計算機アーキテクチャ 87－1
（１９９１. 3. １１）

OHMEGA：数値計算用スーパースカラ・マイクロプロセッサのアーキテクチャ

- ハードウエア構成とパイプライン構造 -

中島雅逸　中野拓　中倉康浩　吉田忠弘
後井良之　中居祐二　瀬川礼二　岸田武　廉田浩

松下電器産業(株) 半導体研究センター

数値計算分野を主たるアプリケーションとするスーパースカラ・マイクロプロセッサOHMEGAを開発した。OHMEGAは、スーパースカラ方式と呼ばれる命令レベルの並列実行方式を採用するとともに、最適化コンパイラによって静的にコード・スケジューリングされたオブジェクト・コードに対して、動的に発生するハザードをout-of-orderの命令実行を含めて実行時に解消することにより高い実行性能を実現する。

本報告では、OHMEGAのアーキテクチャ、特にそのハードウエア構成とパイプライン構造について述べる。

OHMEGA : A VLSI Superscalar Microprocessor Architecture for Numerical Applications

- Hardware Organization and Pipeline Structure -

Masaitsu Nakajima　Hiraku Nakano　Yasuhiro Nakakura　Tadahiro Yoshida
Yoshiyuki Goi　Yuuji Nakai　Reiji Segawa　Takeshi Kishida　Hiroshi Kadota

Semiconductor Research Center , Matsushita Electric Industrial Co ., Ltd.

We have developed a VLSI superscalar microprocessor for numerical applications , it's called OHMEGA processor . OHMEGA processor adopts superscalar architecture that operates instruction - level parallel execution , statically code - scheduling by compiler , dynamically hazard resolution with out - of - order execution . OHMEGA processor realizes a very high performance by taking advantage of these techniques.

This paper describes OHMEGA processor architecture , especially its hardware organization and pipeline structure .

1. はじめに

我々は、科学技術計算分野における2次元および3次元の偏微分方程式を高速に求解するシミュレーション指向型超高速並列計算機ADENA (Alternating Direction Edition Nexus Array：交互方向編集アレイ) 開発[1]の一環として、そのプロセッシング・エレメント(PE)としての使用を目的とする、外部データバス64ビットのRISCマイクロプロセッサ EPU (Elememt Processing Unit) を開発[2)-4)]してきた。現在、並列計算機ADENAは商用化に向けての実用化開発段階にあり、ハードウエアの最終調整及びシステム、アプリケーションを含むソフトウエア整備が進められている。

この開発と平行して次世代システムADENA-Hの開発が進められており、そのPEを構成する上での核となるスーパースカラ・マイクロプロセッサOHMEGAを開発した[5)-6)]。OHMEGAは、スーパースカラと呼ばれる命令レベルの並列実行方式を採用し、数値計算という限定されたアプリケーション分野において、その性能が最大限に発揮できるように設計されている。

本報告では、まず最初にスーパースカラ・プロセッサが持ついくつかの問題点を示した後、それらの問題点に対するOHMEGAアーキテクチャの基本的なアプローチを説明する。続いて、OHMEGAプロセッサのハードウエア構成、及びパイプライン構造を説明し、最後にリバモア・ループを用いた簡単な性能評価結果を示す。

2. スーパースカラ・プロセッサ

スーパースカラ方式とは、1クロックサイクル内に複数の命令を並列に実行することにより、プロセッサの性能を向上させる命令レベルの並列実行方式である。この方式を採用するスーパースカラ・プロセッサを実現する場合に考慮しなければいけない問題点として、少なくとも次にあげるような4つの問題が存在する。

2-1. データ依存関係

データ依存関係は、命令間依存関係のうちメモリあるいはレジスタに格納されたデータに関する依存関係であり、さらに3つに分類することができる。フロー依存(RAW:Read After Write)、逆依存(WAR;Write After Read)、出力依存(WAW;Write After Write)である。これらのデータ依存関係を解決するアルゴリズムとして、もっとも簡単なインターロック制御アルゴリズムをはじめ、コントロール・データ社のCDC6600[8)]で用いられたThorntonのスコアボードアルゴリズム、IBM社のIBM System 360 / model 91で用いられたTomasuloのアルゴリズム[7)9)]等が提案されている。この3つのアルゴリズムのうち、Thorntonのスコアボードアルゴリズムは、データ依存関係の無い命令のout-of-order実行を実現するものであり、Tomasuloのアルゴリズムは、フロー依存以外の命令のout-of-order実行を実現するものである。

2-2. 制御依存関係

制御依存関係とは、命令間依存関係のうち分岐命令に起因して発生するもので、分岐命令に対する命令フェッチアドレスが確定するまでは、分岐命令の後続命令のフェッチ及び、実行が妨げられるというものである。特に条件分岐命令に関しては、条件が確定するまで分岐の成立不成立が判断できないため、このことににとるオーバーヘッドが性能を低下させる大きな要因となる。これを解決する制御方式として、少なくとも以下の2方式、lazy executionと呼ばれる方式と、eager executionと呼ばれる方式がある。また、制御方式の選択と同様に、分岐命令の高速実行、確率の高い分岐予測及び、分岐予測が外れた時のパイプライン復元処理等をいかに実現するかということが問題となる。

2-3. コード・スケジューリング

コード・スケジューリングには、最適化コンパイラによってコンパイル時に命令レベルの並列性を検出する静的コード・スケジューリングと、プロセッサ内部のハードウエアによって実行時に命令レベルの並列性を検出する動的コード・スケジューリングとがある。静的コード・スケジューリングにおいては、命令間依存関係(データ依存、制御依存)の存在によって完全に並列化されたコードの生成が困難であること、及び動的に発生するハーザードに対して対処が困難であるという問題点があり、動的コード・スケジューリングにおいては、これを実現するためのハードウエアが非常に複雑かつ大きくなるという問題点がある。

2-4. バス・ボトルネック

バス・ボトルネックはスーパースカラ・プロセッサに限らずすべてのタイプのコンピュータに共通の問題点である。しかしながらスーパースカラ・プロセッサにおいては、命令の並列実行により内部の演算処理能力が格段に増大しており、それに見合うだけのデータの供給能力をいかに確保するかということがさらに大きな問題となる。

3. OHMEGAの設計方針

2章で示した問題点に対して、スーパースカラ・プロセッサOHMEGAは以下に示すようなアプローチをとることにより、それらを効率的に解決する。また、OHMEGAの設計においては、そのアプリケー

ションを数値計算分野に限定することにより、さらに数値計算専用プロサッサとしての最適化を図っている。

① 静的コード・スケジューリング
（均一長、ロー・ラテンシー パイプライン構造）
（命令レベルの並列性検出の容易さ）
② 動的ハザード解消
（out-of-order の命令実行）
③ インターロックによる制御依存解消
（ノンベナルティ分岐の実現）
④ 大容量、高スループット データCACHE
（8kbyte 2 way set associative, Pipelined）
⑤ ハーバード・スタイル・バス
（128bit データバス、32bit命令バス）
⑥ 大容量、6ポートレジスタファイル
（32b x 32w integer、64b x 64w floating）
⑦ PLD（Pair LoaD）命令の実装
（64b x 2 のデータを1サイクルでロード）

4．ハード・ウエア

OHMEGA は、以下に示すような主要なブロックより構成される。図1に、OHMEGA の概略ブロック構成図を示す。命令バスとデータバスが外部で分離されたハーバード・アーキテクチャーをとっている。外部データバスは、アドレス32bit、データ128bitであり、倍精度浮動小数点データ（64bit）を外部データ・メモリから、2個同時にフェッチすることができる。また、128bit の外部データバスを効率的に使用するために、PLD（Pair LoaD）命令を備える。PLD（Pair LoaD）命令により、連続する2個の倍精度浮動小数点データを、1命令で連続する2個のレジスタにロードすることが可能である。

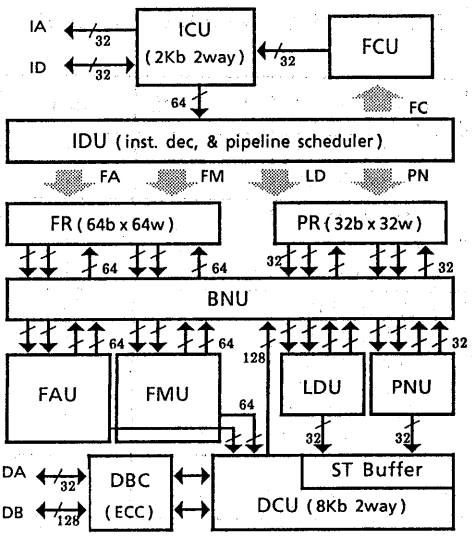

図1 内部ブロック構成図

（1）ICU（Instruction Cache Unit）

2ウエイ・セット・アソシアティブ方式LRU 制御の命令キャッシュ。ラインサイズは16 byte、データ容量は2Kbyteであり、毎サイクルごとに、2命令（64bit）を供給することができる。また、外部命令メモリインタフェス（アドレス 32bit、データ 32bit）を内蔵し、命令キャッシュ・ミス時には、外部より命令をフェッチする。

（2）FCU（Flow Control Unit）

命令アドレス生成用の32bit 加算器を内蔵し、分岐アドレスの生成等、フロー制御を行う。また、制御レジスタ、32bit x 32word のスタックメモリを内蔵する。

（3）IDU（Instruction Decode Unit）

ICUからフェッチされた2命令を同時にデコードする。命令間の依存関係の検出、演算実行ユニットに対する制御情報の生成を行い、発行可能な命令から命令の発行を行う。

（4）PR（Pointer Register）

32bit x 32word、読み出し32bit x 4ポート、書き込み32bit x 2ポートの整数データ・レジスタ。

（5）PNU（Pointer execution Unit）

32bit ALU、及びバレルシフタを内蔵し、PR データに対する算術論理演算処理及び、ST（Store）命令時のデータアドレス計算を実行する。

（6）LDU（LoaD execution Unit）

32bit ALUを内蔵し、PR データに対する算術論理演算処理及び、LD/ST（Load/Store）命令時のデータアドレス計算を実行する。

（7）FR（Floating point Register）

64bit x 64word、読み出し64bit x 4ポート、書き込み64bit（128bit）x 2ポートの浮動小数点データ・レジスタ。

（8）FMU（Floating Multiply Unit）

2段パイプライン構成の倍精度浮動小数点乗除算器。倍精度浮動小数点データに対して乗算、及び除算を実行する。

（9）FAU（Floating Alithmetic Unit）

2段パイプライン構成の倍精度浮動小数点加減算器。倍精度浮動小数点データに対して加算、減算、及び形変換命令を実行する。

（10）BNU（Bypass Network Unit）

4つの演算実行部（PNU LDU FMU FAU）は、それぞれ独立に、対応するレジスタに対して、読み出しポートを2ポート、書き込みポートを1ポートずつ持っており、これらのポートとそれぞれの演算実行部間は、BNU を介して接続されている。また、4つの演算実行部からの、任意の演算結果出力をバイパス

して、他の演算実行部にソーズデータとして供給することが可能であり、さらに、演算結果のバイパスは、演算実行ステージの第1段目からでも実行可能である。

(11) DCU (Data Cache Unit)

2ウエイ・セット・アソシアティブ方式LRU制御のデータキャッシュ。データのリプレース方式は、ライト・スルーであり、バススヌーピングの機能を持つ。ラインサイズは16byte、データ容量は8Kbyteであり、毎サイクルごとに、128bitのデータ(倍精度浮動小数点データ x 2)を供給することができる。また、2ポート構成、4エントリーのストア・バッファーを内蔵し、1サイクルでST(Store)命令を同時に2命令実行する。

(12) DBC (Data Bus Controller)

ECC(Error Checking & Correction)回路、及びSRAM/DRAMインタフェスを内蔵し、外部データバス(アドレス 32bit データ 128bit)の制御を行う。

5. パイプライン

5-1. パイプライン構造

OHMEGAは、図2に示すような5本の命令実行パイプラインを持ち、それぞれ次に示すようなような命令群を実行する。また、それぞれのパイプラインは、すべて独立に制御される。

(1) FC パイプ

分岐命令、内部制御命令、制御レジスタアクセス

(2) PN パイプ

算術論理演算命令(32bit)、シフト命令、ST命令(アドレス計算)、MVP系命令

(3) LD パイプ

算術論理演算命令(32bit)、LD/ST命令(アドレス計算)、MVP系命令

(4) FM パイプ

浮動小数点乗除算命令、MVF系命令、LD命令(レジスタ書き込み)

(5) FA パイプ

浮動小数点加減算命令、形変換命令、MVF系命令、LD命令(レジスタ書き込み)

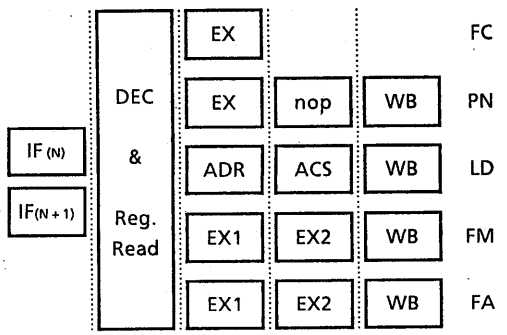

図2 内部パイプライン構成図

演算命令に関しては、基本的に上記の5本のパイプラインが独立に動作して命令を実行処理するが、ST(Store)命令の場合には、PNパイプとFMパイプの組合せ、あるいはLDパイプとFAパイプの組合せの2つのパイプラインを組み合せて命令を実行し、LD(Load)命令の場合には、LDパイプとFMパイプの組合せ、あるいはLDパイプとFAパイプの組合せによって命令を実行する。

また、浮動小数点除算命令の実行時、LD(Load)命令実行時のデータ・キャッシュミス時、及びDCU内のストア・バッファーでLD/STのアドレスコンフリクトが発生した場合、ストア・バッファー・フルの場合には、対応する実行パイプラインにパイプロック制御が行われ、命令を受け付けない状態となる。(リソース・コンフリクト)

5-2. パイプライン処理ステージ

OHMEGAは、基本的に以下に示すような5段のパイプライン処理により、命令を実行する。

(1) IF (Instruction Fetch) ステージ

FCU内のIP(Instruction Pointer)の値に基づいて、ICUから連続する2命令(32 bit x 2)を読み出す。命令アライメント機能を持たないため、ミスアライメントを起こした命令に関しては、NOP命令に置換される。

(2) DEC (instruction DECode) ステージ

ICUからフェッチされた2命令を同時にデコードし、発行可能な命令から順次、あるいは、並列に命令

命令グループ	サイクル数	Latency
分岐	1	1
内部制御	1	1
整数ALU	1	1
整数シフト	1	1
整数MV	1	1
浮動小数点加算	1	2
浮動小数点減算	1	2
浮動小数点乗算	1	2
浮動小数点除算	8	9
浮動小数点MV	1	1
形変換	1	2
Load(LD)	1	2
Store(ST)	1	-

表1 命令実行サイクル数

発行する。フェッチされた2命令は、まずどの実行パイプラインに発行すべきか判断され、その後で、命令依存関係の検出が行われる。命令依存関係の検出と同時に、レジスタ(FR、PR)からオペランドデータの読み出しが行われ、発行可能な命令であれば、対応する実行パイプラインに対して命令発行する。また、分岐先アドレスの計算も命令デコードと同時に行われる。

(3) EX1 (EXecution 1) ステージ

IDUから発行された命令は、それぞれの演算パイプラインで即座に実行される。それぞれの演算パイプラインごとに次のような処理が行われる。また、表1に命令ごとの実行サイクル数を示す。浮動少数点除算命令以外の全ての命令が、1サイクルごとに実行可能である。

(i) FCパイプ

分岐命令の実行、及び内部制御命令の実行。

(ii) PN、LDパイプ

算術論理演算の実行、あるいはLD/ST命令のデータアドレス計算。

(iii) FM、FAパイプ

浮動小数点演算の第1実行ステージ、及び先読み演算結果フラグの生成。

(4) EX2 (EXecution 2) ステージ

演算実行ステージ第2段目。このステージで、最終的に命令依存関係を解消する。

(i) PN、LDパイプ

NOP(整数演算時)、あるいはLD/ST命令実行時の、DCUアクセス。

(ii) FM、FAパイプ

浮動小数点演算の第2実行ステージ。

(5) WB (Write Back) ステージ

レジスタ(FR、PR)内のデスティネーション・レジスタに対して実行結果を格納する。それぞれの演算実行パイプラインが独立したレジスタの書き込みポートを使用できるため、WBステージに進んだ命令については、その実行結果が即座にレジスタに書き込まれる。

5-3. 命令発行制御

命令は、FCU内のIP(Instruction Pointer)によって、ICUから2命令同時にフェッチされる。ICUにおいて、キャッシュ・ミスが発生した場合は、NOP命令がフェッチされ、パイプラインを乱すことなく、先行命令の実行が進められる。

フェッチされた2命令は、IDU内のDECステージで同時にデコードされる。まず、命令グループがチェックされ、どの実行パイプに対して、フェッチされた命令を発行するかが判断される。この時点で、2命令の同時実行が可能かどうかが判断される。表2に、2命令の組合せと、そのそれぞれの組合せにおいて2命令の同時実行が可能かどうかを示す。ほとんどの場合で、同時実行が可能である。

命令発行すべき実行パイプが決まると、その実行パイプにおいてリソース・コンフリクトが発生していないかどうか、さらに、その命令と先行命令との間のデータ依存関係、あるいは制御依存関係の検出が行われる。同時に、命令の特定フィールドからソース・レジスタ番号と、デスティネーション・レジスタ番号が切り出され、このソース・レジスタ番号を用いて、各演算パイプに対応するレジスタから、BNUを介してオペランド・データが読み出される。オペランド・

(2nd Instruction)

命令グループ (1st Instruction)	BR	CTRL	PALU	PMV	PSFT	FA/S	CONV	FM/D	FMV	FLD	FST
分岐(BR)	X	X	O	O	O	O	O	O	O	O	O
内部制御(CTRL)	X	X	O	O	O	O	O	O	O	O	O
整数ALU(PALU)	O	O	O	O	O	O	O	O	O	O	O
整数MV(PMV)	O	O	O	O	O	O	O	O	O	O	O
整数シフト(PSFT)	O	O	O	O	X	O	O	O	O	O	O
浮動小数点加減算(FA/S)	O	O	O	O	O	X	X	O	O	O	O
形変換(CONV)	O	O	O	O	O	X	X	O	O	O	O
浮動小数点乗除算(FMUL)	O	O	O	O	O	O	O	X	O	O	O
浮動小数点MV(FMV)	O	O	O	O	O	O	O	O	O	O	O
浮動小数点Load(FLD)	O	O	O	O	O	O	O	O	O	X	O
浮動小数点Store(FST)	O	O	O	O	O	O	O	O	O	O	O

表2 命令並列実行　　O ・・・ 実行可能　　X ・・・ 実行不可能

データの読み出しが終了し、リソース・コンフリクト、及び依存関係が検出されなければ、その命令を、順次、あるいは同時に発行する。

発行された命令は、パイプライン処理によって実行され、リソース・コンフリクト、データ依存関係、及び制御依存関係が発生しないような理想的なオブジェクト・コードの場合には、完全に2命令ずつ並列に実行が進んでいく。また、命令発行時にはout-of-orderの命令発行を、異なる演算パイプ間ではout-of-orderの命令実行を行っているが、同一パイプに発行された命令は、すべてin-orderに実行される。

6．性能評価

GDTを用いてOHMEGAの機能設計を行い、その性能を評価した。OHMEGAの製造プロセスとして、0.8 μm CMOSプロセスを用い、高速演算器・メモリ設計技術、及びクリティカルパスの最適化により、クロック周波数40MHz、サイクルタイム25 nsec、ピーク性能80MIPS/80MFLOPSを実現する。

性能評価用のプログラムとして、以下に示すようなLivermore Fortran カーネルのカーネル1と3を用いた。それぞれのカーネル内の最内ループのみを示す。実際のカーネルでは、さらにこのループの外側にdo文が存在する。

```
      Kernel 1  Hydro Fragment
      do 1 k = 1,n
1     X(k)=Q+Y(k)*(R*ZX(k+10)+T*ZX(k+11))

      Kernel 3  Inner Product
      Q = 0.0
      do 3 k = 1,n
3     Q = Q + Z(k)*X(k)
```

コンパイルは、ハンドコンパイルによって行い、ループをアンローリングしないコンベンショナルなコンパイル手法によってコンパイルしたオブジェクト・コードと、ループを4重にアンローリングして最適化したオブジェクト・コードとを用意し、最適化コンパイラによるアンローリングの効果を調べた。アンローリングを行ったプログラムに関しては、さらに、PLD(Pair Load)命令を使用してオブジェクトの最適化を図っている。また、比較のためにEPU(ピーク性能20MIPS/20MFLOPS)の評価結果とEPUのクロック周波数を単純に40MHzに高速化した場合(H-EPU ピーク性能40MIPS/40MFLOPS)の評価結果を示し、H-EPUの性能を1としたときの相対性能も同時に示す。

Livermore Kernel	H-EPU (40MHz)	OHMEGA (hit)	OHMEGA (miss)
Kernel 1	1.5	2.4	2.0
Kernel 3	1.4	2.8	2.5

表4　ループ・アンローリングの効果

表3に、配列のベクトル長が短く、すべてのデータがデータキャッシュに格納可能な場合(キャッシュ・オール・ヒット)の性能と、配列のベクトル長が長く、データがデータキャッシュに格納しきれない場合(キャッシュ・オール・ミス)の性能を同時に示す。外部データメモリのアクセスサイクルは、H-EPUもOHMEGAもともに2サイクル(50 nsec)、命令キャッシュはオール・ヒットとする。また、表4には、表3の性能評価結果をもとに、アンローリングを行うことにより、アンローリングを行なわない場合の何倍の性能がでるかを相対値で示している。

表3より明らかなように、OHMEGAの性能は同一クロック周波数(40 MHz)のH-EPUに対して、データキャッシュのヒット状態で1.7～3.7倍、データキャッシュのミス状態でも1.1～1.9倍という高い性能が達成できている。特に、データキャッシュのミス状態では、データキャッシュミスによるオー

Livermore Kernel	EPU (20MHz)	H-EPU (40MHz)	OHMEGA (40MHz) DCU all hit	OHMEGA (40MHz) DCU all miss
Kernel 1 (unrolling 無)	5.9 MFLOPS (0.5)	11.7 MFLOPS (1)	20.0 MFLOPS (1.7)	16.2 MFLOPS (1.4)
Kernel 1 (4 loop unrolling)	8.8 MFLOPS (0.5)	17.6 MFLOPS (1)	47.8 MFLOPS (2.7)	33.0 MFLOPS (1.9)
Kernel 3 (unrolling 無)	4.5 MFLOPS (0.5)	8.9 MFLOPS (1)	15.8 MFLOPS (1.7)	9.8 MFLOPS (1.1)
Kernel 3 (4 loop unrolling)	6.6 MFLOPS (0.5)	12.3 MFLOPS (1)	44.2 MFLOPS (3.6)	24.7 MFLOPS (2.0)

表3　性能評価

バーヘッドが存在するにもかかわらず、高い性能が得られている。

また、表4より、ループ・アンローリングというコンパイル時の最適化手法は、OHMEGAにとって非常に有効であることがわかる。これは、ループ・アンローリングによってデータ依存関係を持たない命令が増加し、命令の並列実行が効率的に行われるということと、PLD(Pair Load)命令を採用したことによるバス・アクセス回数の低減によるものである。

7．まとめ

スーパースカラー方式による命令並列実行を行うマイクロプロセッサOHMEGAを開発した。OHMEGAは、静的コード・スケジューリング、動的ハザード解消等の特徴を持ち、数値計算分野に対してその性能を最大限に発揮できるよう最適化されたマイクロプロセッサである。

性能評価においては、最適化されていないオブジェクト・コードに対しても高い実行性能を示すとともに、ループアンローリング等のコンパイラによるコードの最適化により、さらに高い実行性能を実現することができる。

参考文献

1) H.Kadota et all., "VLSI Parallel Computer with Data Transsfer Network : ADENA". Proceedings of the 1989 International Conference on Parallel Processing, Aug. 1989

2) K. Kaneko et al., "A 64b RISC Microprocessor for a Parallel Computer System", 1889 IEEE Internatinal Solid - State Circuit Conference, WPM7.2, Feb. 1989.

3) K. Kaneko et al., "A VLSI RISC with 20MFLOPS Peak 64bit Floating - Point Unit", 1889 Journal of Solid - State Circuit, Vol. 24~5, PP 1331 - 1340, Oct. 1989.

4) K. Kaneko et al., "Processing Element Design for a Parallel Computer", 1890 IEEE MICRO, Vol. 10, Num. 2, PP 26 - 38, April 1990.

5) M. Nakajima et al., "OHMEGA : A VLSI SuperScalar Processor Architecture for Numerucal Applications", Proceedings of the 18th Annual International Symposium on Computer Architecture, May 1991

6) H. Nakano et al., "A 80 MFLOPS 64 - bit Microprocessor for Parallel Computer", Proceedings of the 14th Annual Custom Integrated Ciucuit Conference, May 1991

7) K. Murakami et al., "SIMP (Single Instruction Stream / Multiple Instruction Pipeline) : A Novel Hogh - Speed Single Processor Architecture", Proceedings of the 16th Annual International Symposium on Computer Architecture, May 1989

8) J. E. Thornton et al., Design of a Computer - The Control Data 6600. Glenview, IL: Scott, Foresman and Co., 1970.

9) R. M. Tomasulo, "An Efficient Hardware Algorithm for Exploiting Multiple Arithmetic Units", IBM journal, January 1967