AI言語向きRISCアーキテクチャ

2W-3

丸山 勉、神津 信一†、横田 実、森島 潔
日本電気(株) C&Cシステム研究所 †半導体応用技術本部

1 はじめに

本稿では、現在評価中であるAI言語向きRISCアーキテクチャについて述べる。1チップ化することを念頭に現在のデバイス技術を踏まえてアーキテクチャの設計を行なった。どの様な命令セットがAI的処理の高速実行に有効であるかを中心に述べる。

LISP,PROLOG等の処理では、一般には分岐が多いと言われているが、応用プログラムの評価ではメモリアクセス命令、レジスタ間転送命令等な単純の命令の実行頻度が最も高い。従って、分岐等の高速化を行なうとともに、単純な命令の高速化が不可欠である。このため、本アーキテクチャではクロックスピードの高速化を重視してRISCアーキテクチャを採用した。また、分岐の高速化を実現するために、よく分岐すると考えられる方向の高速化のための分岐命令および複合命令等を用意した。さらに、組み込み関数呼び出しの高速化および単純な命令の同時実行による高速化等を図った。

結果として、簡単なベンチマーク等を用いたサンプルコーディングでは、MIPS等のRISCプロセッサに対して同じクロックスピードで動作した時、LISPの場合で約1.5~2倍、PROLOGの場合で2~3倍の高速化を達成できる見通しを得た。

2 アーキテクチャ設計方針

PROLOGマシンCHI-II[1]及びLISPマシンLIME[2]での評価結果をみると、メモリアクセス命令、レジスタ間転送,即値の代入等の簡単な命令、分岐命令、関数呼び出し等の命令の実行頻度が高い。これらの命令の実行頻度を考えて以下のような設計方針をとった。

2.1 速いクロックスピードの実現

より高速なクロックスピードを実現するためにRISCアーキテクチャを採用した。上述したようにLISP, PROLOG等の実行においてもメモリアクセス、レジスタ間転送などの簡単な命令頻度が最も高いため、汎用マイクロプロセッサと同程度のクロックスピードを実現することは非常に重要である。

2.2 メモリアクセスの高速化

AI的処理においては特にメモリアクセスが多い。メモリアクセスのディレイタイムの高速化はデバイス技術により決定するため、アーキテクチャとしては以下の機能を実現した。

1. 連続リード/ライト時の1ワード/1クロックサイクルのスループットの実現
2. レジスタのpost increment/decrement
   スタックアクセス等に有効である。
   命令例 read Rdata, Raddr++
   /* Rdata = *Raddr & Raddr += 4; */
3. ベース+オフセットによるメモリアクセス
   nn ローカル変数等のアクセスに不可欠である。
   命令例 read-with-offset Rdata, Raddr, Offset
   /* Rdata = *(Raddr+Offset); */

2.3 分岐方向の高速化

RISCでは一般に分岐の高速化のためにディレイド分岐が用いられてるが、LISP,PROLOGでは、命令列が

分岐命令 → メモリアクセス → 分岐命令 →

というサイクルになることが多いため、ディレイドスロットを埋めることはそれ程容易ではなく単なるディレイド分岐のみでは不十分である。また、多くのタグ判断は例外的な条件の検出に用いられることが多いため本アーキテクチャでは以下のような方法によってメインバスの高速化をはかっている。

1. SPUR [3] に見られるような2種類の分岐命令
   (a) 分岐する方を優先する分岐命令
   (b) 分岐しない方を優先する分岐命令
2. メモリアクセス & 条件分岐命令の組み合わせ

1-(a)では分岐が起きた場合には通常のディレイド分岐となり、分岐が生じなかった場合にはディレイドスロットの命令の実行を無効化してその次の命令から実行する。1-(b)では分岐が生じた場合にはディレイドスロットに相当する命令の実行が無効化され分岐先の命令から実行されるが、分岐が生じなかった場合にはディレイドスロットの命令から実行される。このような分岐命令によりメモリアクセス命令を自由にディレイドスロットに置くことができるため、分岐予測が正しかった場合の処理を高速化することができる。

命令例
```
jump-if-[not-]-condition[-delayed] Condition, Offset
  /* if(ConditionIsTrue) PC+=Offset */
jump-if-tag[-not]-equal[-delayed] Rd, TAG, Offset
  /* if(TagOf(Rd)==TAG) PC+=Offset */
jump-if-[not-]-bit[-delayed] Rd, N, Offset
  /* if(NthBitOf(Rd)==1) PC+=Offset */
```

delayedが指定された場合には1-(a)、そうでない場合には1-(b)の分岐となる。notが指定された場合には条件判断の否定がとられる。

2.においては、組み合わされた分岐条件の判断結果によって、メモリアクセスが無効化される。このため、メモリアクセスと同時にメモリアクセスに用いるデータの条件判断を実行することができ、リストの第1要素の読み出し等を高速化することができる。

例 read-with-check Rdata, Raddr &
jump-if-tag-not-equal Raddr, CONS, Offset
Raddrのタグが CONS ならば read 命令を実行、それ以外ならば read 命令を無効化して分岐

A RISC Architecture for AI Languages
Tsutomu Maruyama, Shinichi Kouzu,
Minoru Yokota and Kiyoshi Morishima, NEC Corporation

2.4 組み込み関数呼び出しの高速化

LISP,PROLOGでは組み込み関数が多用されるので高速化が重要である。組み込み関数の呼び出しの高速化を実現するために以下のような機能を採り入れた。

1. 絶対番地へのコール命令 & 割り込みマスク
   例 call-firmware-delayed Absolute-Address
   /* PC = Absolute-Address */
2. 複数の引数の同時転送(相手先は引数用レジスタ)及び関数の実行結果の書き込みレジスタの指定
   例 move-arguments Rarg1 Rarg2 Rarg2 Rreturn
   /* R32 = Rarg1, R33 = Rarg2, R34 = Rarg3, SpecialRegister = NumberOfRreturn */
3. 関数の実行結果の書き込みレジスタの間接指定によるレジスタ間転送
   例 move-indirectly Rresult
   /* R[SpecialRegister] = Rresult */

2.5 簡単な命令の同時実行による高速化

本アーキテクチャでは、以下に示すような命令同時実行による高速化の機能をできるだけ採り入れた。

1. 各種命令 + レジスタ間転送命令
   例 read Rdata, Raddr & move Rdest, Rsrc
2. 各種命令 + ジャンプ命令
   例 add Rd, Ri, Rj & jump-delayed Offset
3. 各種命令 + 条件分岐命令
   例 move Rd, Rs &
   jump-if-tag-not-equal Rs, CONS, Offset

3 ハードウェア概要

ハードウェアの概要を以下に示す。

1. データ幅は40ビット(GC用タグ2ビット + タグ6ビット + データ32ビット)
2. 4段のパイプライン(各ステージ間では演算結果をすぐ利用する場合の待ちを防ぐためにデータをバイパス)
3. 64個の汎用レジスタ(レジスタウィンドウは用いてはいない)。
4. 命令はオペコード、複合命令指定フィールド及び4つのオペランドからなり、命令の数は約100種。
5. 数Kワードの命令/データキャッシュ(on-chip)を想定(現状では数Kワードのものを想定しているができるだけ大容量化が望ましい)

4 プログラミング例

以上述べた命令セットを実際にどのように用いるかを図1に示す。図1はLISPの関数であるassocのコーディング例の一部である。上記の組み込み関数用命令によって、$assocのコール/リターンがほとんどオーバーヘッドなしに実現されているのがわかる。また、メモリアクセス & 分岐命令によってリストのCAR部の読み出し(図1中の(car Rlist), (caar Rlist)の部分)が高速に実行されていることがわかる。更に、複合命令(subtract&jump-delayed)および上記の1-(b)タイプの分岐しない方を優先する分岐命令の組み合わせにより、ループが高速に実現されている。このようにしてMIPS等のRISCプロセッサに対して、約2倍の高速化が実現できる。

5 終りに

以上、AI言語の高速実行を目標としたRISCアーキテクチャについて述べた。現在、このアーキテクチャをゲートアレイを用いて製作中である(キャッシュ制御部を含めて7種)。今後、これらのチップを用いてアーキテクチャの評価を行なう予定である。

謝辞 本研究に対して有益なコメントを頂いた日本電気C&Cシステム研究所の大野部長および半導体応用技術本部の永尾主任に感謝致します。

参考文献

[1] Habata,S. Nakazaki, R., Konagaya, A., Atarashi, A. and Umemura, M., "Co-operative High Performance Sequential Inference Machine: CHI", in *Proc. ICCD'87*, New York, 1987
[2] 横田 実, 越前 孝, 越前 章子, "LISP マシン LIME", 計算機アーキテクチャ研究会予稿 74-6, 1989
[3] Taylor, G.S., Hilfinger, P.H., Larus, J.R., Patterson, R.A. and Zorn, B.G., "Design Decisions in SPUR", in *Computer*, IEEE, 1986

```
; 呼び出し側
    call_firmware_delayed Rretaddr, $assoc
    move_arguments Ritem, Rlist, Rany, Rresult

; 関数定義
$assoc:

; タグの最上位ビットが0ならばタグはSYM,INT等、
; 1ならばCONS, VECTOR等。

    jump_if_bit R32, 37, c_assoc
s_assoc:        ; 簡単な比較でよいもの
; (car Rlist)
    read_with_check R35, R33++
        & jump_if_tag_not_equal R33, CONS, eol
; (cdr Rlist)
    read R33, R33
; (caar Rlist)
    read_with_check R36, R35
        & jump_if_tag_not_equal R35, CONS, nil?
; 比較(タグ & データの比較)
    subtract R36,R36,R32 & jump-delayed s_assoc
    jump_if_not_condition Tag&DataEqual, found
found:  ; 等しい要素がある場合
    jump_indirectly_delayed Rretaddr
    move_indirectly R35

eol:    ; LISTの最後まで処理を行なった場合
    jump_if_tag_not_equal R33, NIL, error
    jump_indirectly_delayed Rretaddr
    move_indirectly Rnil_register

c_assoc:        ; 内容の比較が必要なもの
    .....
```

Fig.1 Sample Coding of ASSOC