

Group22 プロセッサ Ver.3.00

ユーザーマニュアル

平成 29 年度 計算機科学実験及演習 3A
3 回生前期学生実験 HW 最終報告

提出期限：6 月 16 日
提出日：6 月 16 日

第 22 班

1029277526 白石 竜也
1029272870 谷 勇輝

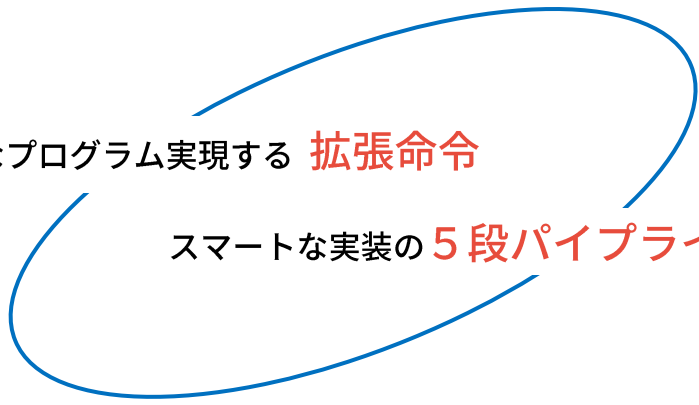
目次

1 概要.....	3
1.1 基本情報.....	3
1.2 設計情報.....	3
2 性能と特長.....	4
2.1 性能.....	4
2.2 特長.....	4
5 段パイプライン方式.....	4
ハーバード・アーキテクチャ.....	5
即値加算命令.....	5
関数呼出命令.....	5
静的分岐予測.....	5
3 命令セット・アーキテクチャ.....	6
3.1 命令セット.....	6
演算/入出力命令形式.....	7
ロード/ストア命令形式.....	8
即値ロード/無条件分岐命令形式.....	8
条件分岐命令形式.....	9
3.2 分岐判断.....	9
3.3 関数呼出.....	10
4 構造と動作.....	10
4.1 入出力構造.....	10
入力.....	10
出力.....	11
4.2 動作.....	12
演算命令 / 即値加算命令.....	12
入力命令.....	12
出力命令.....	12
ロード命令.....	12
ストア命令.....	12

即値ロード命令	12
分岐命令	13
関数呼出命令	13
戻り命令	13
4.3 内部構造	13
5 付録	15
5.1 テスト環境用モジュール	15
5.2 アセンブラ	15

1 概要

Group22 プロセッサ Ver.3.00 は、京都大学工学部情報学科の 2017 年度計算機科学実験及演習 3 において第 22 班が作成したソフトコア・マイクロプロセッサである。SIMPLE アーキテクチャをベースとした独自の拡張アーキテクチャを持つ 5 段パイプライン方式のプロセッサであり、FPGA 上で論理合成し CPU として利用することができる。



柔軟なプログラム実現する **拡張命令**

スマートな実装の **5 段パイプライン方式**

当プロセッサは、CPU として動作するのに最低限必要な命令セットに加え、プログラムを簡易・効率化する**拡張命令**を実装し、使用性の向上を図っている。また、パイプラインの各フェーズでの**機能分担**を深く吟味し、Group22 プロセッサオリジナルの設計を各所に取り入れた。この 2 つの基本方針の元 Group22 プロセッサは設計され、高速・高使用性を実現している。

1.1 基本情報

レジスタ幅：	16 bit
レジスタ：	0 番から 7 番まで計 8 つ (0 番レジスタは 0 レジスタとして働き、値は常に 0 となる)
メモリ：	2048 語 (1 語 16bit) × 2

1.2 設計情報

記述言語：	Verilog HDL 2001
-------	------------------

ターゲット FPGA： Altera Cyclone IV E (EP4CE30F23I7)

設計環境： Altera Quartus II 13.0sp1

開発者： 白石 竜也 / 谷 勇輝

開発期間： 2017 年 4 月 20 日
～ 2017 年 6 月 2 日

2 性能と特長

2.1 性能

最大動作周波数： 73 MHz

FPGA ボード⁽¹⁾上での周波数： 130MHz

回路規模： 1592 論理素子

基本プログラムによる性能の目安

- 1024 個のバブルソート
5,503,666 サイクル / 44.0293 ms
- 1024 個の単純クイックソート
94,002 サイクル / 0.7520 ms
- 1024 個の改良クイックソート(+挿入ソート)
65,812 サイクル / 0.5062 ms

2.2 特長

5 段パイプライン方式

5 つのフェーズが並列して動作する **5 段パイプライン方式** である。**フォワードフィードバック**による高速化の他、各フェーズに役割をうまく**分担**することで最適な動作を実現し、高速化を追及した。最終 WB フェーズについては**半クロック**で動作し早いタイミングでレジスタ書込みを行う設計になっており、データハザード回避による高速化を実現している。

ハーバード・アーキテクチャ

命令用メモリとデータ用メモリを分離する**ハーバード・アーキテクチャ**を採用している。IF フェーズと MA フェーズが別々のメモリを保持し、同時に動作するためメモリアクセスの待機による時間遅延はない。

即値加算命令

SIMPLE アーキテクチャの命令セットに加え、即値加算命令 **ADDI** が使用できる。ループ内で使用する変数等の 1 加算や 1 減算等に重宝し、基本プログラムであるクイックソートでは約 10% のサイクル数削減が達成できた。

関数呼出命令

SIMPLE アーキテクチャの命令セットに加え、関数呼出命令 **JAL** と、戻り命令 **JR** が使用できる。関数としてまとめて記述された命令列に即値ジャンプを使ってどこからでも移り、戻ることができる。また、ジャンプの際に**レジスタの退避**が行われるので、関数先では元のレジスタを気にすることなく動作ができる。この機能を利用して**レジスタの擬似拡張**を行うこともできる。

静的分岐予測

分岐命令時に、分岐判断を待たず次の命令を投機実行する**静的分岐予測**を実装している。分岐は常に成立しない方向に予測され、予測が外れた際には投機実行した命令を破棄し即座に分岐先命令にシフトする。**分岐判断は EX フェーズに繰り上げて**早い段階で行うため、分岐予測が外れた際に発生するストールは 1 クロックのみに抑えられている。

3 命令セット・アーキテクチャ

3.1 命令セット

命令セットは 1) 演算/入出力命令形式、2) ロード/ストア命令形式、3) 即値ロード/無条件分岐命令形式、4) 条件分岐命令形式の 4 つの形式によって 16bit 二進数で表現される。

具体的な命令セットは以下の表 1 に示す通りである。命令コードはそれぞれの形式に従い、図 1～図 3 の通りに定められている。

表 1 命令セット一覧

命令	オペランド	動作
ADD	Rd, Rs	加算を行う
SUB	Rd, Rs	減算を行う
AND	Rd, Rs	bit 毎に論理積演算を行う
OR	Rd, Rs	bit 毎に論理和演算を行う
XOR	Rd, Rs	bit 毎に排他的論理和演算を行う
CMP	Rd, Rs	比較を行い、分岐用条件コードをセットする
MOV	Rd, Rs	Rs を Rd にコピーし格納する
SLL	Rd, d	左論理シフトを行う
SLR	Rd, d	左循環シフトを行う
SRL	Rd, d	右論理シフトを行う
SRA	Rd, d	右算術シフトを行う
IN	Rd	外部入力を取り込む
OUT	Rs	外部出力を更新する
HLT		プロセッサを停止する
LD	Ra, d(Rb)	メモリ読み込みを行う
ST	Ra, d(Rb)	メモリ書き込みを行う
LI	Rb, d	即値をレジスタにロードする
ADDI	Rb, d	即値を加算する

BE	d	比較の結果、等しければ分岐を行う
BLT	d	比較の結果、小なりの関係であれば分岐を行う
BLE	d	比較の結果、小なりイコールの関係であれば分岐を行う
BNE	d	比較の結果、等しくなければ分岐を行う
B	d	無条件分岐を行う
JAL	d	レジスタを一時退避し、関数呼出を行う
JR	d	レジスタを復帰し、関数の呼び出し元に戻る

演算/入出力命令形式

			15 14 13	11 10	8 7	4 3	0
			11	Rs	Rd	op3	d
mnemonic		op3	function				
ADD	Rd, Rs	0000	$r[Rd] = r[Rd] + r[Rs]$				
SUB	Rd, Rs	0001	$r[Rd] = r[Rd] - r[Rs]$				
AND	Rd, Rs	0010	$r[Rd] = r[Rd] \& r[Rs]$				
OR	Rd, Rs	0011	$r[Rd] = r[Rd] r[Rs]$				
XOR	Rd, Rs	0100	$r[Rd] = r[Rd] \wedge r[Rs]$				
CMP	Rd, Rs	0101	$r[Rd] - r[Rs]$				
MOV	Rd, Rs	0110	$r[Rd] = r[Rs]$				
(reserved)		0111					
SLL	Rd, d	1000	$r[Rd] = \text{shift_left_logical}(r[Rd], d)$				
SLR	Rd, d	1001	$r[Rd] = \text{shift_left_rotate}(r[Rd], d)$				
SRL	Rd, d	1010	$r[Rd] = \text{shift_right_logical}(r[Rd], d)$				
SRA	Rd, d	1011	$r[Rd] = \text{shift_right_arithmetic}(r[Rd], d)$				
IN	Rd	1100	$r[Rd] = \text{input}$				
OUT	Rs	1101	$\text{output} = r[Rs]$				
(reserved)		1110					
HLT		1111	$\text{halt}()$				

図 1 演算/入出力命令形式

ロード/ストア命令形式

15 14 13 11 10 8 7 0			
op1 Ra Rb d			
mnemonic	op1	function	
LD Ra,d(Rb)	00	$r[Ra] = *(r[Rb] + \text{sign_ext}(d))$	
ST Ra,d(Rb)	01	$*(r[Rb] + \text{sign_ext}(d)) = r[Ra]$	

図 2 ロード/ストア命令形式

即値ロード/無条件分岐命令形式

15 14 13 11 10 8 7 0			
10	op2	Rb	d
mnemonic	op2	function	
LI Rb, d	000	$r[Rb] = \text{sign_ext}(d)$	
ADDI Rb, d	001	$r[Rb] = r[Rb] + \text{sign_ext}(d)$	
(reserved)	010		
(reserved)	011		
(reserved)	100		
(reserved)	101		
(reserved)	110		
条件分岐命令	111	表 2.2 参照	

図 3 即値ロード/無条件分岐命令形式

条件分岐命令形式

15	14	13	11	10	8	7	0
10	111	cond	d				
mnemonic			cond	function			
BE	d		000	if (Z) PC = PC + 1 + sign_ext(d)			
BLT	d		001	if (S^V) PC = PC + 1 + sign_ext(d)			
BLE	d		010	if (Z S^V) PC = PC + 1 + sign_ext(d)			
BNE	d		011	if (!Z) PC = PC + 1 + sign_ext(d)			
B	d		100	PC = PC + 1 + sign_ext(d)			
JAL	d		101	r[0] = PC + 1, PC = zero_ext(d)			
JR	d		110	PC = r[0]			
(reserved)			111				

図 4 条件命令形式

3.2 分岐判断

分岐は、分岐命令の直前の命令が設定する**条件コード**によって判断される。4 種類の条件コードを以下に示す。

- S 負ならば 1
- Z ゼロならば 1
- C 桁上げがあれば 1
- V 演算結果が符号付 16bit で表せる範囲を超えた場合 1

比較演算命令 CMP は内部的には減算を行い条件コードを設定する。シフトの条件コード C は、シフト桁数が 0 の時、または SLR の時は 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。

条件分岐命令が分岐判断する場合は、以下の通りである。

- BE Z が 1

- BLT S XOR V
- BLE Z OR (S XOR U)
- BNE NOT Z

3.3 関数呼出

関数呼出命令 JAL と戻り命令 JR については、レジスタの一時退避が行われる。

0 番～5 番までの 6 つのレジスタはローカルレジスタであり、関数呼出を行うと値の退避が実行され、戻り命令で復帰させることができる。一方 6 番、7 番のレジスタはグローバルレジスタであり、関数呼出命令、戻り命令によって値の退避は行われない。

関数呼出直後、レジスタの値は呼び出し前の値と同一である。従って関数の引値としては 0 番～7 番の全てのレジスタを使用することができる。戻り命令では、6 番、7 番以外のレジスタの値は関数呼出前の値に戻される。従って関数が返り値を持つ場合、それらは 6 番、7 番のレジスタに格納する必要がある。

4 構造と動作

4.1 入出力構造

入力

以下の表 2 に示す 4 つの入力が存在し、適切な信号を外部から入力して使用する。

表 2 プロセッサの入力

入力信号名	bit 幅	役割
clock	1	プロセッサを動作させるメインクロック
reset	1	プロセッサをリセットする信号
exec	1	プロセッサを一時停止させる信号
in	16	in 命令によって読み込む 16bit 整数

出力

以下の表 3 に示す 2 つの出力が存在し、外部と適切な接続を行い使用する。

表 3 プロセッサの出力

出力信号名	bit 幅	役割
halt	1	プロセッサが動作終了したことを示す信号
out	16	out 命令によって出力される 16bit 整数

また、表 4 に示す 12 の内部状態確認用出力も用意されており、必要に応じて使用することができる。

表 4 プロセッサの内部状態確認用出力

出力信号名	bit 幅	役割
PC_next	16	ID モジュールに入力される PC カウンタ+1 の値
instruction	16	ID モジュールに入力される命令コード
control	6	EX モジュールに入力される制御コード
Rs_Ra	16	EX モジュールに入力される第 1 演算数
Rd_Rb	16	EX モジュールに入力される第 2 演算数
stall	1	分岐ハザードの発生通知信号
mem_in	16	MA モジュールに入力されるメモリに書き込まれるデータ
result	16	MA モジュールに入力される演算結果
LDresult	16	WB モジュールに入力されるメモリ読み出し結果
WBdata	16	WB フェーズでレジスタに書き込まれるデータ
WBadd	16	WB フェーズで書き込みを行うレジスタアドレス
RegW	1	WB フェーズのレジスタ書き込み制御信号

4.2 動作

命令に対するプロセッサの主な動作は以下の通りである。

演算命令 / 即値加算命令

IF (命令フェッチ) → ID (命令デコード/レジスタ読出し) → フォワーディング、データハザード対処 → EX (演算) → (MA) → WB(レジスタ書込み) → ID

入力命令

IF (命令フェッチ) → ID (命令デコード/入力取得) → (EX) → (MA) → WB(レジスタ書込み) → ID

出力命令

IF (命令フェッチ) → ID (命令デコード/レジスタ読出し) → Out (出力更新)

ロード命令

IF (命令フェッチ) → ID (命令デコード/レジスタ読出し) → フォワーディング、データハザード対処 → EX (アドレス演算) → MA (メモリ読出し) → WB(レジスタ書込み) → ID

ストア命令

IF (命令フェッチ) → ID (命令デコード/レジスタ読出し) → フォワーディング、データハザード対処 → EX (アドレス演算) → MA (メモリ書込み)

即値ロード命令

IF (命令フェッチ) → ID (命令デコード/レジスタ取得) → (EX) → (MA) → WB(レジスタ書込み) → ID

分岐命令

IF (命令フェッチ) → ID (命令デコード/レジスタ取得) → EX (分岐判断) →
IF(PC 更新)

関数呼出命令

IF (命令フェッチ) → ID (命令デコード/レジスタ取得) → EX (分岐) (→
IF(PC 更新)) → (MA) → (WB) → ID(レジスタ退避)

戻り命令

IF (命令フェッチ) → ID (命令デコード/レジスタ取得/レジスタ復帰) → EX (分岐)

4.3 内部構造

Group22 プロセッサは、5 段パイプラインの各フェーズを担当する次の 5 つの主要モジュールを持つ。

- | | |
|------------|------------------|
| ◆ IF モジュール | 命令フェッチを行う |
| ◆ ID モジュール | 命令の解釈、レジスタ読出しを行う |
| ◆ EX モジュール | 各種演算、分岐判断を行う |
| ◆ MA モジュール | メモリ管理を行う |
| ◆ WB モジュール | レジスタ書込みを行う |

また、パイプラインの動作を補助する次の 4 つの補助モジュールを持つ。

- | | |
|-------------------------|------------------|
| ◆ Forwarding モジュール | フォワーディング処理を行う |
| ◆ HazardDetection モジュール | データハザード処理を行う |
| ◆ Stop モジュール | 外部からの一時停止信号を管理する |
| ◆ Out モジュール | 外部出力を管理する |

Group22 プロセッサは以上の 9 つのモジュールで構築される。図 5 にブロック図を、図 ?? にトップレベルの回路の概観を示す。

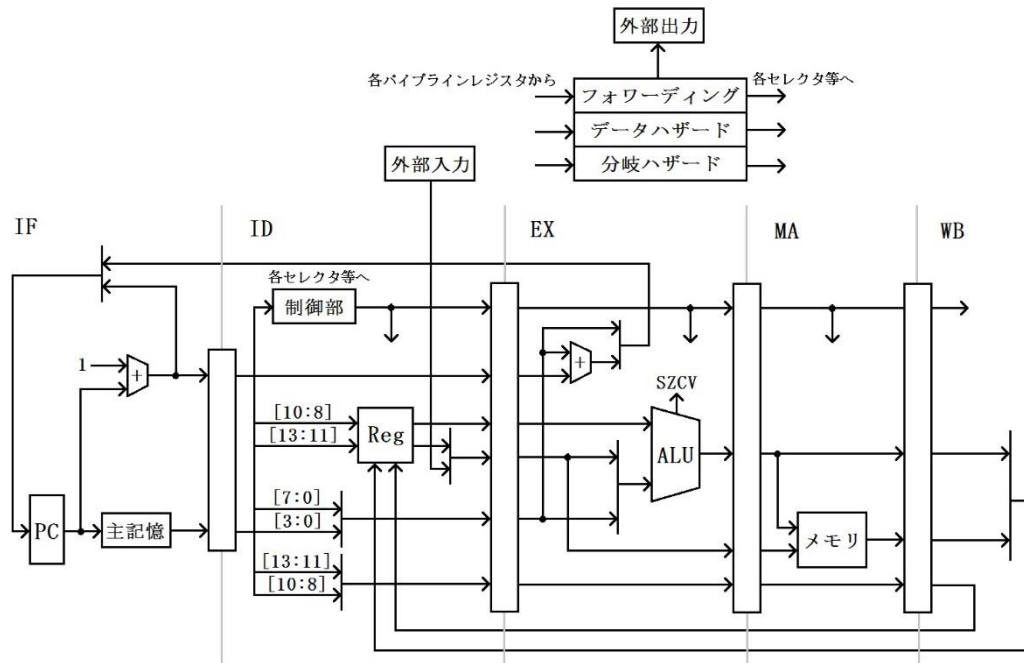


図 6 プロセッサ (ブロック図)

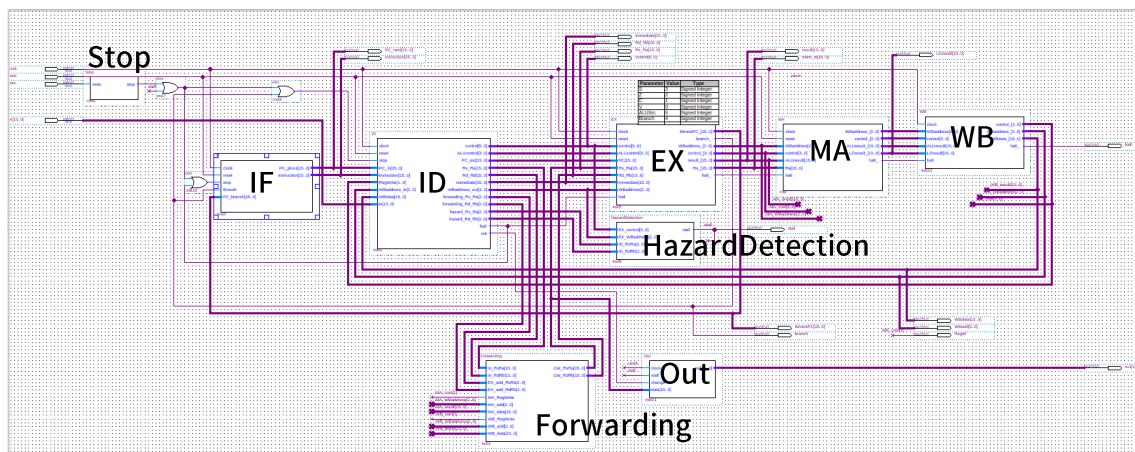


図 5 トップレベル概観

5 付録

5.1 テスト環境用モジュール

付録として、FPGA ボード PowerMedusa MU500-RX/RK 用の入出力・表示支援機能付きテスト環境用モジュール **TestEnvironment** が利用可能である。プロセッサの入出力を各種用意された配線に接続することにより、簡単にボードに表示を行い、プロセッサの動作を確認することができる。

5.2 アセンブラ

Group22 プロセッサ命令セットに準拠したバイトコードを出力する専用アセンブラ **CUIAssembler** が利用できる。CUIAssembler.java ファイルが存在するディレクトリで以下のコマンドを入力すると起動し、指示に従ってファイル選択を行うことでアセンブリコードをバイトコードに変換できる。

```
> java CUIAssembler
```