計算機アーキテクチャ 117－10
（1996. 3. 5）

マルチスレッドアーキテクチャに於ける動的命令発行に関する研究

細江広治† 市川周一† 島田俊夫†

メモリアクセスレイテンシの隠蔽によりプロセッサの利用効率を高めるための技術として、マルチスレッドアーキテクチャ方式が提案されている。しかしマルチスレッドアーキテクチャでは、スレッド間の命令スケジューリングを静的に行なうことが出来ない。そのため命令デコード時に機能ユニットに対する資源競合が発生し、当該スレッドからの命令発行をストールさせなければならない状況が発生し得る。本稿では、リザベーション・ステーションを拡張することにより、これを回避する方法を提案し、あわせてシミュレーションによる性能評価の結果、およびそのハードウェア量を報告する。

Research on out-of-order issue for multithreaded architecture

KOJI HOSOE,† SHUICHI ICHIKAWA† and TOSHIO SHIMADA†

Multithreaded architecture has peen proposed for efficient processor utilization by hiding memory access latency. However, decoded instructions can compete for resources among threads, which consequently makes some threads to stall. This paper describes an extension to reservation station to alleviate thread stall overhead. Some simulation results and hardware cost estimation are also presented.

1. はじめに

近年の半導体技術の急速な進歩は、VLIW/スーパースカラ等の命令レベル並列処理技術のハードウェアへの実装を可能にした。しかし、アプリケーションプログラム内に存在する並列処理可能な命令数の限界は高々2～3命令であり、機能ユニットの増設による性能向上はあまり望めないことが示されている。[1)]

そのため複数の命令スレッドを1チップ内に実装することにより、命令レベル並列よりも更に高い並列性を得ることを可能にするマルチスレッド方式の研究が、盛んに行なわれている。[2)~4)]

しかし、スレッド間で機能ユニットを共有するマルチスレッド・スーパースカラ・アーキテクチャの場合、各スレッド間のコンパイラによる静的なスケジューリングが不可能であるために、デコードされた命令が機能ユニットに関する資源競合を起こす可能性が高い。これはマルチスレッド化による弊害であり、プロセッサのスループットを低下させる原因となる。

これまでに提案された多くのマルチスレッドスーパースカラ・アーキテクチャでは、機能ユニットの資源競合が起こった場合、資源を確保できなかったスレッドをストールさせている。そこで本稿では、スレッドのストールによるプロセッサスループットの低下を防ぐハードウェア支援案を、トマシュロにより提案されたリザベーション・ステーションを[5)]拡張することにより実現し、その有用性をソフトウェアシミュレーション、ハードウェア量の見積もりを行ない評価した。

2. 動的命令発行技術

図1に、本稿で取り扱うマルチスレッドスーパースカラ方式のアーキテクチャモデルを示す。図の様に、各スレッドは独立した命令デコーダとレジスタセットを持ち、独自のプログラムカウンタを用いて動作する。本章では、図のようなマルチスレッドスーパースカラアーキテクチャにおいて、プロセッサの最高性能を引き出すために不可欠な、動的命令発行の実現とその問題点を述べる。

従来の単一スレッドプロセッサでは動的命令発行の実現のために、機能ユニット毎に動的命令スケジューリングを行なうリザベーション・ステーション方式や、分散したリザベーション・ステーションを一箇所に集めることによりハードウェアコストを削減した集中ウィンドウ方式[6),7)]等が提案された。これらの方式の中からマルチスレッドアーキテクチャに適した命令発行機構を選択する場合、次の様な条件を満たすことが前提となる。

- スレッドレベル並列性の利用によりスループットが向上することを有効に利用するために、水平方向の命令発行の拡張性に優れていること

† 名古屋大学工学部
Faculty of Engineering, Nagoya University

- スレッド制御機構及び命令発行幅の拡張により、制御が複雑にならないこと

この２つの条件を考慮に入れ、先に挙げた動的命令発行機構から最適な機構を選択する場合、集中ウィンドウ方式よりもリザベーション・ステーション方式の方が、マルチスレッドアーキテクチャには適していると思われる。何故なら、リザベーション・ステーションは各機能ユニットに分散して配置されているため、１つのリザベーション・ステーションは、ただ１つの機能ユニットへ発行される命令のみ制御すればよい。そのため、各リザベーション・ステーションの容量を増加させない限り、機能ユニットを増設しても命令処理にかかるサイクルタイムは増加しない。一方集中ウィンドウ方式では、全ての機能ユニットに対する命令発行を１箇所で集中制御している。そのため機能ユニットを増設すると、ウィンドウの容量を増加させなくとも、命令発行処理にかかるサイクルタイムは増加する。またスレッド毎に独立したレジスタファイルを持ち、更に集中ウィンドウにスレッド制御機構が導入されるため、ウィンドウに接続されるレジスタセットからのポート数が増え、サイクルタイムが増加すると思われる。

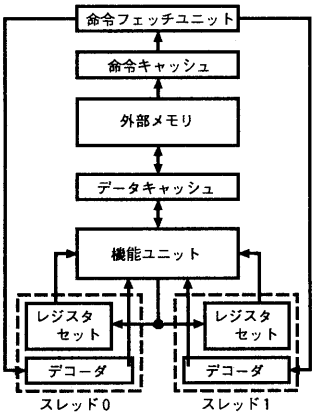

図１　マルチスレッドアーキテクチャ

しかしリザベーション・ステーションをマルチスレッド・アーキテクチャに適用すると、単一スレッドアーキテクチャでは回避できた次のような問題が発生する。つまり、スレッド間の静的な命令スケジューリングをコンパイラにより行なうことは不可能であるために、異なるスレッド間で同一の機能ユニットを使用する命令が同一サイクル中にデコードされると、機能ユニットに対して資源競合が起こるのである。資源競合が発生すると、資源を確保できなかったスレッドの命令デコードは中断されストールする。また集中ウィンドウ方式では、ウィンドウに空きがある限り同一の機能ユニットを使用する命令を発行することが出来るため、集中ウィンドウ方式を採用せずに、リザベーション・ステーション方式をマルチスレッド・アーキテクチャに採用するメリットは少なくなる。

そこで本稿では、このような命令発行時の機能ユニットに対する資源競合が起こった際の、動的な命令スケジューリング機構を、リザベーション・ステーションを拡張することにより実現する方法を提案し、その性能及び実装にかかるハードウェア量とクリティカルパスを報告する。

3. 遅延オペランドフェッチ

2章で挙げたような問題を回避するために、本稿ではリザベーション・ステーションの拡張による遅延オペランドフェッチ方式を提案する。遅延オペランドフェッチは、資源競合を起こしてオペランドデータの転送が出来ない場合に、命令発行をストールさせないで、リザベーション・ステーション実現のために拡張されたレジスタセットのタグを使用して、次サイクル以降にオペランドデータを目的のリザベーション・ステーションへ転送する機構である。

3.1 トマシュロのアルゴリズム

まず遅延オペランドフェッチを実現するためのベースとなるリザベーションステーションの動作を、簡単に説明する。図2にリザベーションステーションのハードウェア構成を示す。

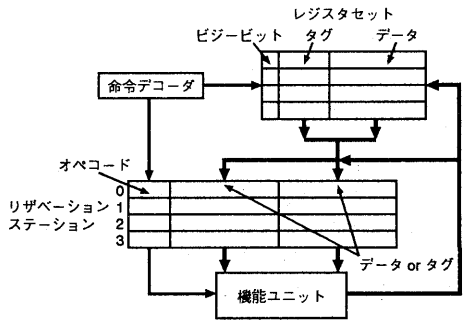

図２　リザベーション・ステーション

トマシュロのアルゴリズムでは次のようなプロセスで、真の命令依存関係による命令ストールを回避するためのレジスタリネーミングを行なっている。

各レジスタにはタグとビジービットがある。ある命令がデコードされ、リザベーションステーションに空きのエントリが存在すると、命令はそのエントリに格納され、レジスタにあるソースオペランドのビジービットがチェックされる。ビジービットは、当該レジスタの内容が近い将来最新値に更新されることを示しており、もし更新されるようであれば、RAW ハザードを防ぐために、データの代わりにタグをリザベーション・ステーションへコピーする。タグにはソースオペランドとなるデータを生成する命令が発行されたリザ

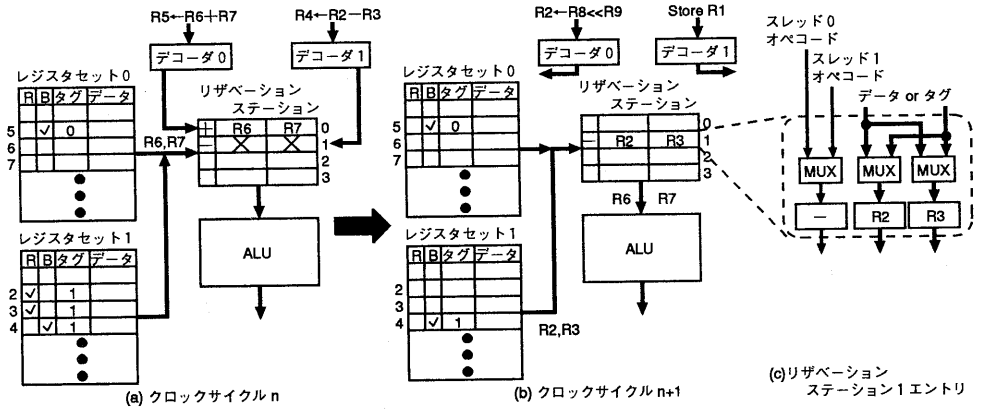

図3　遅延オペランドフェッチ

ベーションステーションの番号が記されており、そのタグを基に実行結果のフォワーディングを受ける。また、もし最新値がレジスタセットに存在すれば、そのデータをリザベーションステーションへ転送する。そしてリザベーション・ステーションにおいて必要なオペランドデータが揃うと、命令の実行が開始される。

演算結果が生成されると、各レジスタでは演算結果のタグと自分自身のタグが一致するかどうかをチェックする。もし一致するならば、自分自身のデータを最新値に更新する。

このプロセスに使用されるレジスタのタグは、あるレジスタがディスティネーションレジスタとなり、最新の演算結果によってレジスタの内容が更新されるまでの間のみ使用される。本稿で提案する遅延オペランドフェッチは、このタグが使用されていない間に有効利用する方式である。

3.2　遅延オペランドフェッチの命令発行アルゴリズム

図3に遅延オペランドフェッチの原理を示し、この図を用いて動作を説明する。ここで図中レジスタセットのBはビジービット（以下Bビット）、Rは本手法の実現のために付加した、次サイクル以降にレジスタを読み出すための予約がなされていることを示す1ビットデータ（以下Rビット）である。

まず、図の(a)のクロックサイクルnの時に、図中スレッド0はR6とR7の加算を行ないR5に格納する命令が、スレッド1ではR2からR3を減算しR4に格納する命令が、それぞれの命令デコーダでデコードされたとし、命令発行の優先度はスレッド0の方が高いとする。つまり通常の命令発行方式では、優先度の低いスレッド1の命令発行はストールする。

このような資源競合が起こった場合に於いて、遅延オペランドフェッチ機構を用いてこの問題を回避する。

まず、スレッド0では通常通り命令の発行が行なわれ、レジスタからALUへのオペランドバスは2本ともビジー状態となる（図中(a)のレジスタセット0及びオペランドバス）。次にスレッド1の命令は、リザベーション・ステーションのエントリは確保できたが、ALUへのオペランドバスが確保できなかったために、遅延オペランドフェッチ機構が働き、次サイクル以降にオペランドデータを読み出すための予約手続きが行なわれる。このときに行なわれる操作は、

- 当該レジスタのBビット、Rビットをチェックし、タグが使用されていないことをチェックする。
- 使用されていなければ、Rビットをbusyにして読み出しの予約をする。もし使用されている場合は、タグの競合により読み出しの予約は出来ないため、資源競合を起こした命令はストールする。
- 読み出し予約が出来るのなら、ディスティネーションレジスタの扱いと同様に、当該レジスタのタグに命令が発行されたリザベーション・ステーションの番号を記録する（図(a)では1番）。
- デスティネーションレジスタには通常の命令発行と同じ操作が行なわれる。

である（図(a)のレジスタセット1）。

クロックサイクルがn+1に進むと、まず各スレッドのレジスタセットの中でRビットがbusyとなっているレジスタがないかどうか検索する（図(a)レジスタセット1のR2、R3）。ここで該当するレジスタがあれば、優先的にリザベーションステーションへのオペランドバスの使用権が与えられ、Rビットは再びnot busyとなる（図(b)レジスタセット1）。またこのとき、ある機能ユニットに対して読み出し予約されたレジスタがただ1つで、オペランドバスを1本しか使わない場合、残りの1本のバスをリザベーション・ステーションにマルチプレクサを付加することにより、他のスレッド又は同一スレッドの後続命令が使用することも可能としている（図(c)）。これによりオペランドバスの有効利用がなされる。

この遅延オペランドフェッチ機構により、通常ならばクロックサイクルn+1で発行される命令R4←R2-R3を、クロックサイクルnで発行することが可能となる。

これにより、スレッド1の後続命令（図(b)Store R1以降の命令）を、それぞれ1クロックサイクルづつ早く発行することが可能となる。

本手法の実現のために行なったハードウェアの増設は、各スレッドのデコーダから、各機能ユニットのリザベーション・ステーションへ、他のスレッドとは独立に、デコード後のオペコード、即値などのデータ転送用のバスを接続。レジスタセットに、読み出し予約のためのビット付加。読み出し予約ビットを参照して遅延オペランドフェッチを実行するコントローラである。

この拡張によるハードウェアコストの増加の見積もりは5章で行なう。

4. シミュレータによる評価

ソフトウェアシミュレーションにより、本手法の性能評価を行なった。ここでシミュレーションパラメータとして、以下のような条件を設定した。

- パイプラインは命令フェッチ2ステージ、命令デコード1ステージ、演算結果書き戻し1ステージ、それぞれの機能ユニットにおける実行ステージにわかれており、1命令あたりの最小実行サイクルは5サイクルである。また命令発行は乱発行、乱終了である。
- 実装する機能ユニットは、今回ベンチマークプログラムとして採用したリバモアループの実行に必要な、整数演算器、分岐ユニット、シフタユニット、ロードストアユニット、浮動小数点加算器、浮動小数点乗算器の6個から成る。
- リザベーション・ステーションのエントリ数は以下の表のように設定した。

表1 リザベーション・ステーションエントリ数

機能ユニット	エントリ数
ALU	10
分岐	8
シフタ	2
ロードストア	10
浮動小数加算	10
浮動小数乗算	10

- 各機能ユニットの結果レイテンシは、ロードストアユニットが2サイクル、浮動小数点加算器が3サイクル、浮動小数点乗算が6サイクル、その他のユニットは1サイクルとする。
- キャッシュは1次キャッシュのみを装備し、ノンブロッキングキャッシュ[8]とする。命令キャッシュとデータキャッシュは独立しており、それぞれ容量は16Kバイト、ラインサイズは32バイトでダイレクトマップ方式とする。なおキャッシュミス時のペナルティは共に12サイクルとした。
- その他、分岐予測のためのBTBを実装し、例外からの回復のための手段としてリオーダバッファ及びフューチャーファイル[9]を採用した。

またベンチマークプログラムには、リバモアループの中でマルチスレッド化により実行サイクル数に改善が見られるもののみ(Loop 1,3,7,9,10,12)を使用した。プログラムのコードはMIPS R3000のものを参考とし、コーディング及び命令スケジューリングはリストスケジューリングを基にハンドコンパイルにより行なった。なおLoop 1とLoop 3に対しては、4重のループアンローリングを施した。

表2 多重発行率(%)

ループ番号	スレッド分割数		
	2	3	4
Loop 1	14.4	11.4	8.6
Loop 3	11.0	18.4	13.8
Loop 7	10.6	6.8	7.6
Loop 9	10.6	7.8	8.9
Loop 10	11.7	9.2	6.2
Loop 12	17.2	8.6	6.9
相加平均	12.6	10.4	8.7

表3 速度向上率(%)

ループ番号	スレッド分割数		
	2	3	4
Loop 1	14.0	9.0	5.2
Loop 3	10.5	12.0	10.6
Loop 7	10.3	6.5	4.3
Loop 9	10.1	7.1	8.3
Loop 10	7.8	5.2	4.6
Loop 12	16.7	7.7	3.4
相加平均	11.6	7.9	6.1

表2に上記の条件の下でシミュレーションを行なった場合の多重命令発行率を、表3、図4にその時の通常命令発行時に対する遅延オペランドフェッチ発行使用時の速度向上率を示す。ここで多重発行率は、全実行命令数に対する遅延オペランドフェッチ発行された命令数である。

表2、表3からわかるように、リバモアループの実行において多重発行された命令は全体平均で10.3%、速度向上は全体平均で8.5%であった。しかし、より資源競合が起こる確率の高いはずの分割スレッド数を増加させた状態で、逆に多重発行率、速度向上率は下がっている。これは命令ミックス、及び命令スケジューリングによって発行される機能ユニットに偏りが生じることにより、リザベーションステーションのエントリの不足が起こり、遅延オペランドフェッチによる命令発行が出来なくなるためであった。ここでリザベーションステーションのエントリを増加させてシミュレーションを行なっても、多重発行率は増すが、

機能ユニットの処理速度が間に合わなくなるため、速度向上率に変化は見られなかった。そのため、より高い効果を得るためには、命令発行機構以外の部分のスループット向上を計る必要がある。

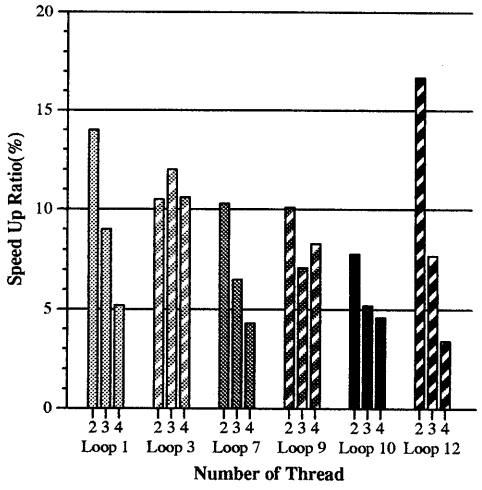

図4　シミュレーション結果

5. ハードウェア量の見積もり

今回提案した遅延オペランドフェッチ方式のハードウェアコスト、及び論理合成された回路の遅延値を見積もった。これらの測定は SFL 言語 (Structured Function Description Language) により記述し、NTT コミュニケーション科学研究所で開発された PARTHENON を使用して行なった。使用プロセスは 1.5 μ CMOS テクノロジである。

遅延オペランドフェッチ方式では、パイプライン中のデコードステージ、つまり命令デコーダ、レジスタセット、リザベーション・ステーションに対してハードウェアの拡張がなされている。以下に各部におけるハードウェア拡張部、及び拡張量を述べる。なお今回のハードウェアコストの見積もりは全て実装スレッド数が2個の場合について行なっている。

5.1 命令デコーダ

表4に命令デコーダの回路合成結果を示す。通常のマルチスレッド・スーパースカラ・プロセッサにおいては、機能ユニットの資源競合の検出はデコーダで行なわれるが、遅延オペランドフェッチ機構では資源競合を検出する回路はレジスタセットに移っている。そのためデコーダの回路規模は、資源競合を検出するための回路を持つ通常方式の方が、遅延オペランドフェッチ機構よりも若干大きくなる。

表4 デコーダのゲート数及び面積

	ゲート数	面積 (mm^2)
通常発行	2963	2.4
遅延オペランドフェッチ	2907	2.4

5.2 レジスタセット

表5にレジスタセットの回路合成結果を示す。レジスタセットにおける主なハードウェアの拡張は読み出し予約ビット、遅延オペランドフェッチのコントローラ及びレジスタから移った資源競合検出回路である。

表からわかるように、遅延オペランドフェッチ機構を導入することにより、ゲート数にして10.6%、面積では18.9%のハードウェア量が余分に必要となった。増設されるハードウェアの大部分がコントローラ部に集中しているため、レジスタ部と比べて集積度が低く、ゲート数の増加率の割に面積の増加が大きいことがわかる。

表5 レジスタセットのゲート数及び面積 (整数レジスタのみ)

	ゲート数	面積 (mm^2)
通常発行	82933	66.6
遅延オペランドフェッチ	91755	79.2

5.3 リザベーション・ステーション

表6にリザベーション・ステーションの回路合成結果を示す。リザベーションステーションでは、通常方式では1つであったデコーダからの命令及び即値の入力ポートが2つになっている。またレジスタからの2本のオペランドバスのどちらからもデータを取り込むことが出来るように、各オペランドバスのポートにマルチプレクサが付随しているため (図3(c)参照)、ゲート数は9.6%、面積は13.3%増加している。

表6 リザベーション・ステーションのゲート数及び面積
(ALU 用、10 エントリ)

	ゲート数	面積 (mm^2)
通常発行	13071	10.5
遅延オペランドフェッチ	14331	11.9

5.4 クリティカルパス

図5にデコードステージ全体のクリティカルパスを示す。これらのクリティカルパスは、先に述べた各部をさらに機能毎に分けたシミュレーションにより求めている。デコードステージ全体では、通常命令発行方式では 89.1ns、遅延オペランドフェッチ方式では 90.8ns となり、クリティカルパスは 1.9%(1.7ns) 増加するという結果になった。

しかしここで重要なことは、命令デコード時間 33.8ns に対し、遅延オペランドフェッチ処理にかかる時間が 32.9ns であるために、本機構の操作時間を完全に隠蔽出来る点である。また、レジスタファイルにおけるオペランドフェッチ、リザベーション・ステー

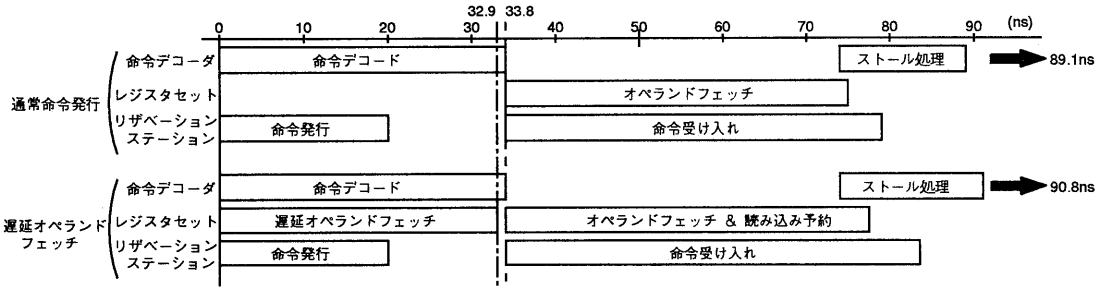

図5 デコードステージクリティカルパス

ションにおける命令受け入れ処理に要する時間は、通常命令発行方式でそれぞれ 41.2ns,45.2ns、遅延オペランドフェッチ方式では 43.8ns,49.6ns と増加している。しかしデコーダでストール処理を行なっているため、これらの時間も隠蔽されている。

だが従来方式ならばデコーダで発見できた機能ユニットの資源競合が、本機構ではレジスタセットにおいて読み出し予約に失敗した時に発見される。そのため命令デコーダに於けるストール処理時間は、通常発行方式 16.0ns に対し遅延オペランドフェッチ方式 17.7ns と、1.7ns 余計にかかっている。そのため全体的に 1.7ns の増加となっている。

6. ま と め

マルチスレッド方式を取り入れたスーパースカラ・アーキテクチャに於いて、動的な命令発行方式にリザベーション・ステーションを採用した場合に起こる、スレッド間の機能ユニット資源競合の回避法に、リザベーション・ステーションを拡張した遅延オペランドフェッチ方式を提案した。ソフトウェアシミュレーションの結果、実装時のクリティカルパスの増加 1.9%に対して平均 8.5%の速度向上が得られることが確認された。

しかし、より資源競合の起きる確率の高い、スレッド分割数が増加した場合に於いてその能力が十分発揮できていないため、さらに効率的な動的命令発行機構に進展させる必要がある。

参 考 文 献

1) N.P.Jouppi,and D.Wall: Available Instruction Level Parallelism for Superscalar and Superpipelined Mach ines. Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems (Apr, 1989), pp.290-302.

2) 平田博章、木村浩三ほか: 多重スレッド多重命令発行を用いる要素プロセッサ・アーキテクチャ, 並列処理シンポジウム. 並列処理シンポジウム JSPP'92 論文集 pp.257-264.

3) George E.Daddis Jr. and H.C.Torng: The Concurrent Execution of Multiple Instruction Streams On Superscalar Process ors. 1991 International Conference on Parallel Processing, pp.76-83.

4) R.Guru Prasadh and Chuan-lin Wu: A Benchmark Evaluation of a Multi-Threaded RISC Processor Architecture. 1991 International Conference on Parallel Processing, pp.84-91.

5) Tomasulo,R.M: An Efficient Algorithm for Exploiting Multiple Arithmetic Units. IBM Journal,Vol.11 (Jan, 1967), pp.25-33.

6) R.D.Acosta and H.C.Torng: An Instruction Issuing Approach to Enhancing Performance in Multiple Function Unit Processors. IEEE Transactions on Computers,Vol.C-35,No.9 (Sep, 1986), pp.815-828.

7) Gurindar S.Sohi and Sriram Vajapeyam: Instruction Issue Logic For High-Performance, Interruptable Pipelined Processors. Proceedings of the 14th Annual International Symposium on Computer Architecture (1987).

8) Gurindar S.Sohi and Manoj Franklin: High-Bandwidth Data Memory Systems for Superscalar Processors. Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (1991), pp.53-62.

9) Smith,J.E.,and A.R.Pleszkun: Implementation of Precise Interrupts in Pipelined Processors. Proceedings of the 12th Annual International Symposium on Computer Architecture (1985).