計算機アーキテクチャ 77－2
（1989. 7. 13）

推論プロセッサ UNIREDⅡのアーキテクチャ

島田 健太郎, 下山 健, 清水 剛, 小池 汎平, 田中 英彦
東京大学 工学部

概要

UNIREDⅡは、並列推論マシン PIE64 の要素プロセッサとしてコミッティド・チョイス型言語 FLENG を効率良く実行するために設計された推論プロセッサの第2版である。UNIREDⅡは並列マシンの要素プロセッサとして用いるのに適した構成となっており、並列記号処理を行う上で高い性能を持っている。本論文では、UNIREDⅡのアーキテクチャについて説明し、多重コンテクスト処理、ネットワーク・インターフェース・プロセッサ及びマネージメント・プロセッサとの協調処理など幾つかの特徴について議論を行なう。

The Architecture of the Inference Processor UNIREDⅡ

Kentaro Shimada, Takeshi Shimoyama, Takeshi Shimizu,
Hanpei Koike, Hidehiko Tanaka

Department of Electrical Engineering, Faculty of Engineering,
University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan

Abstract

We are developing the inference processor UNIREDⅡ. UNIREDⅡ is the second version of UNIRED, which is designed for the parallel inference machine PIE64 and for effective execution of a committed choice language FLENG. UNIREDⅡ is much suitable for an element processor of parallel computers, and has a high performance on symbolic processing. In this paper, the architecture of UNIREDⅡ is proposed, and some features, such as multi-context processing, fast context switching, and communicating with Network Interface Processors and Management Processors, are discussed.

1 初めに

我々は現在並列推論マシン PIE64[1] の製作を進めている。PIE64 では、対象言語として Committed Choice 型言語 FLENG[4]、及びその上位言語として並列オブジェクト指向言語 FLENG++[6] で書かれたプログラムを実行する。PIE64 は 64 台の推論ユニット (Inference Unit: IU) を 2 系統の回線交換ネットワークで結合した構成を採る [3]。各 IU にはネットワークと接続し FLENG 向きの高度な通信機能を提供する NIP(Network Interface Processor)[2]、IU 内での FLENG の実行処理を行なう推論プロセッサ UNIRED、及び全体の管理を行なう汎用プロセッサ SPARC がある。我々はこれまで並列マシンに於ける FLENG の実行方式について検討を行ない [8]、その結果に基づいて UNIRED の概要設計を行なってきた [1, 9]。しかし、実行方式について更に検討を重ねた結果、[1, 9] で設計された UNIRED アーキテクチャはより一層の改善が可能であることがわかった。そこで今回、UNIRED のプロセッサ・アーキテクチャについて、言わば第 2 版の概要設計を行なった。ここでは、その新しく提案された第 2 版の UNIRED(UNIRED II) のアーキテクチャについて報告する。

2 PIE64

初めに、並列推論マシン PIE64 の全体構成について簡単に説明する。

PIE64 は 64 台の推論ユニット (IU) を 2 系統の回線交換ネットワークで結合した構成を採る [3]。このネットワークは自動負荷分散機能等高度な機能を持つスイッチング・ユニット (SU) によって構成される 3 段オメガ網である。同じものが 2 系統あるのでネットワークの閉塞時にも柔軟な運用が可能であり、その自動負荷分散機構の利用の仕方も、1 系統は各 IU の負荷情報を基にし、もう 1 系統には各 IU のメモリ利用量を基にしてデータ配置のバランスを図る[1]などの種々の利用形態が可能である。

各 IU に於いては、ネットワークはネットワーク・インターフェース・プロセッサ (NIP) に接続される。IU 内では NIP、UNIRED、及び汎用プロセッサ SPARC は互いにコプロセッサ・コマンドバスで接続され、コプロセッサ・コマンドとそのリプライの形で互いに通信を行ない、協調動作する。NIP の受けるコマンドにはリモートなデータ・アクセス、ネットワークの自動負荷分散機能の利用の他、サスペンド / アクティベイトなどの同期機構、リモートな変数の束縛、並列環境下での Garbage Collection の支援など、FLENG 向きの高度な通信機能を提供するものがある [2]。

PIE64 に於ける FLENG の実行処理は、IU 内で NIP、UNIRED、SPARC の 3 種のプロセッサによって機能的に分割されて並列に処理される。NIP は他 IU へのリモートなデータアクセス及びゴール間の同期処理を行ない、UNIRED は専用の命令列によってゴール書き換えの処理を行なう。SPARC は Management Processor として、全体のゴール管理及び入出力管理を行なっている。

3 UNIREDIIの設計方針とその方策

UNIREDII は並列推論マシンの要素プロセッサとして用いるために設計された推論プロセッサである。対象言語としては FLENG を独自の機械命令体系にコンパイルしたものを実行する。基本的な設計方針としては、次の 2 点がある。

1. FLENG を汎用プロセッサで実行する場合に比べて、充分高速な処理能力を提供する。
2. 並列マシンの要素プロセッサとして用いるのに適した構成を採る。

UNIREDII では、まず 1. のためにタグ・アーキテクチャを採用、データのタイプを判別するためのタグのチェックとそのデータに対する処理を基本的に 1 命令で行なう。また、FLENG のような論理型言語の処理では頻繁に生ずる Read-Modify-Write 型の処理に対し、これをパイプラインの 1 スロットで行なうことを可能にするため、命令読み出しバスの他にデータ読み出しバスとデータ書き込みバスの計三つのメモリ・バスを持つ。

2. のためには、まずその基本的な方式として多重コンテクスト処理の機構を備える。多重コンテクスト処理とは、予め UNIRED 内に処理可能なコンテクストを複数受け付け、リモート・データ・アクセス等パイプライン・サイクルに比べて時間のかかる処理が行なわれた時に別個のコンテクストの命令を流すことによって動的にパイプラインの空白を充足するものである。この多重コンテク

[1] 現在、実際にはこのデータ配置の問題は、単に IU 間でのバランスを図るだけではなく、局所参照性を考慮した処理方式とすることを検討中である。

スト処理は他に、Jump 命令後のディレイスロットの充填や、パイプライン化されたレジスタ読み出しと書き込みの間のハザードの回避にも有効である[2]。

次に PIE64 のような並列マシンでは、あるコンテクストがプロセッサ間の同期処理で待ち合わせのために停止した時 (FLENG で言えばあるゴールがサスペンドした時)、別の実行可能なコンテクストを選んでコンテクスト・スイッチを行うようになるが、この処理にかかるコストはなるべく小さいことが望ましい。しかし最近の RISC プロセッサのように、プロセッサ内に高速アクセスが可能な大容量レジスタを備えた構成では、このコンテクスト・スイッチのコストはある程度大きくならざるを得なくなる。そこで UNIREDⅡではこの点を考慮し、コンテクスト・スイッチのコストを軽くするため、命令の一部に、あるコンテクストを実行するために必要な情報をレジスタとメモリの両方に置き、書換えが起こったらその両者の値を同時に変更すると言うことを支援するものを設けた。読み出しの時はレジスタに置かれた値のみを参照すれば良く、効率化が可能である。この機能を持つ命令の選択はコンパイラによって行われ、あるコンテクストに関する情報をレジスタのみに置くか、レジスタとメモリの双方に置くかは、その最適化の対象事項である。

4 UNIREDⅡプロセッサ・アーキテクチャ

4.1 主な特徴

UNIREDⅡは前節の設計方針に基づいて設計された。そのプロセッサ・アーキテクチャの主な特徴をまとめてみると、以下のようになる。

- 記号処理向きの機能としてタグ・アーキテクチャを採用、FLENG の処理を効率化している。
- 命令バス、メモリ読み出しバス、メモリ書き込みバスの三つの分離したメモリバスを持ち、バンド幅の広い並列メモリアクセスを行なう。
- 多重コンテクスト処理によって動的なパイプライン充足率の向上を行なう。

[2]もちろん、このように静的な命令解析で充足が可能なものは、コンパイラによって最適化することも行う。

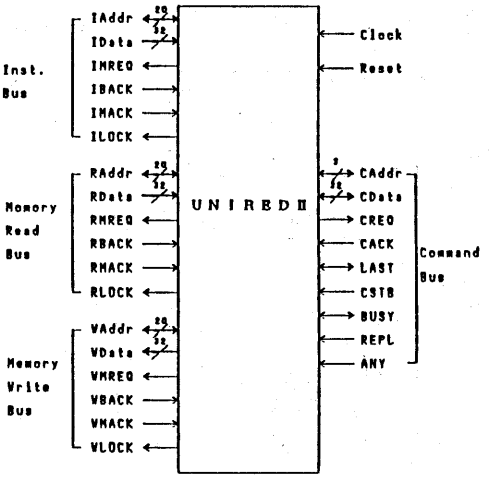

図 1: UNIREDⅡの外部信号線

- NIP、SPARC との間でローカルメモリを共有するため、パイプライン式のアービトレーションを行なうメモリアクセス方式を持つ。
- コプロセッサ・コマンドバスを持ち、NIP、SPARC との間でのコマンド / リプライの通信プロトコルをサポートしており、これらのプロセッサと協調動作を行なう。
- ページング方式の Garbage Collection[7] に対応する機能を持つ。また更に一括型の Garbage Collection を支援する機能も設ける。

現在、同時に処理する多重コンテクスト数は、ハードウエアの実現性から、4 としている。UNIREDⅡは、総端子数 256 ピンのゲートアレイとして実現される。外部信号線を図 1に示す。

4.2 データ形式

UNIREDⅡで扱うデータ形式を図 2に掲げる。タグ部及び値部を併せて 1 ワード 32 ビット構成である。PIE64 ではそれぞれの IU に分割してメモリが置かれることに対応して、リスト等へのポインタは IU 番号 (IU id) と IU 内でのオフセット・アドレスの組で表現されている。なお、ゲートアレイ化する時の端子数の制限のため、実際には IU 内オフセット・アドレスは 20 ビットのみ有効である。また、図で MK 及び BB は Garbage Collection のためのマークビット領域である。

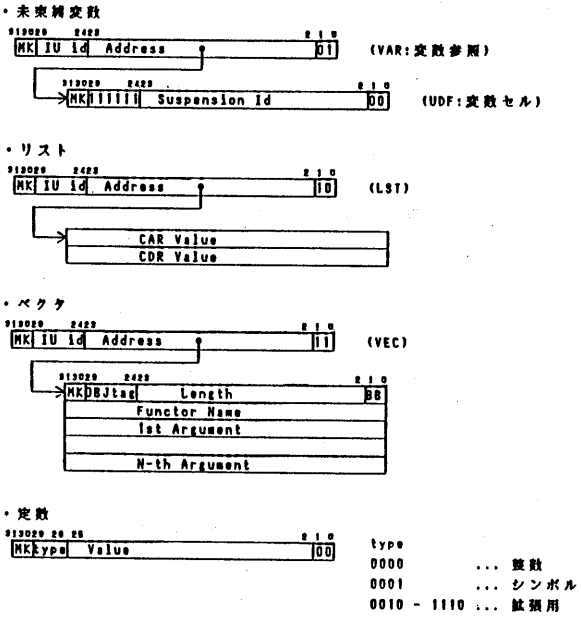

図 2: UNIREDIIに於けるデータ形式

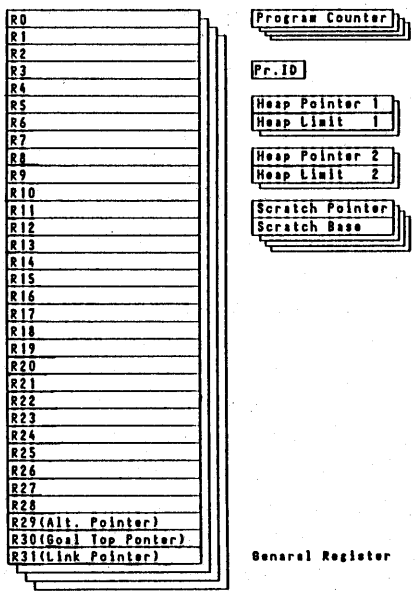

図 3: UNIREDIIのレジスタ・セット

4.3 レジスタ・セット

UNIREDIIにおいて、命令で意識されるレジスタ・セットとしては、現在次のようなものを予定している (図3)。

1. プログラムカウンタ
   20 ビットのプログラムカウンタを 4 コンテクスト分で 4 本持つ。
2. プロセッサ ID レジスタ
   PIE64 に於ける 64 台の IU を識別するための 6 ビットの ID を格納する。
3. 汎用レジスタ
   汎用レジスタは 32 ビット長であり、1 コンテクスト当り 32 本で、4 コンテクストで計 128 本持つ。これらの内、特に各コンテクスト毎に R31 は CALL 命令実行時のプログラムカウンタ退避用のリンク・レジスタ、R30 はそのコンテクストが処理中のゴール・フレーム (ゴールの引数を集めた特別なベクタ) の先頭を指すゴール・トップ・ポインタとして使われる。また、R29 はある候補節の頭部単一化処理の際、次の候補節の命令を指す Alternative Pointer として使うこともできる。
4. ヒープ・レジスタ
   与えられたヒープ領域の現在の未使用域の先頭を指すヒープ・ポインタと未使用域の終わりを保持するヒープ・リミットの 2 種類のレジスタがある。ページング方式の Garbage Collection に対応するため同じもの 2 セットあり、更に常に 1 ページ分の領域を内部でバッファリングするため、1 セットはそれぞれ表裏の 2 組のヒープ・ポインタとヒープ・リミットからなる。
5. スクラッチ領域レジスタ
   他 IU のメモリ上にあるリモートな構造データへの参照が生じた時に、その実体のローカル・コピーを置くスクラッチ領域を管理するレジスタである。各コンテクスト毎に独立しており、それぞれスクラッチ領域の先頭を指すスクラッチ・エリア・ベースと現在の使用域の終わりを指すスクラッチ・エリア・ポインタの 2 本のレジスタからなる。

4.4 内部構成

UNIREDIIの内部構成を図 4に示す。パイプライン構成が採られており、主要な部分のパイプライ

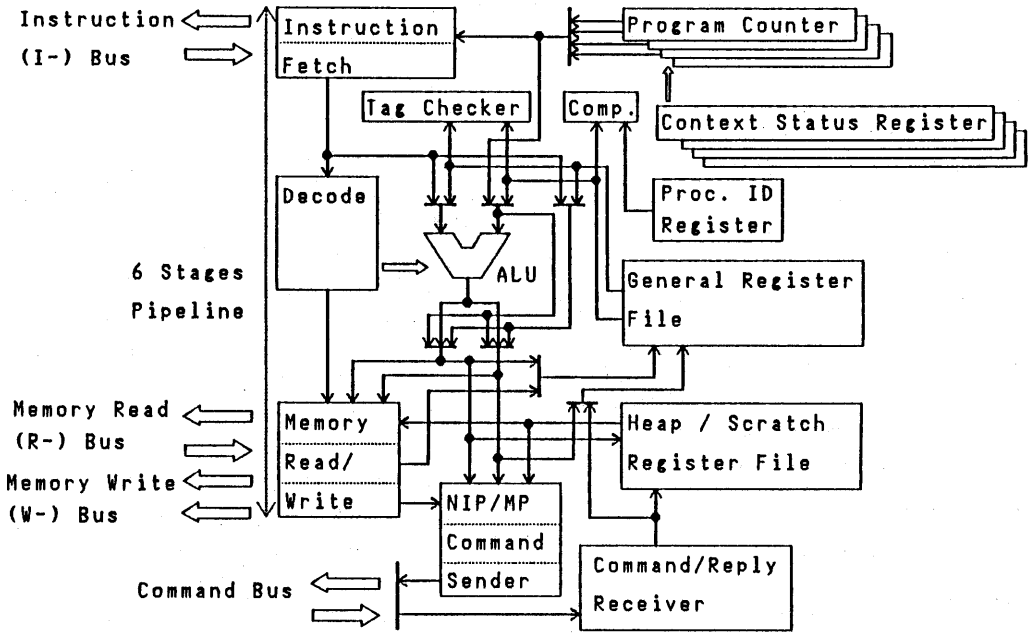

図 4: UNIREDIIの内部構成

ン段数は6段である。NIP、SPARCと共有されているローカルメモリのアクセスは、アービトレーション・フェーズとデータ転送・フェーズの2フェーズに分かれており、これはパイプライン化されている。即ち、主要部の6段パイプラインの内1、2段目が命令フェッチのためのメモリ・アクセスであり、3段目で命令デコード、レジスタ読み出し及びメモリ・アドレス計算等を行なう。更にメモリの読み出しが行なわれる場合には4段目及び5段目、メモリの書き込みには5段目及び6段目の2段づつが使われる。メモリの読み出しと書き込みを1段ずらしてあるのは、メモリ-メモリ間の1ワードのデータ転送を行なうLoad and Store命令、及び未定義変数であることをロックをかけながら読み出しで確認し値の束縛を行なうBind命令を1命令としてサポートするためである。なお、NIP、SPARCに対するコプロセッサ・コマンドを発行する時にはパイプラインの6段目でコマンドの生成がされ、7段目でコマンド・ワード、8段目でパラメータ・ワードの転送が行なわれる。

4.5 主なFLENG実行処理の機構

特にFLENGの実行処理向きに考えられた機構の内、主なものをいくか説明する。

4.5.1 Jump and Load

主要処理部の6段パイプラインにおいて、1、2段目で命令フェッチが行なわれた後、3段目でレジスタの読み出しを行ない、条件Jump命令であればこの段で条件判断をしてJumpの処理を行なう(プログラム・カウンタの書き換えを行なう)。この時後段のメモリ・アクセス・ステージが空くので、例えばJump命令に続くLoad命令がある場合、この2命令を複合してJump and Loadを行なう命令を設けた。

このような場合の典型的なものは、変数の参照の手繰りを行なうDereference命令である。Dereference命令では、あるレジスタの内容が変数への参照であることをそのタグ部で判断すると、後段のパイプライン・ステージにその参照先を読み出して元のレジスタにLoadさせる処理をさせつつ、自分自身へのJumpを行なう。するとDereference命令が再実行されて、先ほど後段の処理でメモリからLoadされた参照先の内容が同じレジスタに

入っているはずなので、今度はそれが読めることになり、変数参照を一段手繰ったことになる。手繰った先が再び変数への参照であれば、同じ処理が繰り返される。

このJump and Load型の命令は、他にも最適化のための複合命令として幾つか用意した。

なお、UNIREDIIでは多重コンテクスト処理を行なっており、Jump命令によるパイプライン・ブレークが起きても別のコンテクストの命令がパイプラインを満たしていることがあるので、通常のRISCプロセッサのようなディレイド・スロットの同一コンテクスト内での利用は特に行なわない。即ち、3ステージ目でJumpの処理が行なわれた後Jump先の命令がフェッチされる前にパイプラインに投入されてしまった同一コンテクストの命令は、無効化される。

4.5.2 ローカルな変数のバインド

UNIREDIIでは、変数が自IU内のローカル・メモリにある場合に限り、ロックを掛けて排他的に値の束縛を行なう機能を持っている。他IUにあるリモートな変数の場合には、NIPにBINDコマンドを発行して束縛処理を依頼する。

ローカルな変数のバインドを行なう方式は次のようである。Bind命令がパイプラインの1、2段目でフェッチされ3段目でデコードされると、4段目において束縛しようとする変数の内容を読み出すためのアービトレーションが行なわれる。それに成功し、そのメモリ・ワードへのアクセス権を得ると、そのメモリ・アドレスに対しロック信号を出すようにする。図1で見る通り、三つのメモリバスのアドレス信号(IAddr, RAddr, WAddr)は双方向であり、アクセス権を得てメモリアドレスを出力していない間はメモリバス上のアドレスを監視し、そこにロックを掛けたアドレスが現れるとロック信号(ILOCK, RLOCK, WLOCK)を出力して、外部のアービタにそのアドレスをロック中であることを通知する。続くパイプラインの5段目において変数の内容が読み出されて未束縛であることが確認されると同時に、書き込みのためにアービトレーションを行なって引続き書き込みのためにアクセス権を保持し、6段目で実際の書き込みを行なう。ロック信号は5段目で書き込みのためにアクセス権を保持することに成功した時点で外される。

4.5.3 リモートなデータの読み出し

UNIREDIIでは、リモートなデータの読み出しはメモリ・アクセス命令においてそのIU idを判定して、自動的にNIPへのコプロセッサ・コマンドに置き換える機能を持っている。

例えばDereference命令において、指定された変数参照のIU idが自IUのものと異なっているとリモートであると判断され、ローカルメモリに対するアクセスは行なわれない。代わりにパイプラインの6～8ステージ目を用いてNIPにREAD1コマンド[3]を発行し、リモートな変数の内容を読み出すことを依頼する。この時、読み出された変数の内容を受け取るべき元のレジスタにはマークを付けておき、NIPからリプライが返ってくる前にそのレジスタを読み出そうとした時は、そのコンテクストを中断状態にする。この中断状態は、リプライを受け取ってレジスタに返ってきた変数の内容がセットされた時点で解かれる。

リストなどの構造データがリモートである時の様子は、また少し異なる。リモートな構造データの参照は、基本的にはまずCopy if Remote命令でIU idを調べ、それが自IUのものと異なっていたらスクラッチ領域から必要な大きさを割り当てて、NIPにリストの場合はREAD2コマンド[3]、ベクタの場合にはREADNコマンド[3]を発行し、スクラッチ領域に実体をコピーさせて、そのコピーを参照して行なう。この時、ベクタへの参照の場合には不用意にスクラッチ領域を越えるような大きさのものをコピーしてくるのを避けるため、そのような可能性のある所にはコンパイラによってベクタの先頭ワードのみを読み出す命令、及びそこに書かれているlength値を調べるCheck Vector Top命令を挿入する。

なお、構造データの扱いについては、最適化のためにDereference命令との複合命令も幾つか用意した。

4.5.4 メモリ管理

UNIREDIIにおけるヒープメモリ・ページの管理について説明する。UNIREDIIでは与えられたヒープ・ページをヒープ・ポインタとヒープ・リミットの二つのレジスタで管理している。あるオブジェクトのための領域を新しくアロケートしようとした時、その大きさを足すとヒープ・ポインタの値

[3]NIPが受け取るリモート・データ・アクセスのコマンドの詳細については文献[2]を参照されたい。

がヒープ・リミットの値を越えるのであれば、そのヒープ・ページ内にアロケートすることを止めてManagement Processorである SPARC に向けてRequest Pageコマンドを発行して新しいページを要求する。この時、SPARC から新しいページが到着するまで待っているとスループットが悪くなるので、常に1ページ分先に要求を出して受け付けておき、裏のヒープ・ポインタとヒープ・リミットの組でバッファリングしておく。すると、Request Pageコマンドを発行した後、該当するヒープ・レジスタの組の裏と表を切り替えるだけで処理を続けることができる。新しく到着したヒープ・ページは裏に回った方の組で、また保持しておけば良い。

なお、ヒープ・レジスタが全体で2セットあるのは、常に単一参照であるゴール・フレームと、それ以外の引数中の構造データ等とをアロケートするページを分けているためである。ゴール・フレームのおかれるヒープ・ページはSPARCによってその中に存在するゴール・フレーム数を管理されており、ページング方式でインクリメンタルなGarbage Collection[7] が行なわれている。

5 命令セット

5.1 命令セットの概要

最後にUNIREDIIの命令セットについて、その概要を表1にまとめておく。基本的には前節で示したような機構を実現するための命令を持っている。良く使われる WAM 型命令 [5] との違いは主に、

- FLENG が Committed Choice 型言語であることによる命令機構の簡素化。
- 並列実行に対応した命令仕様。
- パイプラインの構成を意識し、これを有効利用して最適化を行なうための複合命令。
- 逆にハードウェアのコストを下げるためにすべて1命令1ワード構成とし、1クロックで実行するための命令の分解。

等が行なわれていることである。なお、3節で述べた、コンテクスト・スイッチのコストを軽くする目的でゴールの引数をレジスタとメモリの両方に置く手法を支援するために、例えば Load and Store 命令は読み出すメモリ・アドレスとそのメモリワードを書き込むメモリ・アドレス及びレジスタの三つのオペランドが指定できるようになっている。

```
append([H | T], X, Y) :- Y = [H | Z], append(T, X, Z).
append([], X, Y) :- X = Y.

append/3:
        dcll    r1, $1, r28, r4         ; [H |
$2:
        tlds    r1, 1, gtp, 2, r1       ; T]
        alst    r5                      ; [
        st      r5, 0, r4, r0           ; H |
        sudf    r5, 1, r6               ; Z]
        bind    r5, r3, r7              ; Y = [ H | Z]
        jntg    r7, udf, $fail
        st      r6, gtp, 4, r3          ; Z)
        exll    r1, $2, r28, r4         ; [H |
$1:
        dfcc    r1, nil, $check, r28    ; []
        derf    r2, $3, r28             ; X = Y
$3:     derf    r3, $4, r28
$4:     cvos    r2, r3, r5, r3
        bind    r5, r3, r7
        jntg    r7, udf, $fail
        end     gtp, mp
```

図 5: append のコンパイル例

5.2 コンパイル例

簡単なコンパイル例として、append の例を図5に掲げておく。この例では、append が再帰的に実行される場合、そのループを構成する命令数は8である。

6 終りに

現在、UNIREDIIは詳細設計を進めている段階である。最終的にはCMOSゲートアレイとして実現し、クロックは10MHzとする予定である。これからの課題としてはシミュレーション等も含めたFLENGの実用プログラムに対するプロセッサ・アーキテクチャの評価、実装時の効果的なメンテナンス / デバック方法の検討、及び実際にPIE64に組み込んでの64台の並列動作の評価等が挙げられる。

参考文献

[1] Koike,H.and Tanaka,H.: *"Multi-Context Processing and Data Balancing Mechanism of the Parallel Inference Machine PIE64"* Proc. of Fifth Generation Computer Systems, Tokyo, Japan, November 1988.

[2] 清水, 小池, 島田, 田中: "並列推論マシン PIE64 のネットワーク・インターフェース・プロセッサ" 情報処理学会並列処理シンポジュウム JSPP'89, 1989 年 2 月

[3] 高橋, 田中: "並列推論マシン PIE64 におけるインターコネクションネットワークの作成と評価" 情報処理学会計算機アーキテクチャ研究会 76-1, 1989 年 5 月

[4] Nilsson,M. and Tanaka,H: *"FLENG Prolog - the Laguage which turns Spuercomputers into Prolog Machines"* Proc. of Japanese Logic Programing Conference '86, ICOT, June, 1986.

[5] Warren,D.H.D.: *"An Abstract Prolog Instruction Set"* Technical Note 309, Artificial Intelligence Center, SRI, 1983.

[6] 中村, 小中, 田中: "並列論理型言語 FLENG に基づいた並列オブジェクト指向言語 FLENG++" 日本ソフトウェア科学会, オブジェクト指向計算に関するワークショップ WOOC'89, 1989

[7] Xu, Shimada, Shimizu, Koike, Tanaka: *"Distributed Garbage Collection for the Paralle Inference Machine: PIE64"*情報処理学会第 38 回全国大会 5U-7, 1989 年 3 月

[8] 島田, 小池, 清水, 田中: "PIE64 上での FLENG 実行方式" 情報処理学会第 37 回全国大会 5N-6, 1988 年 9 月

[9] 島田, 清水, 小池, 田中: "並列推論マシン PIE64 の推論プロセッサ UNIRED の概要" 情報処理学会第 38 回全国大会 5U-9, 1989 年 3 月

表 1: UNIREDIIの主要命令セット

Dereference	derf	dereference
	dfcl	dereference and check list
	dcll	dereference and check list and load car
	dfcv	dereference and check vector
	dcvl	dereference and check vector and load top
	dfcc	dereference and check constant
Execute	exec	execute
	exel	execute on list
	exll	execute on list and load car
	exev	execute on vector
	exvl	execute on vector and load top
	exct	execute on constant
Manipulate Structure	cpir	copy if remote
	cprr	copy if remote with register
	cvtp	check vector top
	cvtr	check vector top with register
Load / Store	ld	load
	tgld	tagged load
	ldst	load and store
	tlds	tagged load and store
	st	store
	sudf	store undefind code
	stim	store immediate
Active Unification	bind	bind variable
	bdim	bind with immediate
	cvos	check variable order and swap
Heap Allocation	allc	allocate
	alst	allocate and store
Control	jump	jump
	call	call
	jcmp	jump on compare
	jncp	jump on not compare
	jtag	jump on tag
	jntg	jump on not tag
	stop	stop
Coprocessor	cpcm	send coprocessor command
	susp	send suspend command
	fail	send fail command
	fork	fork new goal
	end	send end_reduce command
Set	setc	set constant
	sett	set mark and tag
Arithmetic and Logical	add	add
	sub	subtract
	and	bit-wise and
	or	bit-wise or
	xor	bit-wise exclusive or
	not	bit-wise not
	shl	shift left
	shr	shift right
	asr	arithmetic shift right
Comparison	grt	greater than
	geq	greater than or equal
	less	less than
	leq	less than or equal
	equl	equal
	neq	not equal