

VPP500 スカラプロセサの特徴

中島 康彦 北村 俊明 田村 秀夫 滝内 政昭

富士通(株)

VPP500 スカラプロセサに関し、まずアーキテクチャの特徴である、1) 最大3操作を毎サイクル発行する64ビット長の長形式命令語; 2) PC相対アドレス指定による条件分岐操作; 3) データ依存関係に基づき実行順序を変更できる非同期操作; 4) 例外および未完了操作の複数同時表示機構; について述べる。次にインプリメントの特徴である、1) ハードウェア量および分岐ペナルティを削減する少段数パイプライン; 2) 2個の固定小数点演算を毎サイクル行う機構; 3) 2個の浮動小数点演算を毎サイクル行う機構; 4) 2本のレジスタへの読み出しを毎サイクル行う主記憶参照機構; について述べる。最後に、実測結果に基づき性能に言及する。

Characteristics of the VPP500 Scalar Processor

Yasuhiko Nakashima Toshiaki Kitamura Hideo Tamura Masaaki Takiuchi

Fujitsu Limited

1015, Kamikodanaka Nakaharaku, Kawasaki 211, Japan

First, we show the architectural features; 1) The 64bit LIW (Long Instruction Word) which issues three operations every cycle; 2) The PC relative conditional branch operation; 3) The asynchronous operations and the out-of-order execution; 4) The interruption facility which reports plural exceptions and unexecuted operations. Next, we show the implementational features; 1) The three stage pipeline which reduces the hardware cost and the branch penalty; 2) The two fixed-point ALUs; 3) The pipelined floating-point units; 4) The wide data path which enables the data transmission from the cache to the paired registers. Last, we mention the performance.

1 はじめに

VPP500は、高性能スカラプロセッサに各々ベクトル機構を付加した並列ベクトル計算機システムである。本システムの特徴は、1) 主記憶分散配置型の並列プロセッサ構成により大容量主記憶および強力な主記憶データ供給能力を実現している点; 2) クロスバーネットワークを採用し例えばプロセッサ間に分散配置した行列を転置する際に必要なプロセッサ間一斉通信を高速化している点; 3) 個々のプロセッサを高性能ベクトルプロセッサとすることにより高い実効並列処理性能を実現している点; である[1]。図1に示すように、ハードウェアはシステム制御を行うCP(Control processor)、演算処理を行うPE(Processing element)、これらを相互接続するクロスバーネットワークから構成される。PEはスカラプロセッサ、ベクトルユニット、データ転送ユニットおよび主記憶から構成される。CPはベクトルユニットの代わりにグローバルシステムプロセッサとの結合機構を有する。本論文ではVPP500スカラプロセッサの特徴について詳述する。

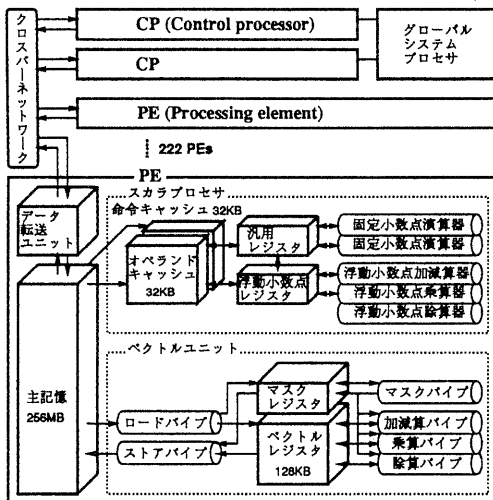


図 1: VPP500 システム構成

さて、従来のVPシリーズベクトル計算機[2]は、Mシリーズ計算機と同じスカラプロセッサ・アーキテクチャを採用してきた。しかし、長い歴史を有するこのアーキテクチャは、命令実行順序の保証、割り込み発生時のPC(Program Counter)値の保証等を規定していることから、命令の並列実行や追い越し処理などインプリメントの工夫による高速化を大きく妨げている。今後、スカラプロセッサの性能を飛躍的に向上させるためには、ソフトウェアに内在する命令の並列性を最大限に引き出し、ハードウェアの並列処理に効果的に写象することのできるアーキテクチャを導入しなければならない。このような背景を基に、我々は、新しいアーキテクチャを探るVPP500スカラプロセッサを開発した。本アーキテクチャの主な特徴を以下に列挙する。

1) 1命令語当り1個のベクトル操作または1～3個のスカラ操作を一度に発行可能とする64ビット長の長形式命令語(Long Instruction Word)方式を採用することに

より、スーパスカラ方式において必要とされる並列実行可能な操作を検出するためのハードウェア機構を不要とし、また超長形式命令語(Very Long Instruction Word)方式の一般的な短所である命令サイズ増大を抑えたこと。

2) 条件分岐操作における分岐先アドレスの指定方法をPC相対アドレスのみとすることにより、分岐先アドレスの計算および分岐先命令のプリフェッチを高速に行うインプリメントを可能としたこと。

3) 非同期操作と呼ぶ実行に複数サイクルを要する固定小数点乗算操作、浮動小数点演算操作、主記憶参照操作およびベクトル操作について、操作を並列実行する際の妨げとなる条件コード更新を必要最小限に抑えと共に、先行する非同期操作の実行完了を待たずに後続命令を発行可能とする非同期実行機構を設け、さらに、データ依存関係を崩さない範囲内において非同期操作の実行順序をハードウェアが変更できる規定としたこと。

4) 例外を検出した際には、実行中の操作のうち可能なものは実行を完了し、例外を検出した操作との間にデータ依存関係があるために実行不可能なものは未実行状態とした上で、例外を検出した操作の個々のアドレスが不明であってもソフトウェアが例外処理および例外処理からの復帰を正しく行うことができるよう、ハードウェアが複数の例外および未実行状態に関する十分な情報をソフトウェアに通知する機構としたこと。

次に、本アーキテクチャを最大限に利用したインプリメントの主な特徴を以下に列挙する。

1) 固定小数点演算操作および分岐操作のためのパイプラインを3段と短くし、ハードウェア量を抑えとともに、高速な分岐先アドレス計算機構、および、次アドレスまたは分岐先アドレスから毎サイクル2命令をプリフェッチする機構を装備することにより、分岐ペナルティをほぼ0としたこと。

2) 各々独立した条件コードレジスタを有する2個の固定小数点演算器を装備し、1命令語中に記述された2個の固定小数点演算操作を毎サイクル実行可能としたこと。

3) 各々1個の浮動小数点加減算器、乗算器、除算器を装備し、特にパイプライン化された加減算器および乗算器により、1命令語中に記述された加減算操作と乗算操作の組を毎サイクル発行可能としたこと。

4) 主記憶の連続アドレスから一度に2本の32ビット汎用レジスタまたは2本の64ビット浮動小数点レジスタへデータを読み出すことができる主記憶参照機構を装備することにより、毎サイクル2個の主記憶オペランドを読み出し可能としたこと。

本論文では、まず第2章においてアーキテクチャの特徴について述べ、続く第3章においてインプリメントの特徴について述べる。そして第4章においてLivermore14ループを用いた実測結果に基づき性能に言及する。

2 アーキテクチャの特徴

2.1 概要

VPP500スカラプロセッサは、ゼロレジスタを含む33本の32ビット汎用レジスタ(レジスタ1は、ユーザ・モード用とスーパーバイザ・モード用に各々1本ある)、ゼロレジスタを含む32本の64ビット浮動小数点レジスタ、内

蔵 TLB および内蔵キャッシュを有し、32 ビット固定小数点演算、IEEE754 準拠の 32/64 ビット浮動小数点演算、32 ビット仮想アドレス空間、32 ビット実アドレス空間およびコプロセッサインタフェースを提供する汎用プロセッサである。また、主記憶空間とは別に、1024 本分のスカラ制御レジスタ空間、8192 本分のシステム制御レジスタ空間を有しており、制御レジスタを主記憶空間に割り付けた場合に生じるアドレス変換のオーバーヘッドおよび TLB の消費を回避している。

2.2 命令語

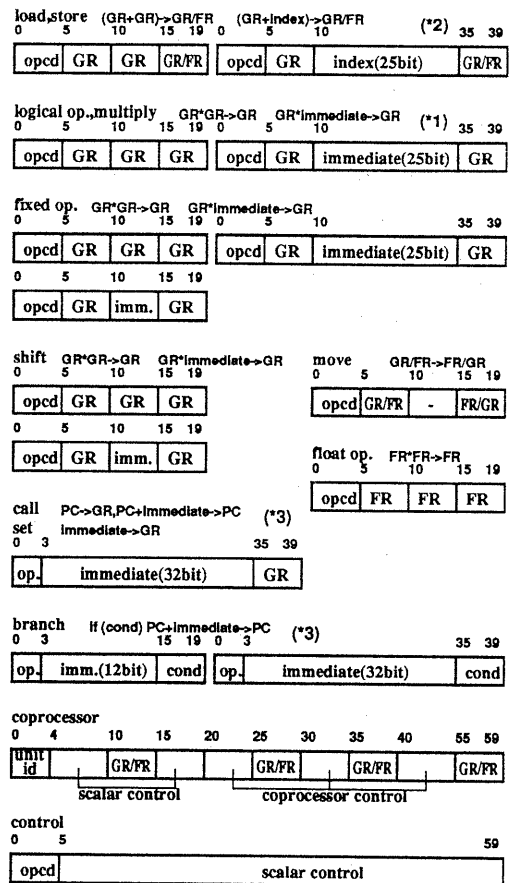
長形式命令語方式の採用

複数の操作を並列に実行する代表的手法に、スーパースカラ方式と長形式命令語方式がある。異なる点は、前者の場合、ハードウェアが操作間のデータ依存関係を調査し並列実行可能な操作を特定するために、相当のハードウェア量を必要とするのに対し、後者の場合、同一命令語中の操作のスケジューリングを全面的にコンパイラに任せ、ハードウェアには同一命令語中の全操作を必ず同時に実行開始することだけが要求されることから、ハードウェアを軽量化できることである。このような理由から、VPP500 では長形式命令語方式を採用した。さらに、ベクトル操作を含めた全命令語を固定長とすること、および、命令サイズの増大を抑えることを目的として、64 ビット長の命令語中に 1 個のベクトル操作または 1～3 個のスカラ操作を記述する独自の命令語形式を採用した。おおまかに最大 2 個の固定小数点演算操作、最大 1 個の分岐操作、最大 1 個の固定小数点乗算操作、最大 2 個の浮動小数点演算操作、最大 1 個の主記憶参照操作を組み合わせて 1 命令語中に記述することが可能である。

命令語および操作の形式

0 3 4	23 24	43 44	63
2	load,store	float add/sub/cnv	float mul/div
3	fixed op.,shift	float add/sub/cnv	float mul/div
4	fixed op.,shift	load,store,move	float add/sub/mul/div
5	fixed op.,shift	fixed op.	float add/sub/mul/div
6	fixed op.,shift	load,store,move	branch
7	fixed op.,shift	fixed op.	branch
8	load,store,move	fixed op.	
9	fixed op.,shift	fixed op.,load,store,mul	
A	float add/sub/mul/div	fixed op.,load,store,mul	
C	load,store,move,mul	call,branch,set	
D	fixed op.,shift	call,branch,set	
E	float add/sub/mul/div	call,branch,set	
F	coprocessor (vector etc.)		
B	control		

図 2: 命令語の形式



GR: General register FR: Floating-point register

図 3: 操作の形式

図 2 に命令語の形式、図 3 に操作の形式を示す。命令語の先頭 4 ビットが命令語の形式を規定する。この値により、残る 60 ビットの領域が、20+20+20 ビット長操作、20+40 ビット長操作、60 ビット長操作のいずれであるか、また、各操作がどのグループに属するかが決まる。グループおよび各操作の先頭 3～5 ビットが示すオペコードの組により、操作が一意にデコードされる。20 ビット長操作は 3 オペランド形式を基本とし、第 2 オペランドとして 5 ビットのレジスタ番号または即値を指定可能である。40 ビット長操作はオペランドとして 25 ビットまたは 32 ビットの即値を指定可能である。操作毎に、出現頻度の高い即値領域ビット長が異なることを利用し、20 ビット長操作と 40 ビット長操作の両方にオペコードを用意する、または、いずれか一方に制限することにより、5 ビットのオペコードを効率よく使用している。例えば図 3 の (*1) に示すように、乗算および論理演算に使用できる即値は 25 ビット即値だけである。これは一般的に、2 の巾乗倍を除く乗算および論理演算は、出現頻度が極めて小さく、かつ、5 ビット即値に収まる演算よりも 25 ビット即値を必要とする演算の出現頻度が大き

いに基づいている。(※2)に示すように、ロード/ストア操作についてもインデックス指定には25ビット即値のみ使用可能である。(※3)に示す条件分岐操作およびcall操作では、PC相対アドレスにより分岐先アドレスを指定する。

2.3 同期操作

VPP500では、乗算を除く固定小数点演算操作および分岐操作を同期操作と呼ぶ。本節では簡単のために同期操作についてのみ説明し、非同期操作を含む正確な規定については2.4において説明する。ハードウェアは、同一命令語に含まれる全ての同期操作の入力オペランドを同一命令語中の操作が更新する前に読み出す。即ち、同一命令語中において、複数の同期操作の入力オペランドとして同一レジスタを指定することができ、また、入力オペランドとして指定したレジスタを実行結果の格納先レジスタとして指定することができる。ただし、同一命令語中において、複数の同期操作の実行結果の格納先として同一レジスタを指定した場合、どの操作の実行結果が格納されるかは予測できない。

固定小数点演算操作

1命令語中に2個の固定小数点演算操作および1個の分岐操作を記述することができる。例えば、操作"add grX,0,grY"と操作"add grY,0,grX"を1命令語中に記述することにより、汎用レジスタXとYの内容を1命令で交換することができる。また、文字列処理の際に必要なNULL検出処理のために、レジスタ内容をバイト毎に比較し少なくとも1バイトが一致すれば条件コードが真になる操作を用意しており、1命令語中に本操作を2個記述することにより一度に8文字分の処理を行うことができる。2個の固定小数点演算器は、7ビットの条件コードレジスタを各々1組有しており、更新結果は次命令の条件分岐操作に使用することができる。

分岐操作

条件分岐操作では、常に真である1ビット、各々7ビットからなる2組の固定小数点条件コードレジスタ、2つの零フラグの論理和を示す1ビット、後述する7ビットの浮動小数点条件コードレジスタの合計23ビットの中から1ビットを指定し、そのまま用いるかまたは反転して用いる。例えば、比較操作"sub grX1,grY1,gr0"と"sub grX2,grY2,gr0"を1命令語中に記述し、2つの零フラグの論理和を用いることにより、2組の汎用レジスタX1,X2とY1,Y2を一度に比較することができる。条件分岐操作およびcall操作において指定する分岐先アドレスは、全てPC相対アドレスである。この規定は、3.2において述べるように、分岐ペナルティの削減のために大きな効果がある。

2.4 非同期操作

VPP500では、実行に複数サイクルを要することが予想される、固定小数点乗算操作、浮動小数点演算操作、主記憶参照操作およびベクトル操作を非同期操作と呼び、先行する非同期操作の実行終了を待たずに後続命令を毎サイクル発行可能とする非同期実行機構を規定している。非同期操作は、操作のキューイングまたは入力オペランドの読み出しを行う同期実行部分と操作実行結果の格納を行う非同期実行部分に分けて実行される。長形式命令語方式による操作の並列実行の効果を最大限に発揮す

るために、アーキテクチャとしては、2.3において述べた同期操作および同期実行部分が1サイクルで実行されることを推奨する。ハードウェアは、同一命令語に含まれる全ての同期操作および非同期操作の入力オペランド(条件分岐操作の場合には条件コードを含む)を同一命令語中の操作または後続命令語中の操作が更新する前に読み出す。一方、同一命令語に含まれる複数の非同期実行部分は、互いに独立に動作してよく、ハードウェアは非同期実行部分を並列に処理することができる。即ち、同一命令語中において、複数の同期操作または非同期操作の入力オペランドとして同一レジスタを指定することができ、また、入力オペランドとして指定したレジスタを実行結果の格納先レジスタとして指定することができる。ただし、同一命令語中において、複数の同期操作または非同期操作の実行結果の格納先として同一レジスタを指定した場合、どの操作の実行結果が格納されるかは予測できない。

互いに異なる命令に含まれる操作の間にデータ依存関係が存在する場合には、ハードウェアが操作の実行順序を保証する。

以上の機構を装備し、かつ、コンパイラが、先行操作の非同期実行部分が終了するまで先行操作の実行結果を参照する新たな後続操作を発行しないように、操作のスケジューリングを行うことにより、非同期操作の見かけの実行サイクル数を同期実行部分に要するサイクル数のみ、即ち1とすることができ、非同期操作を含む命令列を毎サイクル実行開始することが可能となる。なお、第4章においてスケジューリングの具体例を示す。

例外検出時の基本動作

ところで、先行操作の完了を待たずに次々と操作を発行するためには、例外発生時の割込み処理に関して特別な考慮が必要となる。VPP500では、命令列の実行中に例外を検出した場合、全ての非同期操作の実行終了を待ち合わせた後に割込みを発生し、1個または複数個の例外を同時に報告する。この際、例外を検出した操作との間にデータ依存関係があるために実行できなかった操作は未実行操作として報告する。

まず、例外の要因が同期操作または非同期操作の同期実行部分のみに存在する場合について説明する。この時点で検出される例外は全て、未定義命令や特権例外など、アドレスの通知のみを必要とし、オペランドの通知を必要としないものである。また、OPC(Old PC:更新前のPCの値)によりその操作のアドレスを特定することができる。従って、ハードウェアはソフトウェアに対し、例外種別およびOPCだけを通知する。

次に、例外の要因が非同期操作の非同期実行部分に存在する場合について説明する。この時点では、OPCの値が先に進んでいる可能性があり、OPCの値により操作を特定することができない。また一般に、例外を検出した操作のアドレスだけでなくオペランドの通知を必要とする。例えば、ストア操作実行時にページフォルトを検出した場合、ページインを行った後ストア操作を再実行するために、ストアデータおよびストア先アドレスの通知を必要とする。もし、後続操作によるレジスタへの上書きを止める機構を設ければ、オペランドをレジスタ中に保持することができるため、操作アドレスから操作を取り出し、レジスタ番号を元に必要なオペランドを得ることができる。しかし、後続操作の実行を止める機構は、ハードウェアを複雑化するばかりでなく、後続操作の実行が遅れる要因となる。一方、オペランドをレジスタから取り出し、他の手段により通知することができ

ば、後続操作によりレジスタが上書きされても構わない。VPP500では、ハードウェアの軽量化のために主に後者の方法を採用しており、例外を検出したまたは未実行となった操作およびオペランドの多くを前述の制御レジスタにより通知している。通知内容の詳細については、個々に後述する。

固定小数点乗算操作

固定小数点乗算操作は条件コードを更新せず、また演算時に例外を検出することもない。本規定により、条件コードに関する操作間の依存関係を無くし、操作の並列実行可能性を高めるとともに、非同期実行部分进行处理するハードウェアの軽量化を図っている。固定小数点乗算操作は、同期実行部分において汎用レジスタから入力オペランドの読み出しを行い、非同期実行部分において汎用レジスタに対する乗算結果の格納を行う。

浮動小数点演算操作

浮動小数点演算操作に関しては、7ビットの浮動小数点条件コードレジスタを1組規定しており、条件分岐に使用することができる。条件コードレジスタは、比較操作以外の演算操作が更新することはできない。本規定により、固定小数点乗算操作と同様、条件コードに関する操作間の依存関係を最小限とし、操作の並列実行可能性を高めている。また、浮動小数点演算は、一部の機能をソフトウェアがシミュレートすることにより、IEEE754の単精度/倍精度浮動小数点演算に準拠しており、ハードウェアの軽量化を図っている。例えばハードウェアは、介入要桁あふれ例外、介入要下位桁あふれ例外、無効操作例外、零除算例外および不正確例外を検出する。このうち介入要桁あふれ/下位桁あふれ例外はマスクすることができない。ハードウェアが浮動小数点レジスタに格納した中間結果を使用して、ソフトウェアがIEEE754準拠の桁あふれ/下位桁あふれ例外をシミュレートしている。浮動小数点演算操作は、同期実行部分において制御レジスタ空間内のFQ(Floating-point operation queue)と呼ぶキューイング機構にエンキューされる。次に非同期実行部分において、ハードウェアがレジスタ依存関係に基づき浮動小数点レジスタの読み出し、演算、演算結果の格納を行う。アーキテクチャは、レジスタ依存関係が無い場合の演算順序変更を許しており、ハードウェアはFQにエンキューされた操作を任意の順序で実行することができる。

浮動小数点演算例外検出時の動作

前述の例外を検出した際には、操作のオペコードおよびオペランドレジスタ番号が、FQのエントリ毎に設けた例外表示領域に表示される。例外を検出した先行操作との間にレジスタ依存関係があるために実行できない後続の浮動小数点演算操作は、同様にFQ内に未実行操作として表示される。ソフトウェアに対しては、例外処理を行った後、FQ内において未実行となった操作をFQの先頭から詰め直すことが要求される。ハードウェアは再構成されたFQの内容に基づき浮動小数点演算を再開する。ところで前に述べた通り、オペランドを表示する際には、レジスタ番号ではなくレジスタ内容を表示するべきである。しかし、各々64ビットの浮動小数点数を多数格納するためには相当のハードウェア量を必要とする。ハードウェア量を抑えるためには、むしろ後続操作の実行を停止させる方法が有効であるため、浮動小数点演算例外については、レジスタ番号を表示する規定としている。

主記憶参照操作

主記憶参照操作に関しては、1命令語中に最大2個の

固定小数点演算操作または最大2個の浮動小数点演算操作を記述できることに対応して、主記憶の連続アドレスから、連続する2本の汎用レジスタへの8バイトロード操作、同様に連続する2本の浮動小数点レジスタへの16バイトロード操作を規定している。主記憶参照操作は、同期実行部分において制御レジスタ空間内のAQ(Access operation queue)と呼ぶキューイング機構にエンキューされる。次に非同期実行部分において、ハードウェアがアドレス変換を行い、レジスタ間および主記憶アドレス間の依存関係に基づき主記憶参照を行う。浮動小数点演算操作同様、アーキテクチャは、レジスタ間および主記憶アドレス間に依存関係が無い場合の主記憶参照順序の変更を許しており、例えば、先行するロード操作がキャッシュミスした場合でも、後続のロード操作がキャッシュヒットした場合には、後続のロードデータを先に使用して演算を続行することができる。即ちソフトウェアによる主記憶からキャッシュへのプリフェッチ[3]を実現可能としている。命令語列の空き部分に、プリフェッチのためのロード操作を埋め込むことにより、命令語数を増やすことなくプリフェッチを行うことができる。

主記憶参照例外検出時の動作

ページフォルト等の例外を検出すると、1) ロード操作の場合、主記憶仮想アドレス、データ長、ロードデータ格納先レジスタ番号; 2) ストア操作の場合、主記憶仮想アドレス、データ長、ストアデータ; がAQのエントリ毎に設けた例外表示領域に表示される。例外を検出した先行操作との間にデータ依存関係があるために実行できない後続の主記憶参照操作は、同様にAQ内に未実行操作として表示される。ソフトウェアに対しては、ページイン等の処理を行った後、例外を検出した操作および未実行操作を再実行することが要求される。

ベクトル操作

ベクトル操作は、ベクトルレジスタ以外に、汎用レジスタまたは浮動小数点レジスタをオペランドとする場合がある。同期実行部分において汎用レジスタまたは浮動小数点レジスタを読み出した後、ベクトル操作は、非同期実行部分を司るベクトルユニットに対してエンキューされる。ベクトル操作において指定した汎用レジスタまたは浮動小数点レジスタに関し、先行する操作との間にレジスタ依存関係が存在する場合には、ハードウェアがベクトル操作の発行を待ち合わせる。同様に、先行するベクトル操作が実行結果を汎用レジスタまたは浮動小数点レジスタに格納し、後続操作がこれを参照する場合、ハードウェアが後続操作の発行を待ち合わせる。

3 インプリメントの特徴

3.1 概要

図4にスカラプロセッサの構成を示す。スカラプロセッサはIU(Instruction control unit), MU(Fixed-point multiply unit), FU(Floating-point unit), AU(Access control unit)と呼ぶ各々独立に動作する4つのユニットから構成される。IU, MUおよびFUの実現には、ゲート遅延時間60ps(Pico second)のGaAs(Gallium arsenide)素子によるゲート数25000のLSIを5個、キャッシュを含むAUの実現には、ゲート遅延時間70ps, RAMアクセス時間1600ps, RAM容量64KbitのECL(Emitter coupled logic)素子によるLSIを13個使用している。同期操作および同期実行部分をIUが処理し、非同期実行部

分をその他のユニットが処理することにより、固定小数点演算操作、固定小数点乗算操作、浮動小数点演算操作、主記憶参照操作およびベクトル操作を並列に実行する。

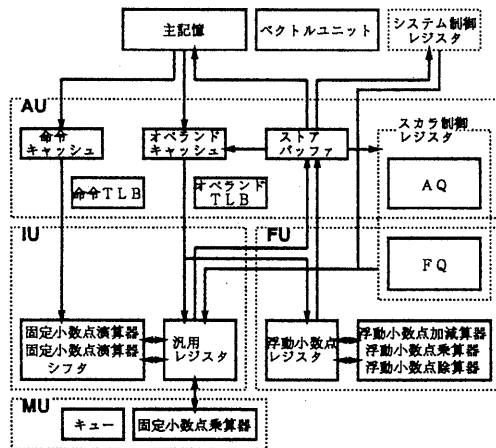


図 4: スカラプロセッサの構成

各ユニット内およびユニット間では、先行操作と後続操作の間にデータ依存関係が存在する場合にのみインタロックが生じる。例えば先行操作が主記憶からのロード操作、後続操作が浮動小数点演算操作である場合を考える。一般にロード操作の実行には、キャッシュミス等により多くのサイクル数を要する場合がある。ロード操作の非同期実行部分は AU、浮動小数点演算操作の非同期実行部分は FU が実行する。後続の浮動小数点演算操作がオペランドとしてロード結果を使用する場合には、FU が AU の処理完了を待ち合わせる。一方、ロード結果を使用しない場合には、FU は AU の処理完了を待つことなく浮動小数点演算を開始する。即ち後者の場合、AU においてキャッシュミスが検出されロード操作の実行が遅延した場合においても、このために FU における浮動小数点演算が遅延することはない。表 1 に、主なパイプライン・インタロックによるペナルティ・サイクル数を示す。

操作 A の結果を操作 B が使用する場合 A → B と記す	ペナルティ サイクル数
固定小数点演算 → 条件分岐	≒0
浮動小数点演算 → 条件分岐	≒3
固定小数点演算 → 固定小数点演算	0
固定小数点乗算 → 固定小数点演算	5~7データ依存
浮動小数点加減乗算 → 浮動小数点演算	3
浮動小数点除算 → 浮動小数点演算	4~9データ依存
ロード → 固定小数点/浮動小数点演算	2

表 1: 主なペナルティ・サイクル数

3.2 命令パイプライン

命令パイプラインは 3 ステージからなり、3.5 において述べる命令フェッチパイプラインと密接な関係がある。

図 5 に示すように命令パイプラインは、操作のデコードおよび汎用レジスタの読み出しを行う D ステージ、演算を行う E ステージ、演算結果を汎用レジスタに格納し、条件コードおよび PC を更新する W ステージから構成される。

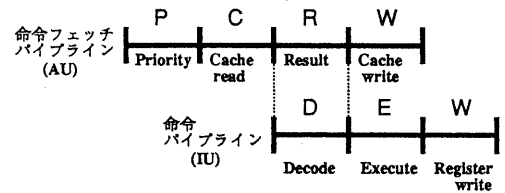


図 5: 命令パイプライン

命令フェッチについては、以下の方式により、分岐操作のフェッチと同時に分岐先アドレス計算を行うこと、および、分岐操作のデコードと同時に分岐方向を決定することが可能となった。分岐が連続し、プリフェッチが間に合わない場合を除き、分岐時のペナルティは 0 である。

- 1) 各々 2 命令分の現命令バッファ、次命令バッファおよび分岐先命令バッファを装備し、128 ビット幅のデータバスを命令キャッシュとの間に設けることにより、連続する 2 命令を毎サイクルプリフェッチする。
- 2) 2.3 において述べたように、条件分岐操作および call 操作における分岐先アドレスが常に PC 相対アドレスであることを利用して、PC と PC 相対アドレスを常に加算しておき、分岐操作のフェッチと同時に分岐先アドレスを得る。
- 3) 命令パイプラインを 3 ステージと短くし、分岐操作のデコード前に直前の固定小数点演算操作が更新する条件コードを参照可能とする。

一方、サブルーチンから戻る際に使用する jmp 操作 (レジスタ間接分岐) の場合、分岐先アドレスはレジスタ中に保持されているため、上記方式だけでは jmp 操作のフェッチと同時にアドレス計算を行うことができない。jmp 操作については、以下の方式により高速化を行った。まず、VPP500 ではサブルーチン呼び出し時に call 操作を使用し汎用レジスタ GR3 に戻りアドレスを格納するとした。次に、GR3 に格納した値をハードウェアが jmp 操作の分岐先予測アドレスとして内部バッファに保持し、GR3 を使用する jmp 操作を実行するまでに GR3 に対する書き込みが行われなければ、本内部バッファの内容を分岐先アドレスとして使用するとした。本機構により、条件分岐操作と同様、jmp 操作についても分岐先アドレスを予め知ることができる。

3.3 固定小数点演算機構

IU は、33 本の 32 ビット汎用レジスタ、2 個の ALU および 1 個のパレルシフトを有する。IU はパイプライン化されており、「ALU 操作またはシフト操作」と「ALU 操作」の 2 つの操作を毎サイクル同時に実行することができる。

MU は、最大 1 個の乗算操作の非同期実行を可能とするキューイング機構および乗算器からなる。32 ビット × 32 ビットの乗算器は、32 ビットまたは 64 ビットの演算

結果を生成することができる。なお、MUはパイプライン化されていない。即ち、先行する乗算操作が終了するまで次の乗算操作の実行が待たされる。乗算結果を次命令の入力オペランドとして使用する場合のペナルティ・サイクル数は、乗数値の大小に依存しており、5～7である。

3.4 浮動小数点演算機構

FUは、32本の64ビット浮動小数点レジスタ、最大12個の演算操作の非同期実行を可能とするFQ、1個の加減算器、1個の乗算器および1個の除算器からなる。加減算器では加減算操作の他、比較操作、浮動小数点-浮動小数点型変換操作、浮動小数点-固定小数点型変換操作を行う。除算器を除く加減乗算器がパイプライン化されており、「加減算」と「乗算」の2つの操作を毎サイクル同時に実行開始し、一方で演算結果を生成することができる。「除算」については、先行する除算操作が終了するまで次の除算操作の実行が待たされる。なお、操作間にレジスタ依存関係がない場合、加減乗算は除算を追い越して演算結果を格納することができる。

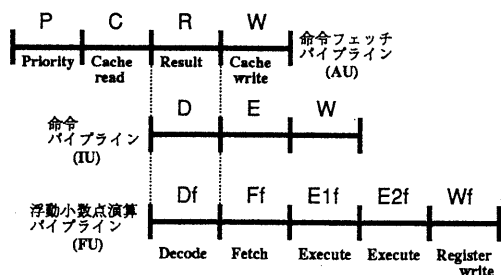


図 6: 浮動小数点演算パイプライン

図 6 に示すように、浮動小数点演算パイプラインは、操作をデコードする Df ステージ、浮動小数点レジスタの干渉検査および読み出しを行う Ff ステージ、演算を行う E1f および E2f ステージ、演算結果を浮動小数点レジスタに格納する Wf ステージから構成される。浮動小数点演算操作は、命令パイプラインの D ステージにおいてデコードされると同時に、FQ に空きがあれば FQ にエンキューされる。FQ は、1 命令語中に記述される加減乗算操作と乗除算操作の組を 6 組までキューイングすることが可能である。先行操作の実行結果を待ち合わせる必要がない場合には、同一サイクルの Df ステージにおいて操作のデコードが行われる。除算はパイプライン化されておらず、E1f および E2f ステージの代わりに複数の Ef ステージから構成される。但し、各 Ef ステージを半サイクル長とすることにより除算を高速化している。加減乗算結果を次命令の入力オペランドとして使用する場合のペナルティ・サイクル数は 3、除算結果を使用する場合は、被除数値の大小に依存しており、単精度演算の場合 4～6、倍精度演算の場合 4～9 である。但し、レジスタ依存関係が FQ 内の浮動小数点演算間に閉じている場合には、浮動小数点演算パイプラインのみがインタロックし、命令パイプラインがインタロックすることはない。即ち、FQ が満杯になるか、または、浮動小数点演算以外の操作との間にレジスタ依存関係が生じた場合を除き、命令パイプラインは動き続けることができる。

3.5 主記憶参照機構

AU は、最大 2 個の主記憶参照操作の非同期実行を可能とする AQ、4 個の 8 バイトストアバッファ、TLB およびキャッシュからなる。AU はパイプライン化されており、主記憶参照操作を毎サイクル実行開始することができる。また、データキャッシュから汎用レジスタおよび浮動小数点レジスタへは、各々レジスタ 2 本分の幅を有するデータバスを装備しており、前に述べた、2 本の汎用レジスタへの 8 バイトロード操作、および、2 本の浮動小数点レジスタへの 16 バイトロード操作を他のロード操作と同じサイクル数で実行することができる。さらに、主記憶参照操作の間にデータ依存関係が無い場合、後続のロード操作が先行するロード操作を追い越すことができる機構を装備しており、2.4 において述べたソフトウェアプリフェッチを可能とした。

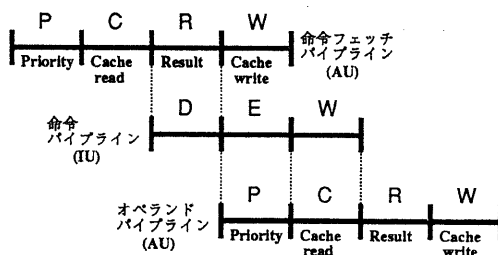


図 7: 命令フェッチ/オペランドパイプライン

図 7 に示すように、命令フェッチパイプラインおよびオペランドパイプラインは、キャッシュ参照権を獲得する P ステージ、TLB およびキャッシュを参照する C ステージ、参照結果を得る R ステージ、キャッシュに書き込みを行う W ステージから構成される。P ステージでは、AU および IU からのキャッシュアクセス要求に対する優先順序付けを行う。C ステージでは、TLB を参照して仮想アドレスから実アドレスへの変換を行った後、キャッシュに登録されているかどうかを検査しキャッシュの読み出しを行う。R ステージでは、キャッシュヒットまたはキャッシュミスの報告、例外の報告、キャッシュへの書き込みが必要な場合の優先順序付けを行う。W ステージでは、必要な場合に於いて TLB またはキャッシュへの書き込みを行う。キャッシュにヒットした場合、ロードデータを次命令の入力オペランドとして使用する場合のペナルティ・サイクル数は 2 である。

命令 TLB およびオペランド TLB にはダイレクトマップ方式を採用している。エントリ数は各々 256、ページサイズは 32K バイトである。65536 個のプロセス各々に対して 4G バイトの仮想アドレス空間を提供する動的アドレス変換機構を装備しており、多重仮想アドレス空間を実現している。TLB の各エントリは、エントリ有効ビット、プロセス識別子、論理アドレス上位 9 ビット、実アドレス上位 17 ビット、ページ制御情報 (Write-protect, Read-protect, Execute-protect, Common-page, Modified-page) を保持する。

命令キャッシュおよびオペランドキャッシュにはライトスルー方式およびダイレクトマップ方式による物理キャッシュを採用している。容量は各々 32K バイト、ラインサイズは 64 バイト、キャッシュミス時のペナルティは 17 サイクルである。

4 性能

スカラプロセッサのサイクルタイムは10nsである。最大2個の浮動小数点演算操作を毎サイクル発行できることから、浮動小数点演算ピーク性能は200MFLOPS(Million floating-point operations per second)である。表2に、FORTRAN コンパイラを使用し、Livermore14 ループを走行した結果を示す。

Loop	演算数	MFLOPS	Loop	演算数	MFLOPS
1	2000	113.718	8	1440	52.574
2	2000	73.723	9	1700	49.312
3	2000	73.748	10	900	22.827
4	1020	34.436	11	1000	22.649
5	2000	23.584	12	1000	40.804
6	2000	23.243	13	896	8.370
7	1920	99.237	14	1650	16.357

表 2: Livermore14 ループの走行結果

Loop1 に関しコンパイラは、4重ループアンローリングおよびソフトウェアパイプラインを行っている。非同期操作であるロード(ld)および浮動小数点加算/乗算(fadd/fmul)に関し、ロード結果は3サイクル後以降、演算結果は4サイクル後以降に使用するようにスケジューリングを行った結果、図8に示す操作列が得られる。固定小数点加算(add)により、各配列に対するロード/ストアのベースアドレスを4要素分ずつ更新している。各操作について、“>”の左辺は入力オペランド、右辺は格納先を示す。矢印はデータ依存関係を示す。この操作列を折り畳んだ結果、図9に示す命令語列が得られる。左端の数字は図2に示した命令語の形式である。この例では、38個の操作が「64ビット長命令*18個=144バイト」に収まっており、仮にスーパースカラ方式とした場合の「32ビット長操作*38個=152バイト」よりも命令サイズが小さくなっている。また、1命令語当りの平均操作数は2.1個であり、命令語形式および並列実行機構が有効に使用されていると言える。

ただし、この命令語列には改良の余地がある。例えば、[*1]の各固定小数点操作を空き領域である[**1]へ移動し、[*2]のロード操作を[**2]へ移動し、さらに[*3]の浮動小数点操作と分岐操作を1命令にまとめることにより、ループ当りの命令語数は16に減少し、1命令語当りの平均操作数は2.4となる。このような緻密なスケジューリングは今後の課題である。

以上に述べたように、操作の並列実行により性能向上を図るためには、ハードウェアとコンパイラの緊密な連携が必要である。特にコンパイラの果たす役割は大きく、ハードウェアの性能を十分に引き出すためには、コンパイラに対して、ハードウェアの並列実行機構を明示することが重要である。本アーキテクチャが規定する長形式命令語および非同期実行機構は、ハードウェアを高速化するための機構であると同時に、まさに、コンパイラに対して並列実行機構を明示する機構である。

5 おわりに

本論文では、VPP500 スカラプロセッサに関し、まず、長形式命令語および非同期実行機構を大きな特徴とするアーキテクチャについて述べた。次に、本アーキテクチャ

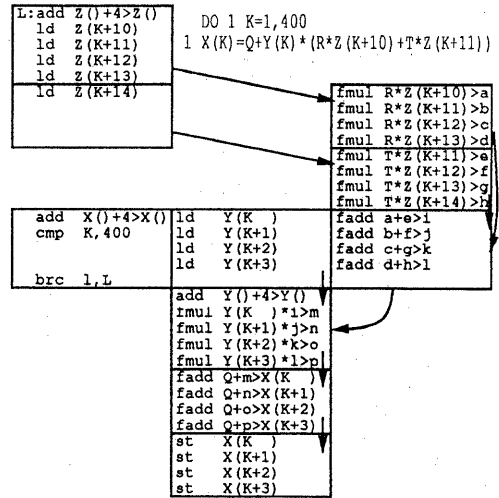


図 8: Loop1 の操作列

7	L: add Z()+4>Z()	add Y()+4>Y() [*1]
2	ld Z(K+10)	fmul Y(K) *i>m
2	ld Z(K+11)	fmul Y(K+1) *j>n
2	ld Z(K+12)	fmul Y(K+2) *k>o
2	ld Z(K+13)	fmul Y(K+3) *l>p
2	ld Z(K+14)	fadd Q+m>X(K) fmul R*Z(K+10)>a
3	**1	fadd Q+n>X(K+1) fmul R*Z(K+11)>b
3	**1	fadd Q+o>X(K+2) fmul R*Z(K+12)>c
2	**2	fadd Q+p>X(K+3) fmul R*Z(K+13)>d
2	st X(K)	fmul T*Z(K+11)>e
2	st X(K+1)	fmul T*Z(K+12)>f
2	st X(K+2)	fmul T*Z(K+13)>g
2	st X(K+3)	fmul T*Z(K+14)>h
4	add X()+4>X()	ld Y(K) fadd a+e>i
4	cmp K,400	ld Y(K+1) fadd b+f>j
4		ld Y(K+2) fadd c+g>k
4		ld Y(K+3) [*2] fadd d+h>l [*3]
7	brc L,L[*3]	

図 9: Loop1 の命令語列

を最大限に利用したインプリメントについて述べ、最後に、アーキテクチャおよびインプリメントが、コンパイラ技術とうまくかみ合っており、極めて有効であることを示した。今後は、分岐操作を越えて操作を移動する広域スケジューリング機能を付加したコンパイラを用い、分岐操作を多く含む命令列に関して、アーキテクチャおよびインプリメントの評価を定量的に行う予定である。

参考文献

- [1] Kenichi Miura, Moriyuki Takamura, Yoshinori Sakamoto, Shin Okada, "Overview of the Fujitsu VPP500 Supercomputer", Digest of Papers, COMPCON Spring 93, 1993.
- [2] N. Uchida, M. Hirai, M. Yoshida, and K. Hotta, "Fujitsu VP2000 Series", Digest of Papers, COMPCON Spring 90, pp.4-11, Feb. 1990.
- [3] David Callahan, Ken Kennedy, Allan Porterfield, "Software Prefetching", Proceedings of ASPLOS-IV, Apr. 1991.