実行文には，代入文，制御文，入出力文および SET 文がある. 制御文には，CASE 文，REPEAT 文等があり，構造的な μPing を可能にしている. SET 文は，機能分岐等を指定するときに用いる. また，入出力文における READ 文，WRITE 文は，主記憶装置とレジスタ間のデータ転送を指定するために，WCS 文は，制御記憶にレジスタの内容を書き込むために用いる.

特殊な文における ASSEMBLER 文は，アセンブリ言語形式で記述された μP をオブジェクトプログラムに埋め込むときに，ORIGIN 文は，μP の配置アドレスを指定するときに，それぞれ用いる.

MPL 200 によるプログラム例を **Fig. 4** (次頁参照)，**Fig. 5** (752 頁参照) に示す.

Table 1 Language specification of MPL 200.

```
宣言文          DCL {〈変数名〉 / (〈変数名〉[,〈変数名〉]…)} EQU 〈リソース名〉;
注釈文          /* 〈注釈〉 */
実行文
 代入文         {〈変数名〉={〈算術式〉/〈論理式〉}
                 〈変数名〉=〈変数名〉{SR/SL}〈シフト回数〉
                 〈変数名〉〈変数名〉=〈変数名〉〈変数名〉{SRD/SLD}〈シフト回数〉};
 制御文
  GO TO 文      GO TO 〈ラベル〉;
  IF 文         IF 〈条件式〉 THEN {〈実行文〉 / DO; 〈実行文の列〉 END;} [ELSE {〈実行文〉 / DO; 〈実行文の列〉 END;}]
  DO 文         DO {〈変数名〉=〈初期値〉,〈最終値〉,〈増分〉; / 〈反復回数〉 TIMES;} 〈実行文の列〉 END;
  CASE 文       CASE 〈変数名〉 OF [〈定数〉:〈実行文の列〉]…END;
  REPEAT 文     REPEAT DO; 〈実行文の列〉 END; UNTIL 〈条件式〉
  WHILE 文      WHILE 〈条件式〉 DO; 〈実行文の列〉 END;
  LOOP 文       LOOP DO; 〈実行文の列〉 END; EXIT IF 〈条件式〉 DO; 〈実行文の列〉 END;
  CALL 文       CALL 〈手続き名〉;
 入出力文
  READ 文       READ [〈変数名〉] [FROM 〈主記憶アドレス〉] [INST];
  WRITE 文      WRITE [〈変数名〉] [INTO 〈主記憶アドレス〉] [INST];
  WCS 文        WCS [〈変数名〉] [INTO 〈制御記憶アドレス〉];
 SET 文         SET {CC / OPR / SA [EX]};
特殊な文
  ASSEMBLER 文  ASSEMBLER; 〈アセンブラ形式の μP の列〉 END;
  ORIGIN 文     ORIGIN 〈μP 配置アドレス〉;
```

定　数：1語 (16 bits) 以内の2進数，10進数，16進数．
算術式：2項演算のみを許す．算術演算子は，＋，－．
論理式：2項演算のみを許す．論理演算子は，AND, OR, EOR.
条件式：2項の比較演算およびスティタステスト．
　　比較演算子は，=，<，>，¬=，<=，>=．
　　スティタステストは，CARRY, COUNTER, その他．
反復回数：16未満の正の整数．
（宣言文は，Fig. 4, Fig. 5 に示すように ', ' で区切って続けてもよい）

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

        ADDRES=IC ;
        READ INST ;          /* INSTRUCTION FETCH */
        WORK3=0 ;
        IC=IC+2 ;
        ADDRES=IC ;
        MASK=X'FFF0' ;
        WORK0=MASK ;
        WORK1=X'E000' ;
        SET OPR ;
        READ INST ;          /* OPERAND OR NEXT INST FETCH */
        DISPLA=DISP ;
        SET SA EX ;          /* DECODE AND BRANCH TO MICRO ROUTINE */
END FETCH ;
```

Fig. 4 Example 1. Instruction fetch routine for U-200 (target machine).

4. μP の有向グラフ表現と Dead アルゴリズム

この章では，μI 相互の情報の流れ (information flow) に関する依存関係を調べる *Dead* アルゴリズムと，そのために用いる μP の有向グラフ表現について述べる．

μP の有向グラフ表現を $G(N, n_0, A)$ と表わす．ただし，N は節の集合，n_0 は G の入口節，$A \subseteq N \times N$ は矢の集合を表わす．各節は1つの μI を表わし，

```
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 */
A :REGSEL = ROTARY;
   IF REGSEL < 8
        THEN DO ;
             OPEREG = ROTARY ;
             WB = SOURCE ;
             OPEREG = WB ;
        END ;
        ELSE CASE REGSEL OF
                    8:  OPEREG = AB ;
                    9:  READ INST ;
                        OPEREG = MB ;
                   10:  OPEREG = PSW ;
                   11:  OPEREG = INTCOD ;
                   12:  OPEREG = FR0 ;
                   13:  OPEREG = FR1 ;
                   14:  OPEREG = FR2 ;
                   15:  OPEREG = FR3 ;
             END ;
   GO TO A ;
   END SELREG ;
```

Fig. 5 Example 2. Register select routine.

矢は μI 間の制御の流れ (μI がフェッチされる順序) を表わす.

節 n に対応する μI (節 n の μI とよぶ) のオペレーション部を $op(n)$ と表わす. U-200 L の μI のオペレーション部は, FMO (機能オペレーション), BMO (B, BAL, BC および SSA の各オペレーション)* および NOP (無効オペレーション) に分かれる.

節 n の μI によって, その値が定義または参照されるリソースの集合を, それぞれ $r_d(n)$ または $r_u(n)$ と表わす. また, 矢 (n_i, n_j) が存在するとき, n_i を n_j の直接先行節, n_j を n_i の直接後行節とよぶ. 節 n の直接後行節の集合を $IS(n)$ と表わす. 矢 (n_1, n_2), (n_2, n_3), …が存在するとき, 節の順序集合 $(n_1, n_2, \cdots)$ を経路といい, n_1 を, この経路の先頭節とよぶ. n を先頭節とする経路の集合を $P(n)$ と表わし, その1つの要素を $p(n)$ と表わす.

節 n_i が n_j に関して *dead* であるとは, n_i の μI によってその値が定義されるすべてのリソース ($r_d(n_i)$) の値が, n_j を先頭節とするすべての経路 ($\forall p(n_j) \in P(n_j)$) で使用されていない状態をいう. n_i が n_j に関して *dead* であるかどうかを判定するアルゴリズムを次に示す. ただし, R はリソースの集合, N と M は節の集合, n と m は任意の節を示す. また P_{d_s} はプッシュダウンスタックを示し, $P_{d_s}\downarrow(m, N, R)$ は, (m, N, R) をスタックすること, $P_{d_s}\uparrow(n, N, R)$ は, (n, N, R) に対応する項をポップアップすることを示す.

〔アルゴリズム *Dead*〕

(1) P_{d_s}, R, N を空 (ϕ) とする.

(2) R に $r_d(n_i)$ を入れる. n_j を n とする.

(3) N に n を入れる.

(4) $R \cap r_u(n) \neq \phi$ なら終る (*not dead*)

(5) $R - r_d(n)$ を R とする.

(6) $R \neq \phi$ なら (8) へ行く.

(7) $P_{d_s} = \phi$ なら終る (*dead*). そうでなければ, $P_{d_s}\uparrow(n, N, R)$ を実行し, (10) へ行く.

(8) $IS(n)$ を M とする. M から1つ要素を取り出し n とする**.

(9) $M \neq \phi$ が成立している間, (9a) をくりかえす.

(9a) M から1つ要素を取り出し m とする. $P_{d_s}\downarrow(m, N, R)$ を実行する.

(10) $n \in N$ が成立している間, (10a) をくりかえす.

(10a) $P_{d_s} = \phi$ なら終る (*dead*). そうでなければ $P_{d_s}\uparrow(n, N, R)$ を実行する.

(11) (3) へ行く.

今, 任意に1つのリソース $r \in r_d(n_i)$ と, 1つの経路 $p(n_j)$ をとってみる. *Dead* は, $p(n_j)$ の先頭節から遂次, 節 n を取り出し, 次のいずれかが成立するまで $r_u(n)$ と $r_d(n)$ を調べてゆく.

(i) $r \in r_u(n)$.

(ii) $r \in r_d(n)$ かつ $r \notin r_u(n)$.

(iii) すべての節 $n \in p(n_j)$ について, $r \notin r_u(n)$ かつ $r \notin r_d(n)$.

上記の (i) が成立すれば, n_i が定義する r の値は使用されている. しかし, (ii), (iii) のいずれかが成立すれば, この値は使用されていない. *Dead* は, $P(n_j)$ の1つの経路で (i) が成立するとき (4) で終了する. また, $P(n_j)$ のすべての経路について (ii) または (iii) が成立するとき, (7) または (10a) で終了する.

5. 最 適 化

この章では, MPL 200 で採用した最適化の手法と, それが適用可能な条件について述べる.

5.1 タイミングの同期用無効命令の削除

U-200 L は並列方式であるため, BMO の実行に,

* B: Branch, BAL: Branch And Link, BC: Branch on Condition SSA: Set Start Address の略. SSA は機能分岐を指定するオペレーション.

** μP は一般に停止しないので, 直接後行節は必ず存在する.

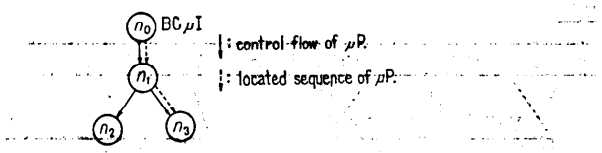


Fig. 6 An example of directed graph representation of μP.

次のような特徴をもっている．すなわち，BMO によって分岐が起こるとき，次の番地の μI が実行された後，分岐先の μI が実行される．これを，**Fig. 6** の例で示す．ここで実線は矢（μP の制御の流れ）を，破線は μI の並び（μI の配置される順序）を，それぞれ示す．また，n_0 は BC μI であり，そのオペランド（分岐先の番地）は，n_2 の μI を指している．このとき，n_0 の BC μI で分岐条件が成立すると，n_0, n_1, n_2 の順序で μI が実行される．

一般に，こうした μI の先取りは，μPing を極めて複雑にする．このようなとき，n_1 の μI として NOP μI を置くことにすれば，μPing は，はるかに容易となり，μP コンパイラを作成する労力も軽減することができる．

MPL 200 コンパイラでは，そのセマンティックフェイズにおいて，n_1 の μI として NOP μI を生成する．しかし，これはオブジェクト μP のステップ数や実行時間を増大させることになる．そこで，その後の最適化フェイズにおいて，この NOP μI を他の有効な μI で置きかえる最適化を行っている．また，そのために，（1）μP の情報の流れの解析，（2）リソースの競合のチェック，などを行っている．

以下に，その最適化技法と，それが適用可能な条件を述べる．なお，条件の記述においては，次のような記法を用いる．n_i と n_j の μI を，この順序でフェッチしてもリソースの競合が発生しないことを $con(n_i, n_j)$，n_i が n_j に関して *dead* であることを $dead(n_i, n_j)$ と，それぞれ表わす．U-200 L の BC μI の分岐条件のなかには，その BC μI が 4 T の DCI と並列に実行されなければ，正常に分岐の判断ができないものがある．そこで，n_j が BC μI のとき，n_i と n_j の μI を並列に実行しなくてもよいことを $sep(n_i, n_j)$ と表わす．

〈最適化 1〉 Fig. 7 に，この手法を示す．

（条件） n_4 が NOP μI のとき，（1） $op(n_1) \notin$ BMO，（2） $con(n_2, n_5)$．n_3 が BAL μI のときは，

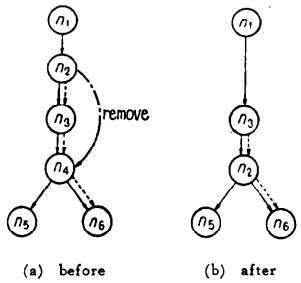


Fig. 7 Optimization technique 1.

さらに，（3） $con(n_1, n_3)$*．n_3 が BC μI のときは，さらに，（4） $sep(n_2, n_3)$，（5） $con(n_2, n_6)$．n_2 の直接先行節が複数個存在するときは，そのすべての節について，上記の条件を満足すること．

（手法） n_2 を n_4 の位置に移す．このとき（1） n_2 のすべての直接先行節の矢の向きを n_3 へ変更する．（2） n_4 のすべての直接先行節（n_3 を除く）の矢の向きを n_6 へ変更する．

〈最適化 2〉 Fig. 8 に，この手法を示す．

（条件） n_2 が NOP μI のとき，（1） n_1 は B，または BC μI，（2） $op(n_4) \notin$ BMO，（3） $con(n_4, n_3)$，（4） $dead(n_4, n_3)$．

（手法） n_4 を n_2 の位置に移す．このとき，n_2 のすべての直接先行節の矢の向きを n_4 へ変更する．

〈最適化 3〉 Fig. 9 に，この手法を示す．

（条件） n_2 が NOP μI のとき．n_1 が B または BAL μI のときは，（1） $op(n_3) \notin$ BMO．n_1 が BC μI のときは，さらに，（2） $con(n_3, n_5)$，（3） *dead*

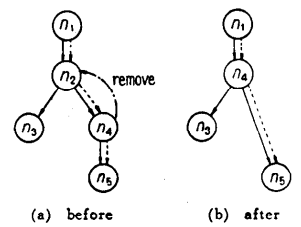


Fig. 8 Optimization technique 2.

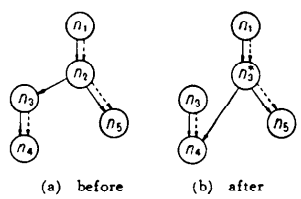


Fig. 9 Optimization technique 3. (n_3*: the copy of n_3).

* n_1 が 4 T の DCI のときリソースの競合が発生する．

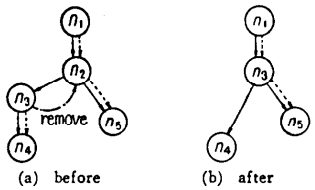


Fig. 10 Optimization technique 4.

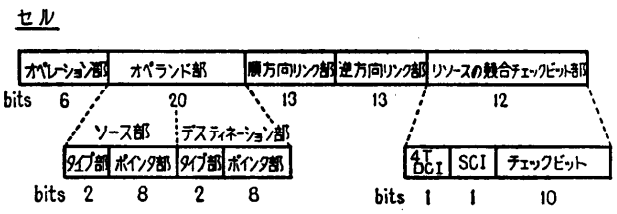


Fig. 13 Specification of fields of a cell.

(n_3, n_5).

(手法) n_3 のコピーを n_2 の位置に置く．n_1 の μI の飛先を n_4 に変更する．また，n_2 のすべての直接先行節の矢の向きを n_5 に変更する．

〈**最適化 4**〉 **Fig. 10** に，この手法を示す．

(条件) n_2 が NOP μI のとき．n_1 が B または BAL μI のときは，(1) $op(n_3) \in$ BMO，(2) n_3 の直接先行節が n_2 のみであること．n_1 が BC μI のときは，さらに (3) $con(n_3, n_5)$，(4) $dead(n_3, n_5)$.

(手法) n_3 を n_2 の位置に移し，n_1 の μI の飛先を n_4 に変更する．

5.2 冗長な μI の削除

冗長な μI の削除は，B μI と C(Compare) μI の場合について行っている．

B μI の削除の例を **Fig. 11** に示す．$op(n_1) \in$ BMO とする．n_3 が B μI であり，n_4 が NOP μI のとき，n_3 の直接先行節が n_2 のみであり，かつ，$con(n_2, n_5)$ なら n_3 と n_4 を削除する．また，n_1 の μI の飛先を n_5 に変更し，n_4 の直接先行節の矢の向きは n_6 へ変更する．

C μI の削除の例を **Fig. 12** に示す．(a)では，C μI と BC μI が並列に実行され，MB レジスタの内容が正のとき分岐条件が成立する．(b)では，A

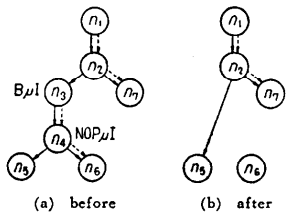


Fig. 11 Deletion of Branch and No-operation microinstructions.

```
A       FR1, MB          A       FR1, MB
C       C0000, MB        BC, P   #0001
BC, P   #0001
   ⋮                        ⋮
(a) before               (b) after
```

Fig. 12 Deletion of Compare microinstruction.

(Add) μI と BC μI が並列に実行され，加算の結果(すなわち，MB レジスタに格納される値) が正のとき，分岐条件が成立する．したがって，(a)と(b)とは同じ処理となる．そこで，(a)の冗長な C μI を削除する．

5.3 主記憶のアクセス時間の有効利用

主記憶の読み出し，または書き込みには AREQ (Access Request) μI が，また，その完了待ち合わせには WAIT μI が使用される．

読み出し時には，WAIT μI が完了すると MB レジスタの内容を読み出しデータとして使用できる．また，書き込み時には，WAIT μI が完了するまで MB レジスタの内容を保持しなければならない．この条件を満足すれば，主記憶の読み出し，または書き込み動作中に，他の μI を実行できる．そこで，コンパイラでは，AREQ μI と WAIT μI の間に，他の有効な μI を移動し，μP の実行時間を短縮している．

6. 最適化のためのデータ構造

ここでは，これまでに述べた最適化を行うためのデータ構造について述べる．

コンパイラは，ソース μP の中間形式を生成する．その単位をセルとよび，その構造を **Fig. 13** に示す．各セルは，オペレーション部とオペランド部をもつ．オペレーション部は，各セルのタイプ (ラベル，μI，その他) を規定する．オペランド部はソース部とデスティネーション部に分かれ，それぞれ，タイプ (リソース，リテラルまたはラベル) 部と，各タイプのテーブルへのポインタ部をもつ．また，各セルは，セル相互の並びを示すための順方向および逆方向のリンク部をもつ．さらに，各セルは，リソースの競合をチェックするためのチェックビット部をもっている．リソースの競合は，2つのセルのチェックビット部の論理演算を行うことによって検出される．

これまでに述べた有向グラフにおける各節は，1個 (μI タイプのセルのみからなる) ないし複数個 (ラベルタイプと μI タイプのセルの対など) のセルに対応

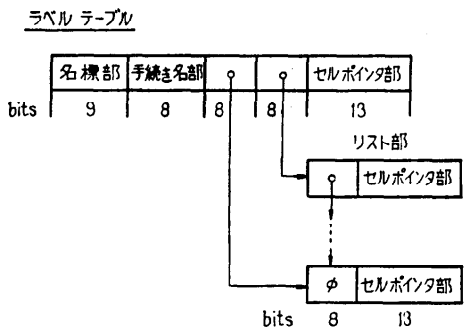
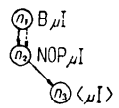
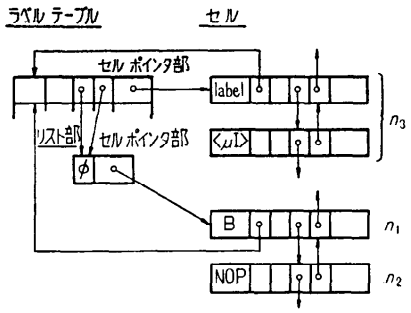


Fig. 14 Specification of fields of the label table.



(a) An example of directed graph (part).



(b) Data structure representation of (a).

Fig. 15 An example of directed graph and it's data structure representation.

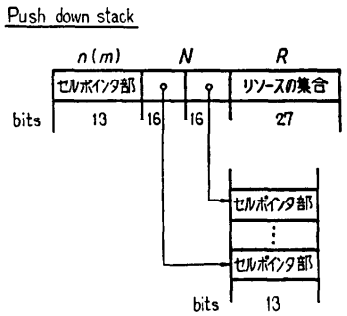


Fig. 16 Specification of fields of push-down stack.

している．したがって，各節に対する有向グラフ上の最適化のための操作は，その節に対応する1個ないし複数個のセルに対する操作を意味している．

また，ラベルを識別するためにラベルテーブルが設けられている（**Fig. 14**）．各ラベルには，そのラベルのセルを示すポインタ部と，そのラベルを飛先とする分岐 μI のセルへのポインタを保持するリスト部がある．**Fig. 15** に，有向グラフと，セルおよびラベルテーブルの対応関係について，簡単な例を示す．

Fig. 16 に，*Dead* アルゴリズムで用いる Push down stack の構造を示す．ここでチェックしている U-200 L のリソースは 27 個である．そのため，リソースの集合は 27 bits で表わし，各 bit 位置に 1 個のリソースを対応づけている．

7. 最適化の効果

Fig. 17(次頁参照) は，**Fig. 5** の μP のオブジェクト μP である．(a)は最適化を行っておらず，(b)では最適化を行っている．これは，CASE 文の例であるが，最適化を行わなければ，(a)に示すように極めて冗長なオブジェクト μP が生成される．しかし，最適化を行うことによって，すべての NOP μI が削除され，冗長な B μI が削除されていることがわかる．

Fig. 4 の μP のオブジェクト μP では，最適化を行っても1ステップしか減少していない．しかし，主記憶のアクセス時間を有効に利用することによって，実行時間が短縮されている[11]．

いくつかの例について最適化の効果をまとめると，**Table 2** に示すようになる．Table 2 の例 3 は，IMPL (Initial Micro Program Load) ルーチン[11]であるが，最適化によって，ハンドコーディングに近いオブジェクト μP が得られていることがわかる．

また，Table 2 に，各例について最適化を行うために要した CPU time を示す．

8. むすび

MPL 200 を用いることの利点は，ドキュメントの見易さと，μP の理解の容易さである．それは，ひいてはプログラム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 μP を得ることができる．しかし，残された問題も多い．

Table 2 Effect of optimization.

	Compiler object	automatic optimization	manual optimization	CPU time for automatic optimization
Ex. 1 (Fig. 4)	15 steps	14 steps	12 steps	0.64 sec
Ex. 2 (Fig. 5)	71	45	42	6.15
Ex. 3	46	37	36	2.49

```
SELREG  MV      KEY1,CT
        LP      000000,ER
        C       ER,CT
        BC,M    #0001
        NOP
        LP      000000,ER
        C       ER,CT
        BC,Z    #0003
        NOP
        LP      000001,ER
        C       ER,CT
        BC,Z    #0004
        NOP
        LP      000010,ER
        C       ER,CT
        BC,Z    #0005
        NOP
        LP      000011,ER
        C       ER,CT
        BC,Z    #0006
        NOP
        LP      000100,ER
        C       ER,CT
        BC,Z    #0007
        NOP
        LP      000101,ER
        C       ER,CT
        BC,Z    #0008
        NOP
        LP      000110,ER
        C       ER,CT
        BC,Z    #0009
        NOP
        LP      000111,ER
        C       ER,CT
        BC,Z    #000A
        NOP
#0003   MV      AB,OPR
        B       #000B
        NOP
#0004   (AREG,INST)
        (WAIT)
        MV      MB,OPR
        B       #000B
        NOP
#0005   MV      STR,OPR
        B       #0008
        NOP
#0006   MV      IRC,OPR
        B       #000B
        NOP
#0007   MV      FR0,OPR
        B       #000B
        NOP

#0008   MV      FR1,OPR
        B       #000B
        NOP
#0009   MV      FR2,OPR
        B       #0008
        NOP
#000A   MV      FR3,OPR
        B       #000B
        NOP
#000B   NOP
        B       SELREG
        NOP
#0001   MV      KEY1,OPR
        MV      GRS,WB
        MV      WB,OPR
        B       SELREG
        NOP
        DC      LP,0000000000001000
        DC      LP,0000000000001001
        DC      LP,0000000000001010
        DC      LP,0000000000001011
        DC      LP,0000000000001100
        DC      LP,0000000000001101
        DC      LP,0000000000001110
        DC      LP,0000000000001111
```

(a) without optimization

```
SELREG  MV      KEY1,CT
        LP      000000,ER
        C       ER,CT
        BC,M    #000D
        MV      KEY1,OPR
        LP      000000,ER
        C       ER,CT
        BC,Z    #0003
        MV      AB,OPR
        LP      000001,ER
        C       ER,CT
        BC,Z    #0004
        LP      000010,ER
        C       ER,CT
        BC,Z    SELREG
        MV      STR,OPR
        LP      000011,ER
        C       ER,CT
        BC,Z    SELREG
        MV      IRC,OPR
        LP      000100,ER
        C       ER,CT
        BC,Z    SELREG
        MV      FR0,OPR
        LP      000101,ER
        C       ER,CT
        BC,Z    SELREG
        MV      FR1,OPR
        LP      000110,ER
        C       ER,CT
        BC,Z    SELREG
        MV      FR2,OPR
        LP      000111,ER
        C       ER,CT
        BC,Z    SELREG
        MV      FR3,OPR
        B       SELREG
        MV      AB,OPR
#0003   (AREG,INST)
#0004   (WAIT)
        B       SELREG
        MV      MB,OPR
#000D   MV      GRS,WB
#0002   B       SELREG
        MV      WB,OPR
        DC      LP,0000000000001000
        DC      LP,0000000000001001
        DC      LP,0000000000001010
        DC      LP,0000000000001011
        DC      LP,0000000000001100
        DC      LP,0000000000001101
        DC      LP,0000000000001110
        DC      LP,0000000000001111
```

(b) after optimization

Fig. 17 Object program of Fig. 5.

1つの問題は，MPL 200 では μP の論理的な誤りを検出することができないことである．μP のデバッグには，一般にシミュレータが用いられる．しかし，複雑なハードウェアタイミングのシミュレーションは困難な場合が多い．そのため，μP の検証システムとしての機能も付加する必要があると思われる．

MPL 200 では，一般的な最適化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，限られた記憶容量内での実行時間の短縮についても検討する必要がある．さらに μP の実行時に，動的に他の μP 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 μPing システムの導入も必要になると思われる[10]．

MPL 200 コンパイラは，PL/1 言語を用い，約 7000 ステップで記述されている．現在，九州工業大学・情報工学科の FACOM 230-45 S の上で，クロスコンパイラとして動作しており，実行には約 73 K 語の主記憶容量を必要としている．

9. 謝 辞

資料を提供していただいた富士通株式会社，および，日頃，ご指導を戴く吉田 将教授（九州大学）に感謝します．

参 考 文 献

1) R. H. Eckhouse: A high level microprogramming language (MPL), Proc. SJCC (1971).
2) R. L. Kleir & C. V. Ramamoorthy: Optimization Strategies for Microprograms, IEEE Trans. on computers, Vol. C-20, No. 7 (July

1971).
3) A. M. Abd-Alla & D. C. Karlgard: Heuristic Synthesis of Microprogrammed Computer Architecture, IEEE Trans. on computers. Vol. C-23, No. 8 (Aug. 1974).
4) T. Agerwara: Microprogram Optimization: A Survey, IEEE Trans. on computers, Vol. C-25, No. 10 (Oct. 1976).
5) D. A. Patterson: STRUM: Structured Microprogram Development System for Correct Firmware, IEEE Trans. on computers, Vol. C-25, No. 10 (Oct. 1976).
6) S. Dasgupta & J. Tartar: The Identification of Maximal Parallelism in Straight-Line Microprograms, IEEE Trans. on computers, Vol. C-25, No. 10 (Oct. 1976).
7) A. K. Agrawala & T. G. Rauscher: Foundations of Microprogramming, p. 416, Academic Press Inc., New York (1976).
8) マイクロプログラミング特集号, 情報処理, Vol. 14, No. 6 (1973).
9) 萩原 宏: マイクロプログラミング, p. 343, 産業図書, 東京 (1977),
10) 重松, 安在: オーバーレイ方式によるダイナミックマイクロプログラミングシステム, 九州工大工学集報, 33 号 (1976).
11) 重松, 有川, 安在: マイクロプログラミング言語 MPL 200 とその最適化について, 情報処理学会計算機アーキテクチャ研究会資料 26 (1977).
12) FACOM U-200 L 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.

(昭和 52 年 8 月 1 日受付)
(昭和 53 年 1 月 27 日再受付)