計算機アーキテクチャ 134－28
（1999．8．4）

Dualflowアーキテクチャとそのコード生成手法

五島正裕[†] グェンハイハー[†] 縣 亮慶[†]
森 眞一郎[†] 富田眞治[†]

Dualflowは，制御駆動とデータ駆動の性質をあわせ持つプロセッサ・アーキテクチャであり，スーパースケーラのようなout-of-order実行をVLIWと同等のサイクル・タイムで実現することができる．ただしそれはコードに対する制約の上に成り立っており，superscalarに比べて実行命令数の増加が避けられない．総合的な評価のため，GCCをベースにしたコンパイラを作成した．本稿では，そこで用いたDualflowのコード生成と最適化の手法について報告する．

Code Generation Method for the Dualflow Architecture

MASAHIRO GOSHIMA,[†] NGUYEN HAI HA,[†] AKIYOSHI AGATA,[†]
SHIN-ICHIRO MORI[†] and SHINJI TOMITA[†]

Dualflow, a hybrid processor architecture between control- and data-driven, can realize out-of-order execution of superscalar with cycle time of VLIW. But it is achieved at the cost of restriction over the code, it requires extra instructions compared with ordinal superscalars. We made a compiler based on GCC in order to evaluate total performance. In this paper, the code generation and optimization scheme used by the compiler is reported.

1. はじめに

Superscalarのout-of-order実行には，真に動的な擾乱に対しても柔軟であるという利点がある．例えばロード命令がキャッシュ・ミスを起こしたとき，VLIWは依存する命令以降を一切実行できないのに対して，superscalarは依存しない命令を発見し，実行を続けることができる．

しかしout-of-order実行機構は非常に複雑なハードウェアであり，サイクル・タイムの点ではVLIWに対して圧倒的に不利である．命令の発行多重度やウィンドウ・サイズが増加するにしたがってその傾向はいっそう顕著になる．

このような背景から我々は，superscalarのout-of-order実行をVLIWと同等のサイクル・タイムで実現する，**Dualflow**と呼ぶプロセッサ・アーキテクチャを提案した[3]．本稿では，そのコード生成と最適化手法について報告する．

以下，2章でDualflowのアーキテクチャについて概説した後，3章でコード生成と最適化手法について述べる．

2. Dualflow

Dualflowは，制御駆動型とデータ駆動型を融合したアーキテクチャである．データ駆動的性質を導入することによって，out-of-order実行のハードウェアを簡単化することができる．以下，2.1節でまず実行モデルについて述べた後，2.2節で実装方法について説明する．

2.1 Dualflowの実行モデル

Dualflowは，以下のように，制御駆動とデータ駆動の両方の性質を合わせ持つ：

制御駆動　プログラム・カウンタがあり，分岐命令によって制御フローを移譲する．

データ駆動　制御駆動では，命令間のデータの授受はレジスタを介して間接的に行われる．一方Dualflowは，レジスタを持たず，データ駆動のように命令間で直接的にデータの授受を行う．

ただし，どちらかと言えば，制御駆動をベースにデータ駆動の性質を導入した感じである．以降では，敢えて述べない部分に関しては通常のsuperscalarと同じであると考えられたい．

以下では，まず全体についてより詳細に述べた後，実行例を用いてモデルの説明を行う．

† 京都大学 情報学研究科
Graduate School of Informatics, Kyoto Univ.

実行モデル

Dualflow の実行は，プレースの順序付けられた無限の列の上で行われる．各プレースは，3 つのスロットからなる．スロットの 1 つは命令を格納する命令スロットであり，残りの 2 つは命令の左右のソース・オペランドを格納するデータ・スロットである．

各プレースには，命令とデータがばらばらに届く．以下のように，命令は制御駆動的にデータはデータ駆動的にプレースに届く：

命令　命令は通常の制御駆動と同様にプログラム・カウンタにしたがってメモリからフェッチされ，フェッチされた順序でプレース列の命令スロットに順に格納されていく．

データ　各プレースが実行されるとその結果は，データ駆動と同様に，命令中に示される宛先に送られる．ただし，宛先の指定の方法はデータ駆動とは異なる．基本的なデータ駆動では，実行結果の宛先は命令であり，各命令は宛先の命令のアドレスを指示する．一方 Dualflow では，宛先は命令ではなくデータ・スロットである．

命令と必要なオペランドが揃ったプレースが，プレースの順序とは無関係に，すなわち，out-of-order に実行される．すなわち，データ駆動では命令のあるところでデータとデータが待ち合わせるが，Dualflow ではプレースで命令とデータが待ち合わせるのである．

ここで，Dualflow の命令フォーマットを図 1 に示そう．命令は 32b 固定長を想定する．宛先は d1/d2 フィールドで示される．各フィールド中，offset サブフィールドは自命令の格納されたプレースと宛先データ・スロット間の変位を，s サブフィールドは宛先データ・スロットの左右を，それぞれ表す．offset サブフィールドは 5b であるから，実行結果を送ることができるのは，距離が 31 以内にある最大 2 つのデータ・スロットに制限される．

実　行　例

では，図 2 に示す |a − b| を計算するプログラムを例に，Dualflow の動作を具体的に説明しよう．このプログラムは，まず，3 行の sub 命令で d = a − b を求める．そして d が負である場合には 4 行の bneg 命令から NEG に分岐し，更に 0 − d を計算して最終的な結果とする．

図 3 に，実行後のプレース列の様子を示す．図中左/右が，4 行の条件分岐命令 bneg が not taken/taken

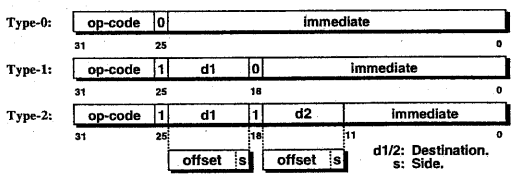

図 1　命令フォーマット：Type-x は，宛先の数が x 個

```
line label   instruction
  1          imm   a     2L
  2          imm   b     1R
  3          sub         1L, 2L
  4          bneg NEG
  5          mov         2L
  6          b END
  7    NEG:  subr 0      1L
  8    END:  mov         X
```

図 2　|a − b| を計算するプログラム

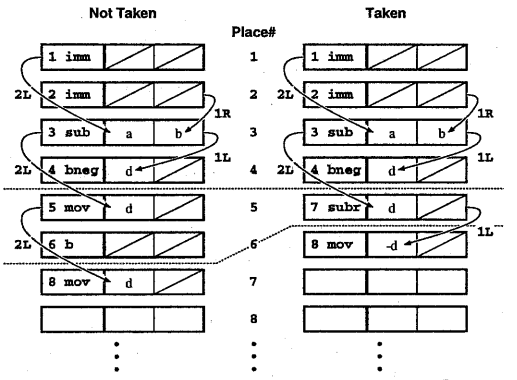

図 3　図 2 のプログラムを実行後のプレースの状態：Not Taken/Taken は，条件分岐の結果

であった場合を表す．

プログラムは以下のように実行される：

(1)　最初プログラム・カウンタは 1 行を指している．この時点で 4 行の条件分岐命令 bneg までの制御の流れは確定しているので，1〜4 行の各命令を，プレース 1〜4 の命令スロットにそれぞれ格納することができる．

(2)　1/2 行の imm 命令は即値を生成する命令で，データを必要としない．したがってフェッチ後直ちに実行されて，値 a/b を 2L/1R で示されるプレース 3 の左/右データ・スロットに送る．

(3)　プレース 3 は，3 行の sub 命令と，1/2 行の imm 命令からのデータの到着によって実行可能となり，実行結果 d は 1L/2L で示されるプレース 4/5 それぞれの左データ・スロット送られる．2L で示されるプレース 5 に入る命令は，条件分岐命令 bneg の結果に依存するので，この時点ではフェッチできないことに注意されたい．したがってこの sub 命令は，そこにどのような命令が来るかに関わらず，プレース 5 に実行結果を送りつけることになる．一方プレース 5 は，命令より先にデータを受け取ることになる．これは，データ駆動ではあり得ないことであるが，superscalar ではごく普通のことである．

(4)　プレース 5 の命令スロットには，bneg が not

takenであれば5行のmovが，takenであれば7行のsubrが格納される．
以降はtakenであった場合（図3右）について説明する．この時点でプレース5以降の制御の流れは確定する．

(5) subrは，subとは逆に，右オペランドから左オペランドを減ずる命令である．この場合は即値0を持っているので，0－dを計算する．subrがプレース5の命令スロットに格納される時点で，データdは既に到着しているので，このプレースはフェッチ後 直ちに実行される．

(6) 結果−dはプレース7のmov命令によってXに送られる．
当然ではあるが，4行の条件分岐bnegによって実行命令数が異なり，このmov命令が格納されるプレースも異なる．

制御駆動では命令が，データ駆動ではデータが，それぞれ計算の主体であると言われる．そのような観点から言えば，Dualflowでは，命令とデータのどちらかが主でどちらかが従であるということはない．

本節で示したように，Dualflowは，データ駆動的性質を強く持つ実行モデルを採用しているが，それは，out-of-order実行のハードウェアを簡単化するためである．次節では，実際にこのことがどのようにハードウェアを簡単化するのかを説明する．

2.2 Dualflowの実装

本章では，まず，superscalarのout-of-order実行機構の複雑さについて述べた後，それがDualflowでどのようにして軽減されるか説明する．

Superscalarの Out-of-Order実行の複雑さ

Superscalarのout-of-order実行は，各命令，あるいは，各命令が生成する個々のデータに対し動的にタグを割り当て，タグによってそれぞれのデータを識別することでデータ駆動型の計算をエミュレートすることであるとみなせる．タグとしては，物理レジスタやリオーダ・バッファのエントリの番号が用いられる．

Superscalarのout-of-order実行は，以下の3つの処理からなる：

タグ変換 仮想レジスタ番号からタグへの変換を行う．

タグ一致検出 命令が結果を出力する時に，そのタグをキーとする連想アクセス，すなわち，命令ウィンドウのすべてのエントリの一致比較器にタグを放送することによって，その結果を待っている命令を検出する．

命令選択 タグ一致検出の結果 発行可能であることが判明した命令の中から，優先順位付比較によって，実際に発行する命令を選択する．

これらのうちタグ変換のレイテンシは，複数サイクルを割り当てることで分岐予測ミスのペナルティに転化することができる．ただしその結果，VLIWでは通常1〜2サイクル程度である分岐予測ミス・ペナルティは，superscalarでは3〜4サイクルにもなる．

一方タグ一致検出と命令選択は，タグ変換とは異なり，2つ合わせて1サイクルで実行する必要がある．なぜなら，発行する命令を決定してから次に発行する命令を決定するまでを1サイクルで実行しないと，実行ステージのレイテンシが1サイクルである命令の結果をバイパスできなくなるためである．このタグ一致検出と命令選択のレイテンシの合計が，superscalarのサイクル・タイムを決定する最も深刻な要因の1つになっている．

命令選択のレイテンシが命令ウィンドウのサイズの対数にしか比例しないのに対し，タグ一致検出のレイテンシは命令発行多重度の2乗と命令ウィンドウ・サイズの2乗の積に比例する．また，タグ一致検出における放送のレイテンシは，ゲート遅延ではなく配線遅延であり，LSIの微細化の恩恵を受けにくい．したがって，LSIがより微細化され，発行多重度とウィンドウ・サイズがより拡大していくにしたがって，タグ一致検出のレイテンシが最も深刻な影響を及ぼすと予想される[2)]．

Dualflowにおける Out-of-Order実行機構

前述したout-of-order実行の3つの処理のうち，命令選択はout-of-order実行にとって本質的であるが，タグ変換とタグ一致検出はデータ駆動型の計算をエミュレートするための処理であり，無駄である．Dualflowでは，以下のようにしてその2つの処理をほとんど完全に省略することができる．

さて，superscalarのout-of-order実行機構の実装方式は，物理レジスタに対するリネーミングを基本とするタイプと，そうではないタイプに分けられる．現存する例としては，R10000[1)]，21264などが前者に；PA-8000，PowerPCなどが後者に属する．Dualflowの実装は，前者により近いものとなる．

Dualflowの実装では，**Rendezvous Station**（以下RSと略）と呼ぶサイクリック・バッファがout-of-order実行機構の中心的役割を果たす．図3に示した振る舞いがそのままRS上で実現されると考えてほとんど差し支えない．

ただし，RSはサイクリック・バッファであるので，若干の注意が必要である．プレース列のうち，未完了の部分だけがRSの上に載せられる．実行を完了したプレースは古いものから順に捨てられ，エントリはサイクリックに再利用される．

RSの深さは，宛先の変位の最大値から，少なくとも32必要である．RSはサイクリック・バッファであるので，宛先のプレースが古い未実行の命令で占められていることがある．その場合，宛先プレースにある命令が完了するまでその命令の実行ができない．したがって，このことがボトルネックにならないように，深さは64程度であることが望ましい．またRSの深さは，ウィンドウ・サイズの上限を与える．

Dualflow の Out-of-Order 実行の複雑さ

タグ一致検出の処理は，Dualflow では以下のようになる：RS エントリ番号に d1/d2 フィールドの値を加えることによって，実行結果の宛先となるプレースのエントリ番号を求められる．したがって，連想アクセスは必要なく，直接アクセスによって待っている命令を検出することができる．

タグ一致検出と命令選択は合わせて 1 サイクルで実行する必要があると述べたが，タグ一致検出のレイテンシがほとんど無視できるので，命令選択をほぼ 1 サイクルかけて実行すればよい．命令選択のレイテンシがサイクル・タイムを制約する可能性は低い．また命令選択のレイテンシは命令ウィンドウ・サイズの対数にしか比例しないので，ウィンドウ・サイズの拡大に対する余裕が大きくなる．

一方タグ変換に対しては，特別な処理は一切必要ない．そのため，デコード・ステージ数，および，分岐予測ミス・ペナルティは，VLIW と同程度にできる．

このように Dualflow は，superscalar に比べて out-of-order 実行機構を大幅に簡略化することができる．それは，実行モデルのレベルからデータ駆動的性質を導入することによって，データ駆動型計算をより直接的に実現できるからであると言える．

一方でこのデータ駆動的性質はコードに対して制御駆動より強い制約を課すことになる．そのため，Dualflow プロセッサの性能はコンパイラの能力により強く依存する．次章では，Dualflow のコード生成について述べる．

3. Dualflow のコード生成

Dualflow アーキテクチャの評価のため，C コンパイラとインタープリタを作成した．以下，まず 3.1 節では，ベースとした用いた GNU C コンパイラ，GCC についてまとめる．次いで，3.2 節で Dualflow 専用パスについてまとめた後，3.3 節で実装した最適化の手法について述べる．

3.1 GCC

Dualflow の C コンパイラを作成するにあたっては，GCC (ver.2.8.1) をベースにした．GCC を通常の制御駆動のプロセッサにポーティングするには，基本的には，Machine Description File と Target Macro という 2 つのファイルをプロセッサの構成に合わせて記述するだけですむ．Dualflow にポーティングするには，それに加えて専用のパスを追加する必要があるが，多くのパスはそのまま流用することができる．

RTL と疑似レジスタ

GCC は，入力ファイルを解析し，Register Transfer Language ——— **RTL** と呼ぶ内部表現を生成する．以下のパスでの作業は，すべてこの RTL に対して行われる．

GCC はまず，無限の数の疑似レジスタがあると仮定して RTL を生成する．疑似レジスタは，静的ではあるが，個々のデータを識別できるタグである．疑似レジスタは，下流のパスで，ハードウェアのレジスタに変換される，すなわち，レジスタ割り当てが行われる．

Superscalar は，タグから変換されたレジスタ番号から，また動的にタグを求める．一方 Dualflow では，疑似レジスタは直接的に宛先に変換され，実行時には宛先をやはり直接的に参照する．

疑似レジスタから宛先への変換は，ごく基本的には，ある疑似レジスタを定義する命令から参照する命令までの命令数を数えればよい．

逆に，一旦レジスタ割り付けが行われてしまった後では，データの同一性に関する情報が失われてしまうので，そこから静的に宛先を求める作業は簡単ではない．Dualflow の実行コードを得るにあたって，既存のマシンのアセンブリ・コードからのトランスレータを製作するのではなく，GCC を流用する方法を選んだのは，主にこの理由による．

パ　　ス

GCC は，以下のパスにしたがって処理を行う：

(1) 構文解析．RTL の生成．
(2) 最適化．データ・フロー解析．
(3) レジスタ割り当て．Spill-out 処理．
(4) Peep-hole 最適化．
(5) アセンブリ・コードへの変換．

Dualflow 専用パス群は，疑似レジスタに基づく RTL を入力とする必要がある．また，(4)Peep-hole 最適化は，ハードウェア・レジスタに基づいて行われるので利用しづらく，利用価値も低い．したがって Dualflow 専用パス群は，(3) レジスタ割り当てと (4)Peep-hole 最適化の代わりに組み込む．(5) アセンブリ・コードへの変化を行うパスは，そのまま利用できる．

レジスタ割り当てより前のパス (1)，(2) は，Dualflow 向けコンパイラでも全くそのまま利用することができる．特に，主要な最適化処理のほとんどとデータ・フロー解析がこの部分で行われていて，非常に都合がよい．ここで行われる最適化の中には，定数の畳み込み，計算強度の軽減，分岐のスレッド化 (jump threading)，無用命令の削除，共通部分式の削除，ループ最適化，局所命令スケジューリングなどがある．

したがって Dualflow 専用パスには，最適化，データ・フロー解析済の RTL が渡されることになる．Dualflow 専用パスついては，次節で詳しく述べる．

3.2 Dualflow 専用パス群

Dualflow は，データ駆動的性質を導入することによって out-of-order 実行機構を大幅に簡略化するが，その代償として，コードはデータの授受に関する制約を受ける．それは，宛先フィールドを静的に計算するという制約である．制約は，以下の 3 つに分類できる：

数と距離　2 個を越えるスロット，あるいは，32 以上

離れたスロットにはデータを送れない.

条件分岐　データの授受が条件分岐を越える場合にも，分岐の結果によって宛先を変えることはできない.

基本ブロック　データの授受が1つ以上の基本ブロックを越える場合には，命令間の距離，すなわち，宛先フィールドの値を静的に求めるとができない.関数呼び出し，if-then-else構造，ループなどを越えたデータの授受が，これにあたる.

3つの場合のそれぞれについて，以下のようにして制約を満たす：

数と距離　データを中継するためのmov命令を挿入する．各mov命令は宛先を2つ持てるので，2分木を組むことができる．2分木の形状は，数と距離を考慮して決める.

条件分岐　taken/not taken側それぞれの基本ブロック内部で適当にmovを挿入することによって受け取るデータ・スロットの配置を一致させる.

基本ブロック　関数呼び出しをまたぐ場合には，距離を静的に決めることは原理的にできないので，メモリを介してデータの授受を行う必要がある．ちょうど，制御駆動においてすべてのレジスタをcaller-saveとした場合と同じと考えればよい.

その他の場合には，間にある各基本ブロックに中継のためのmov命令を挿入する．図2で示したプログラムでは，5行のmovがこれにあたる.

なおいずれの場合でも，距離が長くなりすぎるようであれば，メモリを介してデータの授受を行う方法を採る.

それぞれの処理において，単純な実装では大量のmov命令が挿入されてしまうだろう．このようなmovに対しては，さまざまな最適化手法が考えられる．今回は，最も基本的と思われる条件分岐に対する最適化のみを実装した．次節では，それについて詳しく述べる.

3.3　条件分岐に関する最適化

データの授受が条件分岐をまたぐ場合，データを受け取るスロットの配置を，taken側とnot taken側で同一にしなければならない.

例えば，図3に示したプログラムの実行例の場合，プレース3の実行結果のうち2Lで示されるものは，プレース4の条件分岐命令bnegを越えて，プレース5の左データ・スロットに渡される．このプレース5の命令スロットには，bneg命令の結果によって7行か5行の命令がフェッチされるが，どちらの場合でも正しくなるように7/5行に置く命令を調整する必要がある.

最も簡単には，データのそれぞれに対して，それを受け取るmovを各基本ブロックの先頭に配置すればよい．もちろんそのような方法では受け入れ難い数のmovが挿入されてしまうので，それをどう削減するかが問題である.

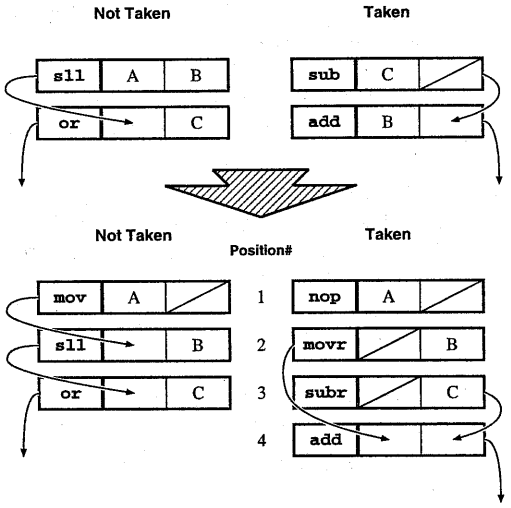

図4　条件分岐の処理

アルゴリズム

では図4に示す例を用いて，今回実装した最適化について説明しよう．図中の上側に示す命令列が入力である．図中のA，B，Cは，それぞれに対応するデータを上流の基本ブロックから受け取るスロットを表す．したがって，A，B，Cの配置をnot taken側/taken側で合わせればよい.

処理は，以下の条件下で実行される：

- このパスを含むDualflow専用パス群の入出力は，実際には，疑似レジスタに基づいたRTLである．前述したように入力されるRTLは既にデータ・フロー解析済である.
- 前節で述べた3種の制約のうち，数と距離に対する処理はこのパスの後で実行される．したがってこのパスでは，数と距離の制約がないものとして処理を行えばよい.
- 分岐確率に偏りがある場合とない場合では最適化の方法を変えた方がよいのは明白であるが，今回はまず分岐確率に偏りがあるものとして実装した.

基本的には，分岐確率が高い側に最適化し，低い側でつじつまを合わせる．例では，not taken側が分岐確率が高いものとする．処理は，以下のように進められる：

(1)　競合の解消.
　受け取り位置に競合がある場合，まずその状態をmovにより解消する.
　図4では，命令sllはデータA，Bを；命令addはBのみを受け取っている．not taken側にAを受け取るmovを挿入し，AとBを受け取る命令を分離する.

(2)　再スケジューリング.
　分岐確率が高い方の基本ブロックに対し局所命

令スケジューリングを施し，(1) で挿入された mov の位置を最適化する．
実際には，GCC の命令スケジューリング・パスを呼び出し，プログラム中の全ての基本ブロックに対して再びスケジューリングを施すこととした．

(3) スロットの配置の整合．
分岐確率が高い側の基本ブロックの受け取りスロットの配置に合わせて，低い側の基本ブロック内の命令を並び替える．
オペランドの左右を入れ換えられる命令では，必要に応じて入れ換えを行う．入れ換えは分岐確率の高い側でも行ってよい．なお，オペランドの左右を入れ換えられる命令には，ロード（アドレス計算），加減乗算（整数および浮動小数点数），論理演算（and，or，xor），そして，mov，nop がある．
図 4 では，taken 側の命令を再配置する．位置 1 には，A を握り潰すための nop 命令を置く．左右を入れ換えれば位置 2 に add 命令を置くことは可能だが，そうすると sub 命令との先行制約が満たせない．したがって add 命令は後にまわして，位置 2 には movr を置く．また位置 3 には，sub 命令の左右を入れ換えた subr 命令を置くことができる．

この処理の結果，図 4 下に示す命令列を得る．

4. おわりに

Dualflow は，その実行モデルのレベルからデータ駆動的性質を導入することによって out-of-order 実行機構を大幅に簡略化することができ，サイクル・タイムの点で superscalar より有利である．ただし一方でデータ駆動的性質はコード生成上の制約になるので，無用な命令の増加による性能低下が懸念された．

そこで全体的な評価ため，実際に GCC をベースとして C コンパイラを作成した．本稿では，そこで用いたコード生成と最適化の手法について述べた．

ただし，このコンパイラはまだまだ不完全で，今回は評価結果を出すには至らなかった．今後はコンパイラを完全なものとするとともに，詳細なシミュレータを作成し，より厳密な性能評価を行う予定である．また，実際にゲート・レベルの設計を行うなどして，サイクル・タイムの評価を行う必要があるだろう．

謝辞

メンター・グラフィックス・ジャパン株式会社には，Higher Education Program の一環として製品とサービスをご提供いただいたので，ここに感謝する．

本研究の一部は文部省科学研究費補助金（基盤研究(B)(2) #10558045，および，(C) #09680334）による．

参考文献

1) C. Yeager, K.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, No. 4, pp. 28–40 (1996).
2) Palacharla, S., P. Jouppi, N. and E. Smith, J.: Complexity-Effective Superscalar Processors, *ISCA24* (1997).
3) 五島正裕, グェン ハイハー, 森眞一郎, 富田眞治: Dual-Flow: 制御駆動とデータ駆動を融合したプロセッサ・アーキテクチャ, 情処研報 98–ARC–130, pp. 115–120 (1998).