2022 年度 3 回生前期学生実験 HW

team02 アーキテクチャ拡張仕様書

アーキテクチャ拡張仕様書: 植田健斗
グループメンバー：
伊藤舜一郎 (学籍番号:1029-32-7548)
植田健斗　(学籍番号:1029-32-6498)

提出期限：6 月 9 日 13 時　提出日: 2022 年 6 月 3 日

1　SIMPLE/B 基本アーキテクチャからの拡張仕様の説明

SIMPLE/B 基本アーキテクチャをもとにして、拡張機能の追加をしたプロセッサを設計した。以下の機能を追加した。

- 5 段のパイプライン処理
- 即値命令 ADDI の実装

2　拡張を行うことによる利益

拡張を行うことで、プロセッサの実行時間が短くなった。以下それぞれの拡張により向上した性能を説明する。

3　5 段のパイプライン処理

3.1　概要

処理を IF,ID,EX,MEM,WB の 5 つのフェーズに分けてそれらを並列に処理を行う。その実装に伴いフォーワーディングユニット、ハザード検出ユニット、パイプラインレジスタを追加し、制御部を一部変更した。

データハザードに対処するためにフォーワーディングを実装した。さらに、LD,IN 命令の MEM フェーズの終わりにわかるデータを次の命令の EX フェーズで使う状況で起こる、フォーワーディングで対処できないデータハザード (パイプラインハザード) には対処してある。具体的には LD,IN 命令で読み込むデータを次の命令で参照する際に LD,IN 命令の次に NOP 命令を挟むことで、対処している。

また、分岐の際の制御ハザードに対処している。次のようにして制御ハザードに対処している。分岐が不成立であると予測して命令を読み込む。分岐が不成立の場合、そのまま処理を続ける。成立した場合、不成立と予測して読み込んだ命令が 1 つあるので、それを NOP 命令に置き換えて、分岐先の新しい番地の命令を読み込む。

3.2　性能の評価

中間報告時の拡張前のマルチサイクル方式での SIMPLE/B では CPI は 5 であったが、最終報告時のパイプライン方式での SIMPLE/B では CPI は以下のようになる。ただし CPI が以下のようになるまでに 4 サイクルかかる。

- ロード命令 (LD) は 1 だが、ロード命令で書きこむレジスタの内容を次の命令で読み出す場合は 2
- 入力命令 (IN) は 1 以上で、exec ボタンが押されるまでのクロック数。
- 条件分岐・無条件分岐命令はいずれも、分岐が不成立の場合は 1、成立する場合は 2
- その他の命令は、すべて 1

これにより、このプロセッサの平均的な CPI は命令数が多い場合は 1 以上 2 以下の値となる。以下のことを行うことで CPI を 1 に限りなく近くすることができる。

- ロード命令 (LD) の次の命令で、LD 命令で読み込むデータを参照しない。
- ループ展開などにより、条件分岐命令を減らす。
- 命令数を多くする。

また、中間報告時のマルチサイクル方式の SIMPLE/B では Total logic element が 2129 であり、最終報告のパイプライン方式の拡張後の SIMPLE/B では Total logic element が 2556 である。パイプライン方式になり Total logic element は 427 増えた。もともとのマルチサイクル方式のおよそ 1.2 倍の回路規模になっている。

4 即値加算命令 ADDI 命令の実装

4.1 概要

命令の上位 5bit が二進数で 10001 のときにこの命令が呼ばれる。Rb フィールド (命令の 10 から 8bit 目) で指定した番号のレジスタの中身を読み出し、命令の下 8bit で指定した符号付きの値 d を足し合わせる。足し合わせた値を Rb フィールドで指定したレジスタに格納する。Rb フィールドの値により、条件コードが設定される。条件コード C には最上位ビットからの桁上げが設定される。

4.2 性能の評価

制御信号を書き換えのみを行い実装したため、追加のコンポーネントはないので拡張命令のコスト遅延はほとんどないと思われる。ADDI 命令の利点としてソートのアルゴリズムを書く際に、-126 以上 125 以下の定数を加算したいときに、ADDI 命令がない場合は LI と ADD 命令を組み合わせ 2 命令かけて実装しなければならないがそれを 1 命令で実行できることがあげられる。

ADDI がない場合、ループ内でインデックスをインクリメントするときに、LI と ADD 命令を使い 2 命令かけて実行するか、それとも、一命令でやる場合にはレジスタに 1 を保存しておく必要がある。ADDI がある場合、レジスタに 1 を保存せずに 1 命令で実行することができるので、レジスタを使わずに実行命令数を増やさないということができる。

実際、度数ソート (dosuSort.mif) のプログラムの一部 (36 から 53 行目) を例に挙げて定量的な評価をおこなう。以下は 0 番レジスタの値がループインデックスになっていて、1024 回の繰り返す 1 重のループ命令になっている。このコードのループを実行されるときには、レジスタは 8 個すべて使われている。0 から 4 番レジスタも変数を格納するのに使っていて、5,6,7 番レジスタはそれぞれ定数 0x100,0x800,0xff の定数を格納するのに使っている。この状況で 1024 回のループを回すのに、定数 1 を保存するためのレジスタを確保できない状況で、0 番レジスタの値 (idx) をインクリメントしなければいけない。

```
LI 0, 1
SLL 0, 10
LI 7, -1
SRL 7, 8
LD 3, 0, 0
MOV 1, 7
AND 1, 3
LD 4, 0, 1
MOV 2, 1
ADD 2, 5
LD 2, 0, 2
ADDI 0, 1
SUB 2, 4
ADDI 4, -1
ST 3, 0, 2
ST 4, 0, 1
CMP 0, 6
BNE -14
```

このような状況で、定数 1 を格納するレジスタは確保することができないので、ADDI 命令がなければ定数レジスタを一つ使い、例えば 0x100 を格納する 5 番レジスタを使い、それをシフトすることで定数 1 を作らなければならない。以下の変更をすることで ADDI 命令がない場合に同じ処理を実行できる。

- ADDI 0, 1 がある場所の上に、SRL 5, 8 を追加する。
- ADDI 0, 1 を ADD 0, 5 に変える。
- ADDI 0, -1 を SUB 0, 5 に変える。
- ADD 2, 5 命令の上に SLL 5, 8 命令を追加する。
- BNE -14 を BNE -16 に変更する。

以下がこれらの変更をし、ADDI を用いずに同じ内容を表現したアセンブリコードである。

```
LI 0, 1
SLL 0, 10
LI 7, -1
SRL 7, 8
LD 3, 0, 0
MOV 1, 7
AND 1, 3
LD 4, 0, 1
MOV 2, 1
SLL 5, 8
ADD 2, 5
LD 2, 0, 2
SRL 5, 8
ADD 0, 5
SUB 2, 4
SUB 4, 5
ST 3, 0, 2
ST 4, 0, 1
CMP 0, 6
BNE -16
```

ADDI が込みでの命令では、14 命令と分岐成立の際に挿入される NOP で 1 回のループ当たり、15 クロックかかるので、初めの 4 命令と合わせて $4+1024*15 = 15364$ クロックサイクルかかる。ADDI がないと、この例ではループ内の命令数が 2 つ増える。このループは 1024 回繰り返されるので、この例では ADDI がある場合のほうがない場合より 2048 クロック多くなり、$15364 + 2048 = 17412$ クロックサイクルになるということがわかる。

この例では ADDI が実装されたことでクロックサイクル数は約 0.882 倍になる。さらに、50MHz で実行する場合を考えると、20ns × 2048 で 40 μ s ほど実行時間が短くなる。