計算機科学実験３A

アーキテクチャ検討報告書

チーム４：竹田原俊介　Chung Mung Tim

レポート作成者：Chung Mung Tim

2022-05-12

この実験の目的は、SIMPLE（SIxteen-bit MicroProcessor for Laboratory Experiment）と呼ばれるコンピュータを設計することである．このレポートでは，私たちのプロセッサのアーキテクチャについて説明する．

第１章　要求仕様，設定目標，設計方針，特長

1.1 要求仕様

私たちが作ろうとしているプロセッサの仕様は以下である：

動作周波数：　/ MHz
面積（logic elements）: /

リソース：

- メモリサイズ：データは 16 ビット幅，アドレス空間は約 33KW
    - 命令メモリとデータメモリに分けられる
- 16 ビット x８語汎用レジスタ
- 12 ビットアドレス　プログラムカウンタ（program counter）

実現する機能

- 実行できる２オペレンド 16 ビット固定長命令
    - 演算命令（算術，論理，比較，移動，シフト演算）
        - 演算結果に基づく条件コード：S（sign），Z（zero），C（carry），V（overflow）
    - 入出力命令
    - 停止命令
    - ロード／ストア命令
    - 即値ロード命令
    - 分岐命令（無条件分岐命令，条件分岐命令）
- 外部から供給される信号
    - reset（リセット信号）
    - exec（起動／停止信号）

1.2 設定目標

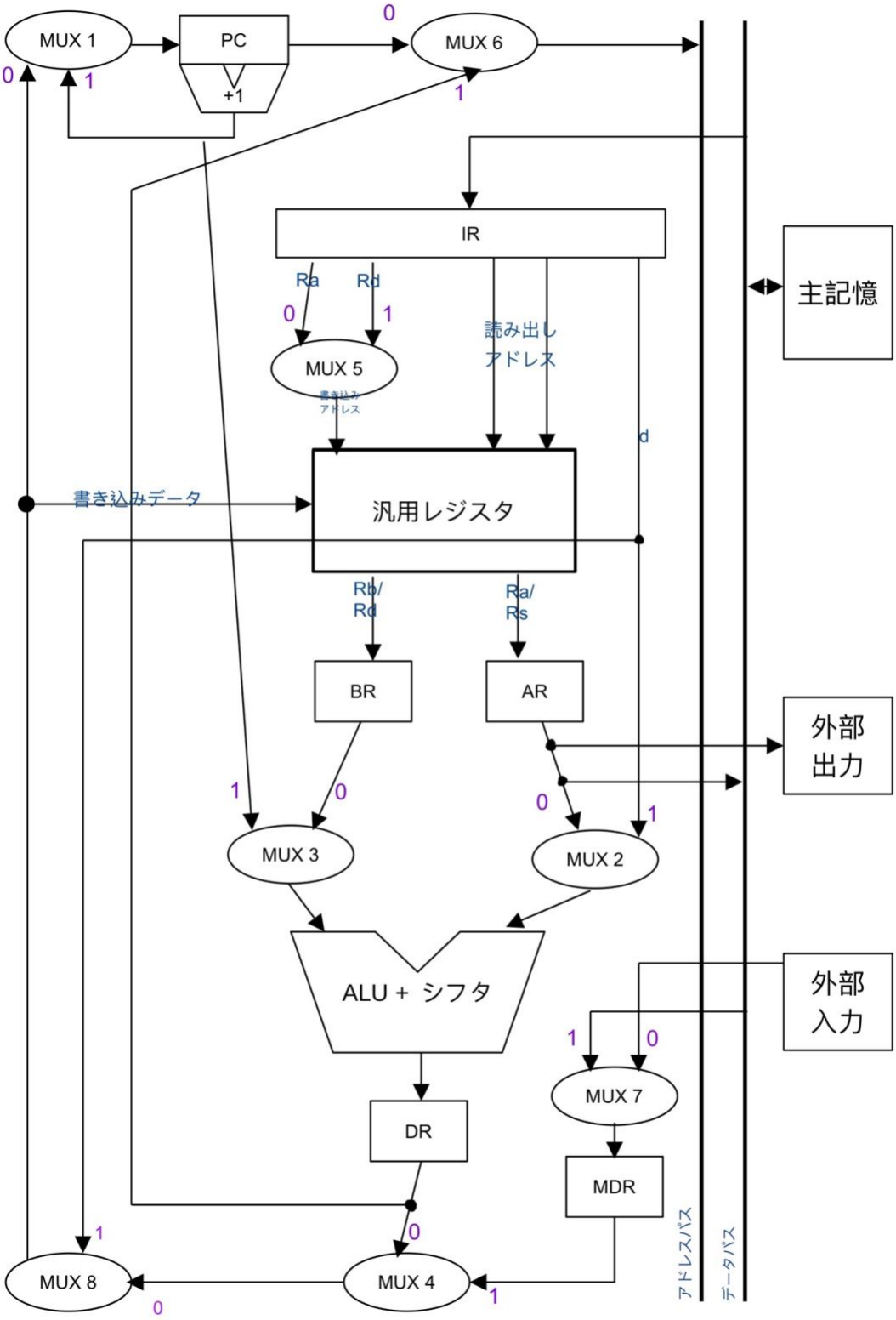
プロセッサ性能の目標数値は以下である：

- サイクル数：　～60000
- 周波数：～150MHz
- 面積（Logical elements）：～1500

1.3 設計方針

ボトムアップ方針を採用し，プロセッサをブロックに分け（図１を参照してください)，必要な部品を全て作ってから，制御回路を作り，最後にトップレベルモジュールで部品の入出力のポートを繋がる．各実装したい命令を試し、必要に応じてマルチプレクサやデータパスを追加していく．

図１：SIMPLE のブロック図



1.4 特長

- ALU とシフタを一体化すること
- プログラムカウンターと加算器を一体化すること
- 汎用レジスタは書き込みイネブラーが持つこと
- レジスタは出力イネブラーを持つこと
- 全てのレジスタは同じクロックを共用していること

第２章　高速化／並列処理の方式

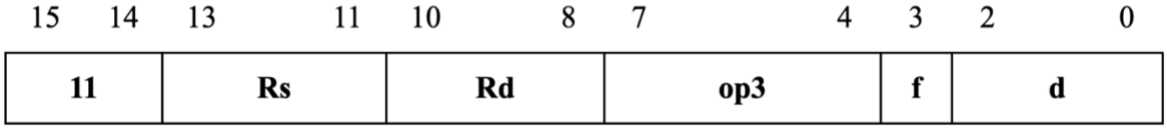
2.1 拡張命令

基本アーキテクチャ以上に，以下の拡張命令や改良も考えられる．

1. 即値オペランドの強化（設計済み・未実装）

基本アーキテクチャでは，演算命令のオペランドはいずれも汎用レジスタであり，レジスタに即値を加えることは少なくとも２命令を要する．r[Rd] + sign_ext(d)を実行できるようにし，そして r[Rd] + r[Rs]と r[Rd] + sign_ext(d)を切り替えられると望ましい。

そこで演算命令形式を以下のようになる．元々の d の中の１ビットをフラグ f に用いるようになった．



- f が 0 の場合，r[Rd] + r[Rs]を計算する．
- f が１の場合，r[Rd] + sign_ext(conc(Rs,d))を計算する．ここで，conc(Rs,d)は Rs フィールドの３ビットと d フィルドの３ビットを結合した値を表す．

このようにして得られるオペランドを sw(i, Rs, d)と表記すると，SIMPLE の演算命令は表 2.2.1 のようになる．こうすることで，即値オペランドを使った計算に必要な命令数を減らすことができる．

表 2.1.1　即値オペランドが強化された SIMPLE の演算命令

mnemonic		op3	function
ADD	Rd, Rs	0000	r[Rd] = r[Rd] + sw(i, Rs, d)
SUB	Rd, Rs	0001	r[Rd] = r[Rd] - sw(i, Rs, d)
AND	Rd, Rs	0010	r[Rd] = r[Rd] & sw(i, Rs, d)
OR	Rd, Rs	0011	r[Rd] = r[Rd] | sw(i, Rs, d)
XOR	Rd, Rs	0100	r[Rd] = r[Rd] ^ sw(i, Rs, d)
CMP	Rd, Rs	0101	r[Rd] - sw(i, Rs, d)
MOV	Rd, Rs	0110	r[Rd] = sw(i, Rs, d)
(reserved)		0111	/
SLL	Rd, d	1000	r[Rd] = shift_left_logical (r[Rd], d)
SLR	Rd, d	1001	r[Rd] = shift_left_rotate (r[Rd], d)
SRL	Rd, d	1010	r[Rd] = shift_right_logical (r[Rd], d)
SRA	Rd, d	1011	r[Rd] = shift_right_arithmetic (r[Rd], d)

2. 入出命令の強化（設計済み・未実装）

基本アーキテクチャでは，FPGA ボードからの入力として IN 命令，ボードへの出力として OUT 命令がある．これらの入出力命令の入出力対象は特定の 16 ビットの入力／出力に固定されている．しかし，ボードには入力／出力対象ともに 16 ビットを超える数があり，入力／出力ともそれらの中から選択して使える方が望ましい．そこで，入出力先を変更する拡張が考えられる．

特に，Power Medusa EC6S ボードの８桁 7SEG LED は 32 ビット分のデータを表示できる．2.1.2 章で挙げた命令フォマットのうちのフラグ f を用い，８桁の 7SEG LED の上位／下位の切り替えに使う．

- F が 0 の場合，7SEG LED の４上位を使って出力を表す.
- F が 1 の場合，7SEG LED の４下位を使って出力を表す

また，IN 命令と OUT 命令で未使用のフィールドを使って，入出力先にアドレスを与え，それを 2.1.2 章で定義した sw(i, Rs, d)で指定するように変更するのも考えられる．そこで，SIMPLE の入出力命令は表 2.1.2 に示すものとなる．

表 2.1.2　入出力命令が強化された SIMPLE の入出力命令

mnemonic		op3	function
IN	Rd	1100	r[Rd] = input[sw(i, Rs, d)]
OUT	Rs	1101	output[sw(i, Rs, d)] = r[Rs]

3. Branch Register (BR)と Branch And Link (BAL) 命令の追加（設計済み・未実装）

- BR（branch register）　　：関数呼び出しに使用される・レジスタ r[Rb]の値を分岐アドレスとして分岐する．
- BAL（branch and link）　：関数呼び出しに使用される・次の命令のアドレス PC+1 を復帰アドレスとしてリンクレジスタ r[Rb]に格納する．それと共に，PC 相対アドレス指定による分岐を行う．なお，分岐アドレスは PC+1 に d を符号拡張した値 sign_ext(d)を加算して求める．

表 2.1.3　SIMPLE の無条件分岐命令の追加

BR（未実装）	(Rb)	101	PC = r[Rb]
BAL（未実装）	Rb, d	110	r[Rb] = PC + 1 ; PC = PC + 1 + sign_ext(d)

4. 条件分岐命令の分岐条件を増やす（設計済み・未実装）

- BGE（branch on great than or equal to）　　：S ^ V が 0
- BGT（branch on greater than）　　：Z || (S ^ V)が 0
- BC（branch on carry）　　：C が 1
- BNC（branch on not carry）　　：C が 0

表 2.1.4　SIMPLE の条件分岐命令の追加

BGE（未実装）	d	100	if (!(S ^ V)) PC = PC + 1 + sign_ext(d)
BGT（未実装）	d	101	if (!(Z || (S ^ V))) PC = PC + 1 + sign_ext(d)
BC（未実装）	d	110	if (C) PC = PC + 1 + sign_ext(d)
BNC（未実装）	d	110	if (!C) PC = PC + 1 + sign_ext(d)

5. 条件分岐の１命令化（未設計・未実装）

基本アーキテクチャでは，条件分岐は，演算の結果によって設定される条件コードを利用している．この構成では基本的に条件分岐を行う前に比較命令を実行しなくてはならない．そこで，より高速的に実行するために，条件分岐と比較を１命令で行わせることを考えられる．

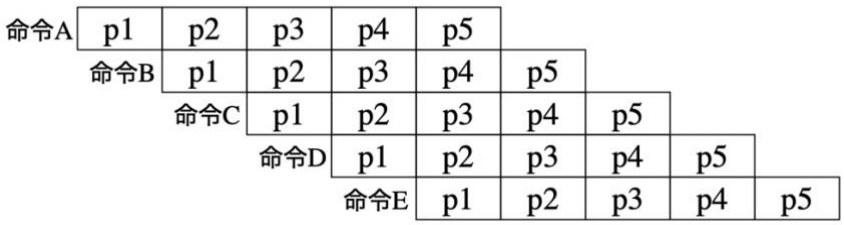
6. 複合演算命令の追加（未設計・未実装）

より効率的に実行するために，１つの命令で二つの演算を行う命令を実装することが考えられる．

2.2 パイプライン化と並列化

フェーズの並列実行による命令サイクルを短縮することができる．現時点でまだパイプライン化していないが，図 2.2 のような p1/p2/p3/p4/p5 の並列実行を目指す．

図 2.2　p1/p2/p3/p4/p5 の並列実行



第３章　性能／コストの予測

この設計はまだ実装に成功していないため，性能に関するデータを提供することはできない．

しかし，機能を拡張するとだいぶ改善が期待される．例えば，p1/p2/p3/p4/p5 の並列実行ができたら，５倍くらいの加速が期待されるだろう．また，命令数を減ること

ができる拡張命令ができたら，サイクル数が減り，面積も減るだろう．

第４章　考察

ボトムアップ設計方針を採用したが，この方針の最大の弱点は，設計を変更する（例えば，新しい部品を追加する，部品の順番を変える）たびに，制御回路全体をかなり作り直さなければならないことである．もしやり直すなら，まずダミーの部品を使って制御部で設計を確定した方がいいかもしれない．