マイクロコード・チェッカ「マイクロリント」の開発

宮内 信仁　　　市吉 伸行
（三菱電機）　　　（ＩＣＯＴ）

1．はじめに

我々は第五世代コンピュータ・プロジェクトの一貫として並列型推論計算機の開発を行っている。実装されている並列論理型言語ＫＬ１の処理系はマイクロプログラムによってコーディングされており，ルーチンの構成は数階層にもわたり，非常に複雑なものである。そこで，デバッグの効率を向上させるためにルーチン呼び出し時の汎用レジスタがマイクロコーダの指示通りに使用されているかどうかのチェックを自動的に行うツールを開発したので報告する。

2．マイクロプログラム開発の現状

2-1．マイクロリント開発の動機

一般にマイクロプログラムは複数のモジュールにより全体が構成される。各モジュールの処理においては，作業用領域として汎用なレジスタを頻繁に使用する一方で，モジュール間の呼び出しのインターフェースとしてレジスタ上で必要なデータの受け渡しをすることも多い。汎用なレジスタはＣＰＵ内では限られた資源であり，高度な処理系を実現するためにはレジスタを効率良く使用する必要がある。

我々が開発した並列型推論計算機Multi-PSI/V2におけるＫＬ１処理系に代表されるような複雑な処理を要求されるマイクロプログラムではコード量が膨大になるために冗長なコーディングでは16KwordのＷＣＳ（プロセッサエレメントは，PSI-IIのCPU）といえども容量は不足する。

このために同様な処理をできるだけ共用できるように，サブルーチンの切り分けを細かくし何階層にも分けて呼び出しを行なうことが複雑な処理を実現するために不可欠となる。したがって，これらのサブルーチン間では呼び出しの入出力や作業用領域に使用するレジスタを明示して，それぞれの階層で使用されるレジスタを破壊しないように注意深いコーディングが必要とされる。

しかし，多くのマイクロコーダが経験するように，サブルーチンの呼び出し前後でレジスタの内容が破壊されてしまうバグが数多くあるのが現状である。我々の経験ではこの種のバグが，初期デバッグ時のバグの３割程度を占めており，マイクロソースの修正過程で入り込むこともあり，デバッグ効率を悪化させるかなり大きな要因となる。

そこで，このようなバグは機械的にチェックできるので，このマイクロコードの各処理のパスでの引数レジスタ及び使用レジスタの整合性を解析してチェックするプログラムがマイクロリントである。

2-2．Multi-PSI/V2 のマイクロコーディング

PSI-II は水平型マイクロプログラミング方式による計算機であり，高度な論理型言語を高速に実行する機械語を実現するためにマイクロプログラム１ステップ中で操作できるハードウェア資源は多い。このために各資源に指示を与える各マイクロ命令フィールドの種類が多いので，読みやすくプログラミングが容易なコーディング形式が使用されている。

各マイクロステップは，フィールドごとのレジスタ操作，メモリ操作，分岐命令などのオペレーションが，コンマで区切られて列挙されており，ステップの終了にはピリオドが付けられる。このマイクロプログラムの１ステップにはラベルが付される場合がある。

１ステップの例としては，以下のようなものがある。

```
label_first:
   mar := r0d, read(mar, mdr),
   ----------  ---------------
        ↑             ↑    if (zero) goto jump_label.
 レジスタ演算操作     |    --------------------------
                      |                 ↑
              メモリのリード        条件分岐
```

mar:メモリアドレスレジスタ
mdr:メモリデータレジスタ

3．マイクロリントの機能

3-1．リントが行うチェックの内容

マイクロコード上の各ルーチンごとに入力レジスタ，出力レジスタ，ワークレジスタを宣言し，マイクロリントは各ルーチンについてレジスタの使用状況を解析し，これらの宣言と整合性がとれているかをチェックする。

(1) 各宣言とのチェック
- 入力レジスタ：本ルーチンで宣言されていない入力レジスタの内容が使用されるか？

Micro Lint for checking Micro Program Code
Nobuhito MIYAUCHI, Nobuyuki Ichiyoshi
(MELCO)　　　(ICOT)

・出力レジスタ：リターン時に出力レジスタに有効な値がのっているか？
・ワークレジスタ：ワークレジスタ以外のレジスタの内容が破壊されていないか？

(2) ルーチン呼び出し時のチェック
呼び出されるルーチンの入力レジスタに既に値が設定されているか？

(3) 各命令のチェック
値の有効でないレジスタの値を参照していないか？

ラベルを区切りとして別々のラベルに囲まれた一群のマイクロコードステップをコードブロック（または単にブロック）と呼ぶことにする。コードブロックへの実行の入り口は先頭のみだが，条件分岐により途中からの抜け出しはありうる。マイクロコードの解析はコードブロック単位に行なう。マイクロリントでは各コードブロックの入力レジスタ，出力レジスタ，ワークレジスタを解析して決定し，この(1)～(3)のチェックを行なう。

3-2. コードブロックのインターフェース宣言

使用レジスタの整合性をチェックするために，マイクロコーダが把握している各コードブロックでのレジスタ情報をインターフェース宣言としてソース中にコメントとして記入しておく必要がある。全てのコードブロックでコメントが必要なわけではない。

・入力レジスタ
　当コードブロック内で一度でもその内容が書き換えられる前に使用されるレジスタである。
・出力レジスタ
　当コードブロックを抜けるときまでにどのパスででも必ず値が設定されるレジスタである。
・ワークレジスタ
　当コードブロック内で一度でもその内容が書き換えられるとワークレジスタに相当する。

4. マイクロリントの処理の概要

4-1. 処理の手順

マイクロリントでは資源の参照と資源への値の代入と分岐処理などのマイクロプログラムのフローの情報が必要となる。そこで，マイクロコードの解析をする前に必要な情報のみを中間コードとして変換している。この後に中間コードを各ブロックごとにレジスタの使用の履歴を解析し，その使用レジスタの整合性を検証している。

この中間コード生成部はマイクロアセンブラを利用して容易に作成できた。マイクロリントの本体の解析部では，まず各ブロックごとにレジスタ情報の評価式を算出している。次に各ブロック間に及ぶレジスタ情報の評価式を整理して具体的に入力，出力，ワークに使用されているレジスタの種類を求める。この後に，各宣言のレジスタの種類と比較し，リントの結果を出力する。

(1) 宣言の読み込みと中間コード生成
(2) 各ブロックのレジスタ情報の評価式生成
(3) ブロック間にわたる評価式の整理
(4) 宣言のレジスタ情報との比較

4-2. 中間コード及び評価式の導出例

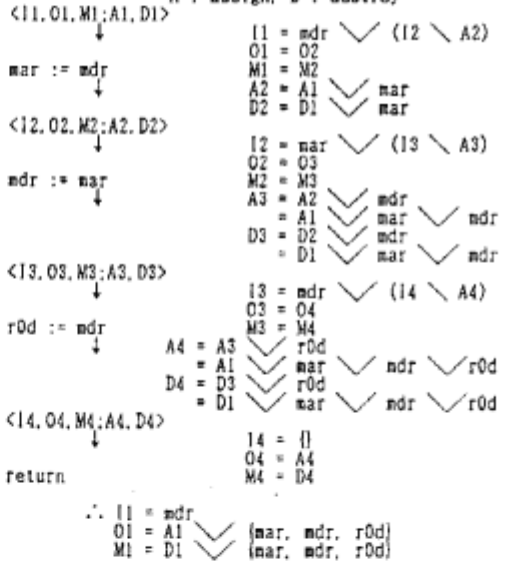
マイクロプログラムソース

```
:: # (input:  mdr).          % インターフェース宣言
:: # (output: r0d).          % インターフェース宣言
:: # (work:   mar, mdr).     % インターフェース宣言
subD_example:                % サブルーチンラベル
        mar := mdr.            % レジスタ操作命令
        read(mar, mdr).        % メモリリード命令
        r0d := mdr, return.    % リターン命令
```

中間コード

```
block(subD_example, [
        (mar := mdr),
        (mdr := mar),
        (r0d := mdr, return),
        [] ]).
```

呼び出された時点での処理のフロー上におけるレジスタの入力，出力，ワークの指標は，呼び出し時点に値が設定されているもの(assign)と代入されたことがあるもの(destroy)を用いて，以下のように，ブロック解析が行われる。
（∨は和集合，∧は要素合の演算記号を示す。）



5. おわりに

今回，マイクロプログラミングのコーディングとデバッグの効率を上げるためのツールとして汎用レジスタの使用方法の整合性をチェックするマイクロリントを作成し，試使用を始めたところである。本ツールは Multi-PSI/V2 のファームウェア用に開発したものだが，同様に他の計算機のファームウェアにも比較的容易に応用できると思われる。

参考文献

[1] K.Nakajima et al. :
"Distributed implementation of KL1 on the Multi-PSI/V2", Proc. of the 6th International Conference on Logic Programming. 1989