アーキテクチャ設計検討報告書

23 班：執筆者　神事倫紀

執筆日:2023 年 5 月 11 日

1　要求仕様、設計目標、設計方針、特長

1.1　要求仕様

以下の命令セットアーキテクチャに対して以下のように動作することが要求されている。

- 算術演算：レジスタ Rd と Rs の加算 (ADD: add) または減算 (SUB: subtract) の結果を Rd に格納し，条件コードを設定する．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．
- 論理演算：レジスタ Rd と Rs の，ビットごとの論理積 (AND: and), 論理和 (OR: or)，または排他的論理和 (XOR: exclusive-or) の結果を Rd に格納し，条件コードを設定する．但し条件コード C は演算結果に関わらず 0 となる．
- 比較演算：レジスタ Rd から Rs を減算し，結果に基づく条件コード設定のみを行う．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．
- 移動演算：レジスタ Rd に Rs の値を単に格納し，Rd の値に基づき条件コードを設定する．但し条件コード C は Rs の値に関わらず 0 となる．
- シフト演算：レジスタ Rd の値を，以下のようにシフトした値を Rd に格納し，条件コードを設定する．
  - SLL (shift left logical)... 左論理シフト．左シフト後，空いた部分に 0 を入れる
  - SLL (shift left logical)... 左論理シフト．左シフト後，空いた部分に 0 を入れる．
  - SLR (shift left rotate) ... 左循環シフト．左シフト後，空いた部分にシフトアウトされたビット列を入れる．
  - SRL (shift right logical) ... 右論理シフト．右シフト後，空いた部分に 0 を入れる
  - SRA (shift right arithmetic) ... 右算術シフト．右シフト後，空いた部分に符号ビットの値を入れる．

  シフト桁数は即値 d (0 ~ 15) である．また条件コード C には，シフト桁数が 0 の時または SLR では 0 が，それ以外では最後にシフトアウトされたビットの値が設定される．条件コード V は常に 0 が設定される．
- 入出力命令：

  - IN (input)... スイッチなどの機器から入力した値をレジスタ Rd に格納する.
  - OUT (output)... レジスタ Rs の値を 7SEG LED などの機器に出力する.
  - HLT (halt)...SIMPLE を停止させる.

- ロード/ストア命令: ソース／デスティネーションは，フィールド Ra で指定されたレジスタ Ra である．また実効アドレスはベースレジスタアドレス指定により，フィールド Rb で指定されたレジスタ Rb と，フィールド d を符号拡張した sign ext(d) を加算して求める.

- 即値ロード/無条件分岐命令:

  - LI ... 即値 sign ext(d) をレジスタ Rb に格納する.
  - B...d を符号拡張した値を変位として，PC 相対アドレス指定による分岐を行う.

- 条件分岐命令: フィールド cond で定められる分岐条件が成り立てば PC 相対アドレスによる分岐を行ない，成り立たなければ単に次の命令に移行する．各命令の分岐条件は以下の通り.

  - BE (branch on equal-to)... 条件コード Z が 1
  - BLT (branch on less-than) ... 条件コード S と V の XOR(S $\hat{}$ V) が 1
  - BLE (branch on less-than or equal-to)...Z または (S $\hat{}$ V) が 1
  - BNE (branch on not-equal-to) ... 条件コード Z が 0

1.2 設計目標

上の命令セットアーキテクチャに対して、動作するマルチサイクルプロセッサが Simple/B である。Simple/B を拡張し独自のプロセッサを作ることが課題において要求されているところである。我々の班がどのような拡張を加え Simple/B より優れているプロセッサを作るかについて、我々の班の設計目標を以下に箇条書きを用いて示す。

- 上の命令セットアーキテクチャに即値加算命令を加え、きちんと動作させる。

- マルチサイクルプロセッサではなく、5 段パイプライン化したプログラムにする。

- さらに高速化を図るために、さらに不成立分岐予測を用いる。また、それを実装する。
- 5 段パイプライン化するので、命令数＋ 4 サイクルで命令を完了、また、周波数も 20MHz より適宜増やしていきたい。

1.3 設計方針

要求仕様の通りに動く Simple/B の回路図が下の図 1 である。これを設

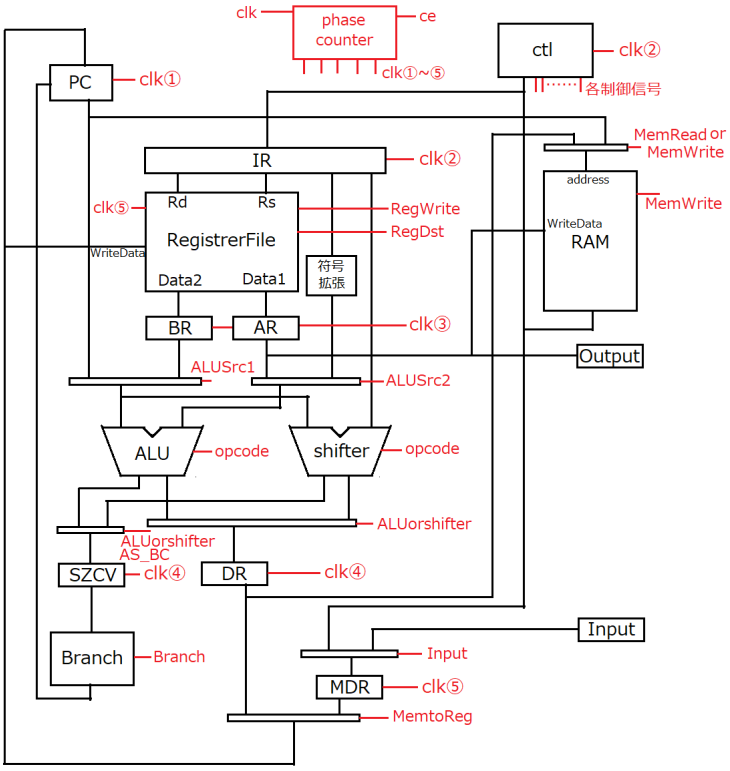

図 1: simple/B の全体図

計目標に合うようにするためには、フォワーディングユニット、ハザード検出ユニット、不成立分岐予測ユニット等を実装し、主記憶をハーバード方式に切り替える、周波数を上げる方法を探す、等のことをしなければいけないので、それを班員で手分をしボトムアップ形式で組み合わせていく。

2　高速化/並列処理の方式

設計目標、設計方針の項目において述べた通り、5 段のパイプライン化をする。また、不成立分岐予測等についても同様に行う。

3　性能/コストの予測

まず、性能についてだが性能は Simple/B よりも格段に上がっていると考えられる。Simple/B においては、マルチサイクルで処理していたものを 5 段のパイプラインで処理するので命令数が多ければ多いほどその差は歴然であろう。命令数を x とおくとマルチサイクルで処理すると 5x サイクル必要だが、5 段パイプライン化を行うと x ＋ 4 命令で済むので、x を十分に大きくするとほぼ 5 倍速くなるといえるだろう。　また、ハードウェア量についてだが新たにいくつかのモジュールをコンポーネントとして追加しなければならないこと、ハーバード方式に切り替えるためにメモリもわけることなどを考えると、2 倍まではいかずとも、1.3 倍程度にはなってしまうのではないかと思う。とても適当に見積もっているので妥当性はかなり低いと思う。　最後にソート速度コンテストの目標に関してだが、サイクル数 10 万前後、時間 1 ｍｓ以下を目標としたいと思う。

4　考察

計算機の構成の授業において、パイプライン化がとても速いと学んだので、ソート速度コンテストでいい成績を出すためにもこれは必須と考えた。また、命令を拡張する必要もあるので、一番簡単そうで一番必要そうな即値加算命令を追加することにした。これ以上の高速化 (ソート速度コンテストにおける) については、命令をどのソートで行うようにするかが大きな要因であると考えられる。現在、バブルソートで行ってみたが、とてもおそかった。よい成績を出すためには他のソートを行わなければいけないと考える。度数ソート、クイックソート等、より早いものを今後探して検証していきたい。