卒業論文

ハード/ソフト協調学習システム
を用いたプロセッサ設計と評価

氏　　名　：　井手　純一
学籍番号　：　2260040008-1
担当教員　：　山崎　勝弘　教授

提 出 日　：　2008 年 2 月 20 日

立命館大学　理工学部　電子情報デザイン学科

内容概要

本論文では、ハード/ソフト協調学習システムを用いて、ハードウェアとソフトウェアの両方の知識を得ることを目的とし、Verilog　HDL によるシングルサイクルプロセッサとマルチサイクルプロセッサを設計した。設計したプロセッサは SARIS（Simple Architecture Referring to Instruction of SOAR）と名づけた。また設計したプロセッサをプロセッサデバッガと接続し、FPGA ボード上で検証した。

本研究では、ハード/ソフト協調学習システムを用いて実際にプロセッサを設計し、ハード/ソフト協調学習システムがプロセッサ設計において、有効であるかを評価することを目的とし、さらにはシステム全体を評価することを目的としている。

目次

図目次

表目次

1　はじめに

近年の急速な半導体製造技術により、LSI の小型化、軽量化と高速化、そして消費電力化が可能となった。携帯電話、自動車、カーナビゲーションシステム、炊飯器、信号機、エレベーター、自動販売機、デジタルカメラ、テレビ、ゲーム機、複写機などに挙げられる組み込み機器は、いずれもハードウェアとソフトウェアから構成される。これら組み込み機器の普及は、これからも広がっていくことが確実視されている。そして要求される仕様は年々大規模かつ複雑になり、実装的には小型、低消費電力化が進み、製品のライフサイクルは縮小し、開発期間の短縮化が求められている。このように、高集積システムＬＳＩ技術の進化の中、システムＬＳＩへ求められる機能は多様化しており、ハードとソフト両方の知識に加え、プロセッサにおける命令セットとマイクロアーキテクチャの知識が必要不可欠である。

半導体製造技術の進歩によって、大規模で複雑なシステムが 1 つのチップ上に構成できるようになったこと、つまり LSI の設計と検証がシステム全体の設計と検証と等価になった。また、組み込み機器のライフサイクルが短くなり，開発期間を短くすることがますます重要となっていることにより、ハード/ソフト協調設計が開発期間短縮に大きく影響する。このような近年の LSI 開発技術はハードとソフトに密接な関係があり、ハードウェアとソフトウェアの両方の知識を習得するためにも、早期の教育が必要である。

以上の背景から、大学の教育でもハードウェアとソフトウェアの関係を意識した学習が必要である。そこで本研究室では、ハード/ソフト協調学習システムを考案し、開発を進めてきた[2-9]。ハード/ソフト協調学習システムとは、プロセッサを通してハードとソフトの両方の学習を進めていくことを目的としたシステムである。本研究の目的は、ハード/ソフト協調学習システムを利用し、実際にプロセッサ設計を行うことによって、どの程度このシステムがハードウェア学習とソフトウェア学習において有効であるのか評価することである。またシステムを評価することによって、システムの改善、または今後のシステム拡張について検討していくことを目的として本研究を行った。

本研究では、ハード/ソフト協調学習システムを用いて、Verilog HDL によるシングルサイクルプロセッサ、マルチサイクルプロセッサの設計を行う。実際にプロセッサ設計を行うことによりシングル、マルチの 2 種類のプロセッサアーキテクチャを理解する。次に、設計したプロセッサを HDL シミュレータにより検証する。HDL シミュレータには、Xilinx 社の Modelsim を使用する。そして設計したプロセッサをプロセッサデバッガと接続し、論理合成を行い、FPGA ボード上に実装し検証する。論理合成には Xilinx 社の ISE 9.1i を使用している。以上の流れから、設計したプロセッサの評価と、さらにはハード/ソフト協調学習システムについての評価を行う。

本論文では、第 2 章でハードソフト協調学習システムについての詳細を説明する。第 3 章では設計した SARIS プロセッサについて、また SARIS の命令セットアーキテクチャについて説明する。第 4 章ではシングルサイクルによる設計と評価、第 5 章でマルチサイク

ルによる設計と評価について説明する。そして第 6 章にハード/ソフト協調学習システムの評価について述べる。

2　ハード/ソフト協調学習システム

2.1　　システム概要

ハード/ソフト協調学習システムとは、プロセッサを通してハードウェアとソフトウェアの両方の知識を学習していく為に考案されたシステムである。

ソフトウェアを学習する面では、アーキテクチャが可変な命令セットシミュレータ（MONI 仮想シミュレータ）を用いてプロセッサのアーキテクチャの仕組みの理解し、アセンブリ言語で書かれたプログラムを評価する。MONI とは、本研究室で MIPS のサブセットとして定義した教育用マイクロプロセッサである。ハードウェアを学習する面では、シミュレータで理解したプロセッサの知識を基に、HDL によるプロセッサ設計を行う。そして学習者が設計したプロセッサを検証、評価することによってプロセッサ設計能力を習得する。次に、ハードウェア学習の際に使用するプロセッサ設計支援ツールについて説明する。命令セット定義ツール、汎用アセンブラ、汎用シミュレータにより、命令セットを独自に定義することができる。またプロセッサモニタとプロセッサデバッガは、学習者が設計したプロセッサを FPGA ボード上で検証する際に使用する。これらのプロセッサ設計支援ツールを使用し、ハードとソフトの両方の学習を進めていくことがこのシステムの目的である。

2.2　　学習体系

図 1 に、ハード/ソフト協調学習システムについての学習体系を示す。ソフトウェア学習の流れは、学習者自身が用意したアセンブリプログラムを MONI 仮想シミュレータ上でシミュレーションを行う。MONI 仮想シミュレータでは、アーキテクチャを単一サイクル、マルチサイクル、パイプライン、スーパースカラの４つが選択可能である。プログラムの命令を実行すると、その命令に対するプロデッサのデータパスが確認できるので、これにより学習者はアセンブリプログラミング技術と MONI プロセッサの構造や動作の学習を行うことができる。次に、プロセッサ設計支援ツールの命令セット定義ツールを用いて、学習者の考えた命令セットを定義し、その出力ファイルを用いて汎用アセンブラと汎用シミュレータを使用する。また汎用アセンブラの出力ファイルは、ハードウェア学習でのプロセッサ検証の際に使用する。学習者がこの３つのプロセッサ設計支援ツールを使用することにより、プロセッサにおける命令セットアーキテクチャを学習することができる。ハードウェア学習の流れは、実際に HDL を用いて MONI プロセッサの設計、またはオリジナルプロセッサの設計を行う。次に、設計したプロセッサを HDL シミュレータによりシミュレーション検証を行い、それから FPGA ボード上に実装し、評価する。FPGA ボード上で検証する際、設計したプロセッサをプロセッサデバッガと接続し、プロセッサモニタを用いてデータを送受信することで検証を行う。動作検証にはソフトウェア学習で作成したプログラムを使用する。このようにしてプロセッサ設計能力の習得、またハードウェア特有の性質である遅延や設計規模などを考慮したプロセッサの設計手法を学習することができ

る。以上の流れから、ソフトウェアとハードウェアの学習を行う。

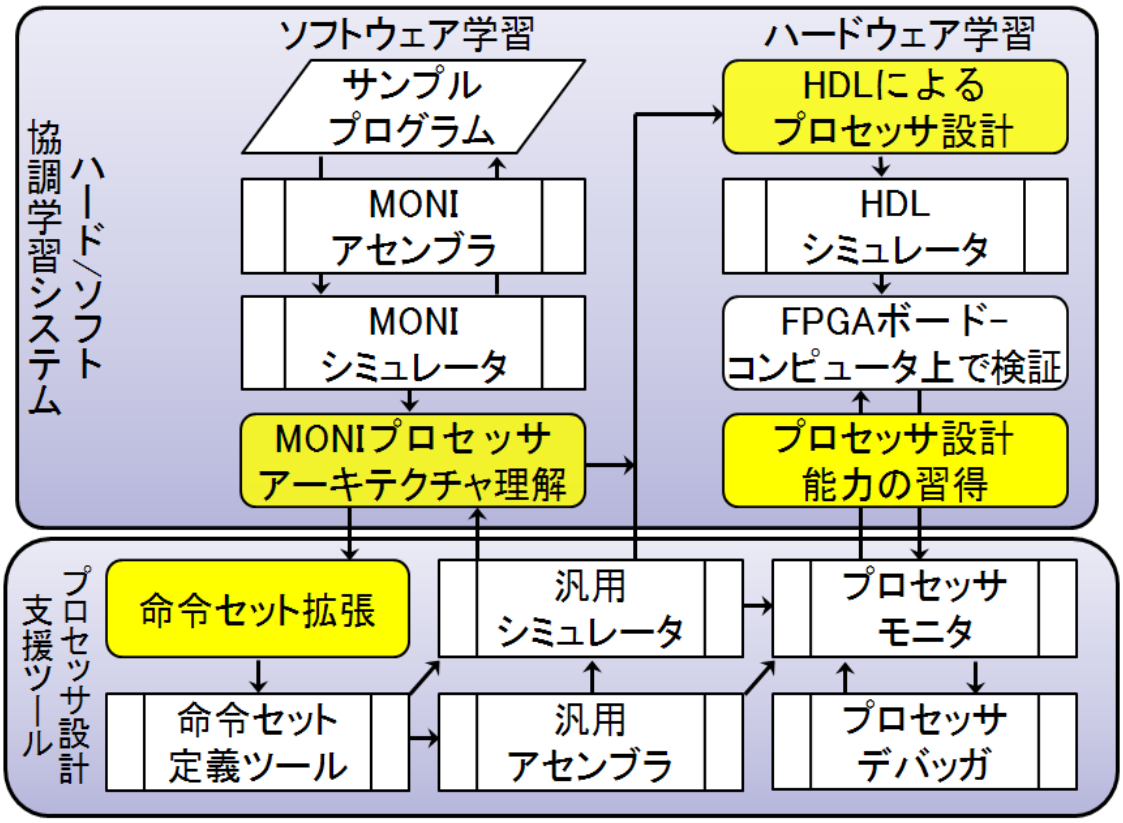

図 1：ハードソフト協調学習システムの学習体系

3 SARIS プロセッサ

3.1 設計思想

本研究では、ハード/ソフト協調学習システムがプロセッサ設計において有効であるかを評価するために、シングルサイクルとマルチサイクルの 2 つのマイクロプロセッサを設計した。設計したプロセッサは SOAR プロセッサの命令セットを参考にし、高速化のためできる限り単純な構成にしているので、SARIS(Simple Architecture Referring of Instruction of SOAR)と名付けた。SOAR プロセッサとは、本研究室を卒業された難波翔一郎氏によって設計された単一サイクルマイクロプロセッサである[3]。SARIS プロセッサには以下のような特徴がある。

・命令長 16 ビット固定。
・3 オペランド命令方式。
・全 22 命令。
・4 つの命令形式。
・フィールド op の上位 2 ビットを形式別に使用。
・56 命令まで拡張可能。

3.2 命令セットアーキテクチャ

SARIS には、JUMP 形式（J 形式）、Register 形式（R 形式）、Immediate5 形式（I5 形式）、Immediate8 形式（I8 形式）の 4 つの命令形式を用意した。J 形式には無条件分岐命令の JUMP、空白命令の NOP、プログラム終了命令となる HALT を定義している。R 形式にはレジスタ間の演算を行う命令を定義している。I5 形式にはレジスタ値と即値演算を行う命令を定義している。I8 形式には条件分岐命令とメモリ・レジスタへのデータ転送命令を定義している。図 2 に SARIS の命令形式を示す。

bit / format	5	2	3	3	3
J	op	immediate			
R	op	f n	r s	r t	r d
I5	op	immediate		r t	r d
I8	op	immediate			(r d)

図 2：SARIS の命令形式

op フィールドの上位 2 ビットを形式別に使用しているので、残り 3 ビットで命令の種類を定義している。上位 2 ビットで形式分けを行っているのは、命令デコードの効率化を図るためである。J 形式なら上位 2 ビットを 00、R 形式なら 01、I5 形式なら 10、I8 形式なら 11 と、このように使用している。56 命令まで拡張可能だが、上位 2 ビットを形式別に使用しているため、J 形式、I5 形式、I8 形式は残り 3 ビットなので 8 種類までとなっている。R 形式においては、フィールド fn でさらに命令を詳細に識別しているため、32 種類まで拡張が行える。表 1 に各フィールドの意味を、表 2 に命令セット一覧を示す。

表 1：命令フィールドの意味

フィールド	意味	bit 幅	用途
op	Operation	5	命令を識別
fn	Function	2	R 形式の命令を詳細に識別
rs	Source　Register	3	演算元レジスタ
rt	Target　Register	3	演算先レジスタ
rd	Destination　Register	3	演算結果を格納するレジスタ
imm	Immediate	5~11	即値

表 2：SARIS 命令セット一覧

	命令	tp	op	fn	rs	rt	rd	命令動作	意味
J	NOP	0 0	0 0 0	××××××××				NOP	PC++
	HALT	0 0	0 0 1	××××××××				HALT	exit
	JUMP	0 0	0 1 0	imm				JUMP imm	Go to (PC = imm)
R	ADD	0 1	0 0 0	0 0	rs	rt	rd	ADD rd rt rs	rd = rt + rs
	SUB	0 1	0 0 0	0 1	rs	rt	rd	SUB rd rt rs	rd = rt – rs
	SLT	0 1	0 0 0	1 0	rs	rt	rd	SLT rd rt rs	(if rt < rs) rd = 1
	SGT	0 1	0 0 0	1 1	rs	rt	rd	SGT rd rt rs	(if rt > rs) rd = 1
	SLE	0 1	0 0 1	0 0	rs	rt	rd	SLE rd rt rs	(if rt ≦ rs) rd = 1
	SGE	0 1	0 0 1	0 1	rs	rt	rd	SGE rd rt rs	(if rt ≧ rs) rd = 1
	SEQ	0 1	0 0 1	1 0	rs	rt	rd	SEQ rd rt rs	(if rt = rs) rd = 1
I5	ADDI	1 0	0 0 0	imm		rt	rd	ADD rd rt imm	rd = rt + imm
	SUBI	1 0	0 0 1	imm		rt	rd	SUB rd rt imm	rd = rt – imm
	SLTI	1 0	0 1 0	imm		rt	rd	SLTI rd rt imm	(if rt < imm) rd = 1
	SGTI	1 0	0 1 1	imm		rt	rd	SGTI rd rt imm	(if rt > imm) rd = 1
	SLEI	1 0	1 0 0	imm		rt	rd	SLEI rd rt imm	(if rt≦imm) rd = 1
	SGEI	1 0	1 0 1	imm		rt	rd	SGEI rd rt imm	(if rt≧imm) rd = 1
	SEQI	1 0	1 1 0	imm		rt	rd	SEQI rd rt imm	(if rt = imm) rd = 1
I8	LDLI	1 1	0 0 0	imm			rt	LDLI rt imm	rt[7:0] ← imm[7:0]
	BEQZ	1 1	0 0 1	imm			rt	BEQZ rt imm	(if rt = 0) PC = imm
	BNEZ	1 1	0 1 0	imm			rt	BNEZ rt imm	(if rt≠0)PC = imm
	LD	1 1	0 1 1	imm			rd	LD rd imm	rd = DM[imm]
	ST	1 1	1 0 0	imm			rs	ST imm rs	DM[imm] = rs

4　シングルサイクルプロセッサの設計と検証

4.1　アーキテクチャ

図 3 に、シングルサイクルのデータパスを示す。

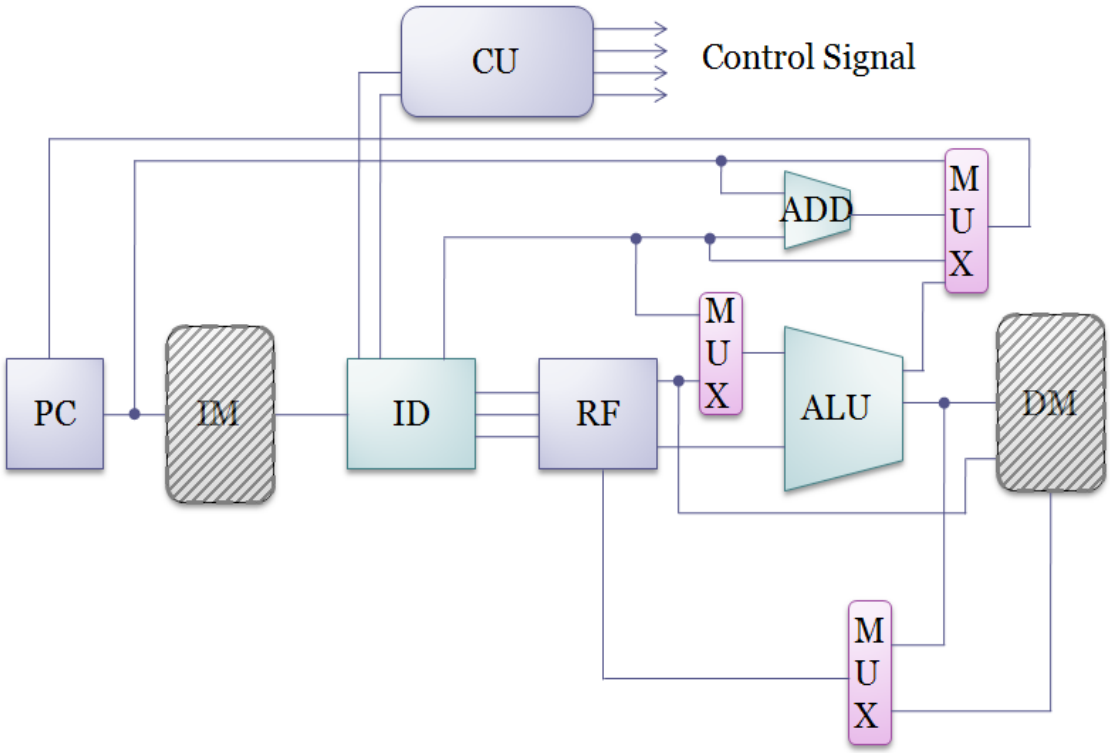

図 3：シングルサイクルデータパス

設計したモジュールは、IM と DM 以外の 7 種のモジュールである。それぞれの動作と意味を以下に示す。

- PC　…　Program Counter
  ⇒命令メモリへアドレスを渡す。
- ID　…　Instruction Decoder
  ⇒命令メモリから読み出した命令を解読し、命令により op、fn、rs、rt、rd、imm を出力。
- RF　…　Register File
  ⇒2 つのソースレジスタと、1 つのデスティネーションレジスタを指定できる。ソースレジスタは、アドレスを指定すると、クロックに非同期で出力する。デスティネーションレジスタは、クロックに同期して指定したアドレスにデータを書き込む。
- ALU　…　Arithmetic Logic Unit
  ⇒算術演算、分岐判定などを行う。分岐判定信号は、プログラムカウンタへデ

ータを送るマルチプレクサに送る。

- ADD

  ⇒分岐先のアドレスの計算を行う。

- MUX　…　Multiplexer

  ⇒複数の入力から、制御信号によって出力を選択。

- CU　…　Control Unit

  ⇒ID から Opecode や Function を受け取り、各モジュールへ制御信号を送る。（ADD には送らない）

SARIS シングルサイクル設計において、命令形式別によるデータパスを図に示しながら以下に解説する。

(1)R 形式

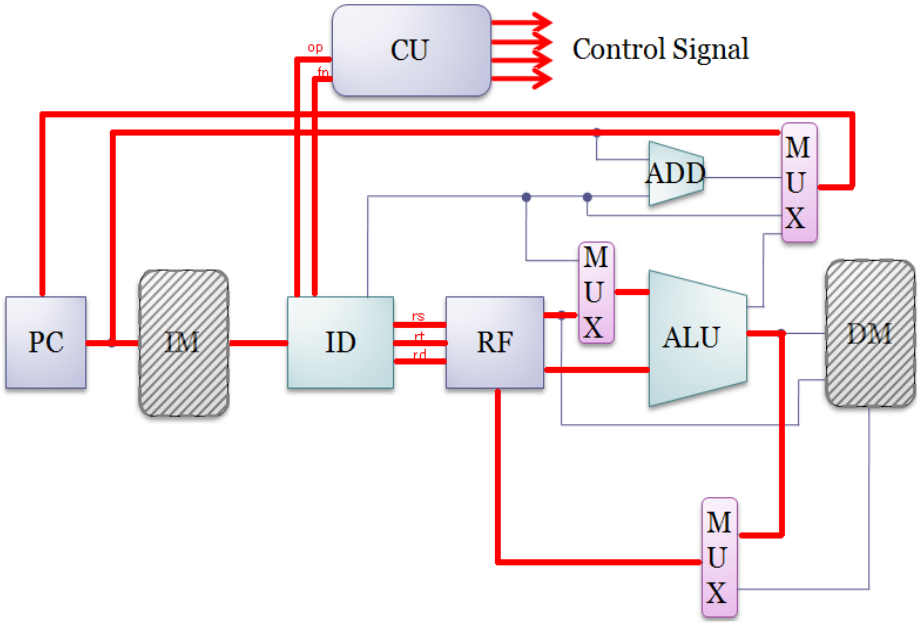

図 4： R 形式の実行過程

R 形式では、ID で IM から受け取った命令の識別を行い、CU に op と fn のデータを送る。また演算データのアドレス rs と rt、演算結果を格納するアドレス rd を RF に出力する。ALU は、CU から演算命令の種類を受け取り、rs データと rt データの演算を行い出力する。PC 値は PC の出力とつながっている MUX で PC＋１を行う。

(2)I5 形式

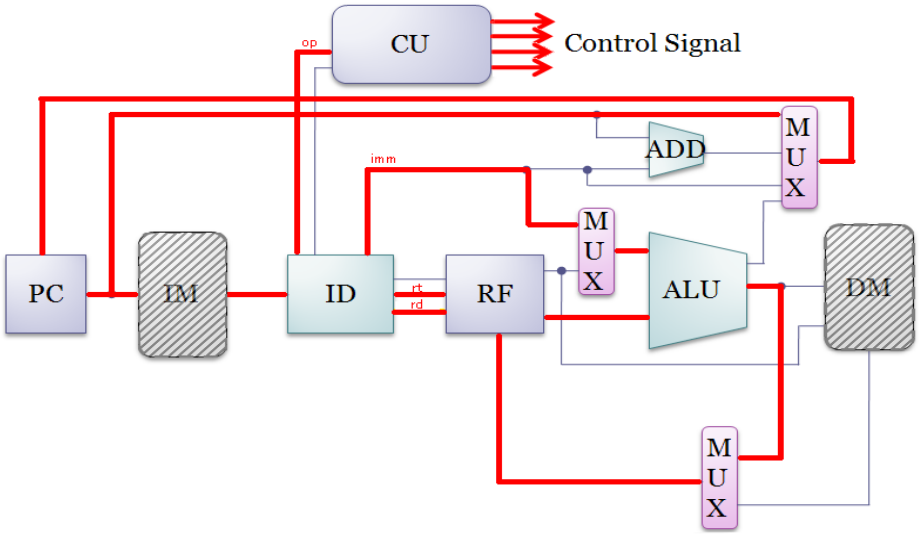

図 5：I5 形式の実行過程

I5 形式では、ID で IM から受け取った命令の識別を行い、CU に op のデータを送る。また演算データのアドレス rt、演算結果を格納するアドレス rd を RF に出力し、もう一方の演算データとなる即値を出力する。ALU は、CU から演算命令の種類を受け取り、即値と rt データの演算を行い出力する。

(3)I8 形式

(a)BQEZ、BNEZ 命令

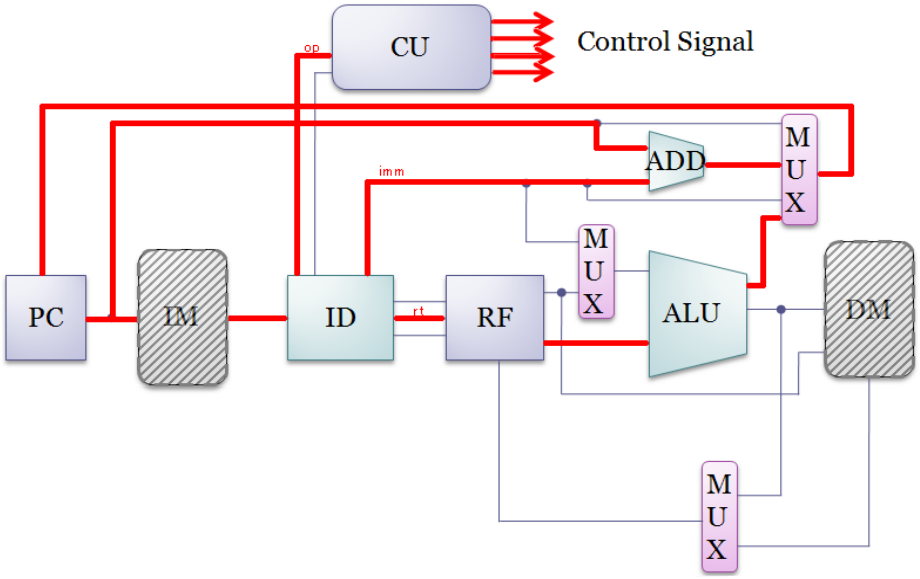

図 6：分岐命令の実行過程

I8 形式の分岐命令では、ID で IM から受け取った命令の識別を行い、CU に op のデータを送る。また条件判定するデータのアドレス rt を RF に出力し、分岐先のアドレス計算を行うデータとなる即値を出力する。ALU は、rt のデータを判定し、分岐するかしないかのデータを出力する。ADD で現在の PC 値と ID からの即値を計算し、分岐先アドレスを出力する。

(b)LDLI 命令

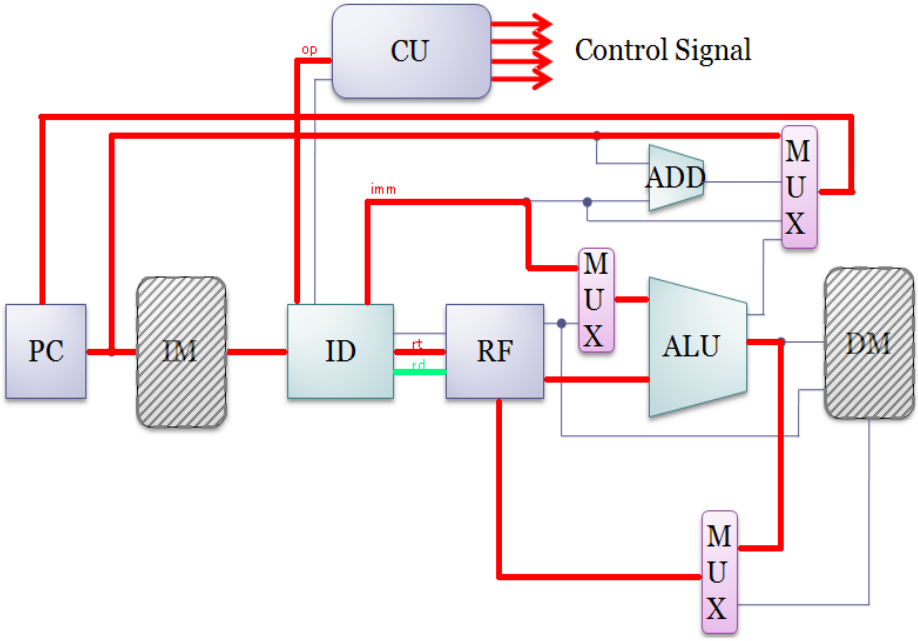

図 7：LDLI 命令の実行過程

LDLI 命令では、ID から演算データのアドレス rt、演算結果格納場所となるアドレス rd を RF に出力し、もう一方の演算データとなる即値を出力する。rd のアドレスは rt と同じアドレスとなっている。

(c)LD 命令

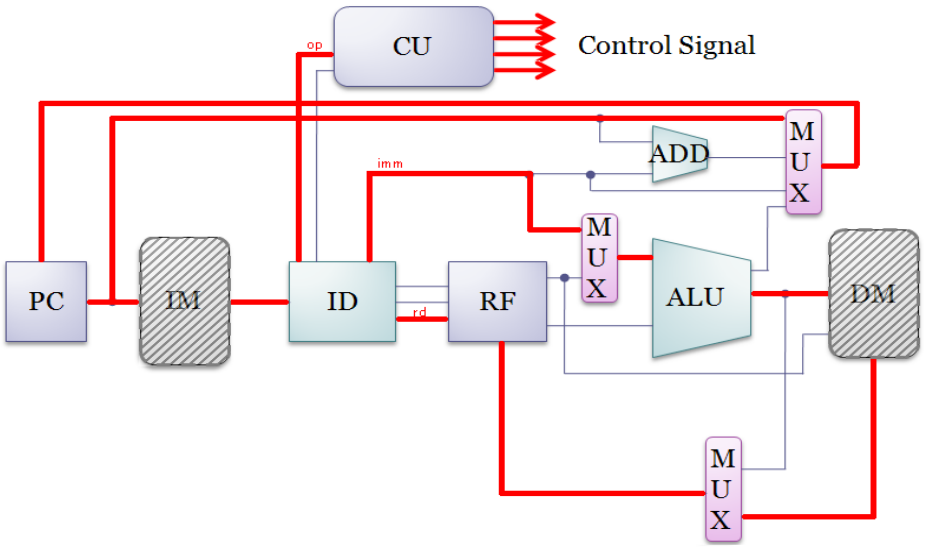

図 8：LD 命令の実行過程

LD 命令では、ID が IM から命令を受け取り、DM からのデータを格納する場所となるアドレス rd を RF に出力し、DM のデータを参照するアドレスとなる即値を出力する。

(d)ST 命令

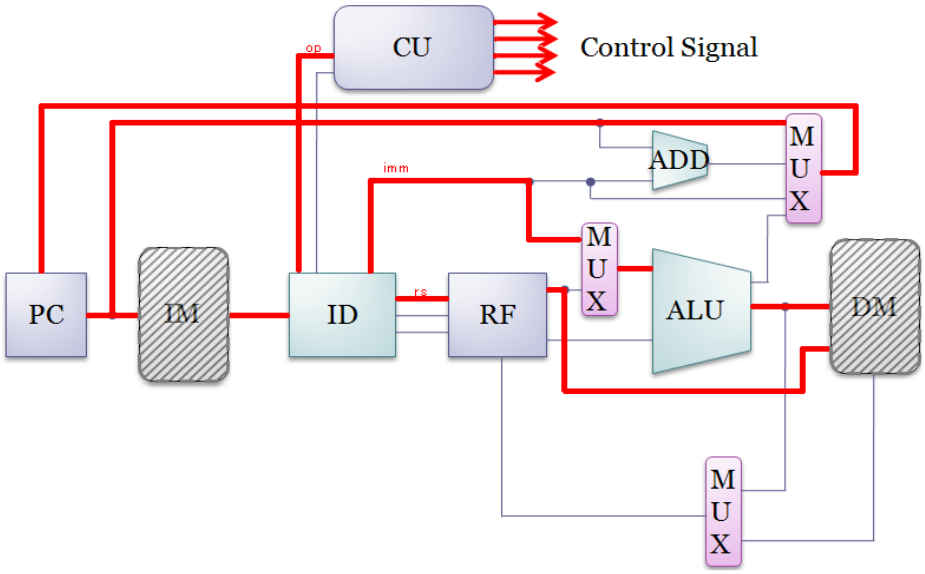

図 9：ST 命令の実行過程

ST 命令では、ID が IM から命令を受け取り、DM に格納するデータとなるアドレス rs を RF に出力し、DM のデータ格納場所となるアドレスを即値として出力する。

(4)J 形式

(a)JUMP 命令の実行過程を以下に示す。

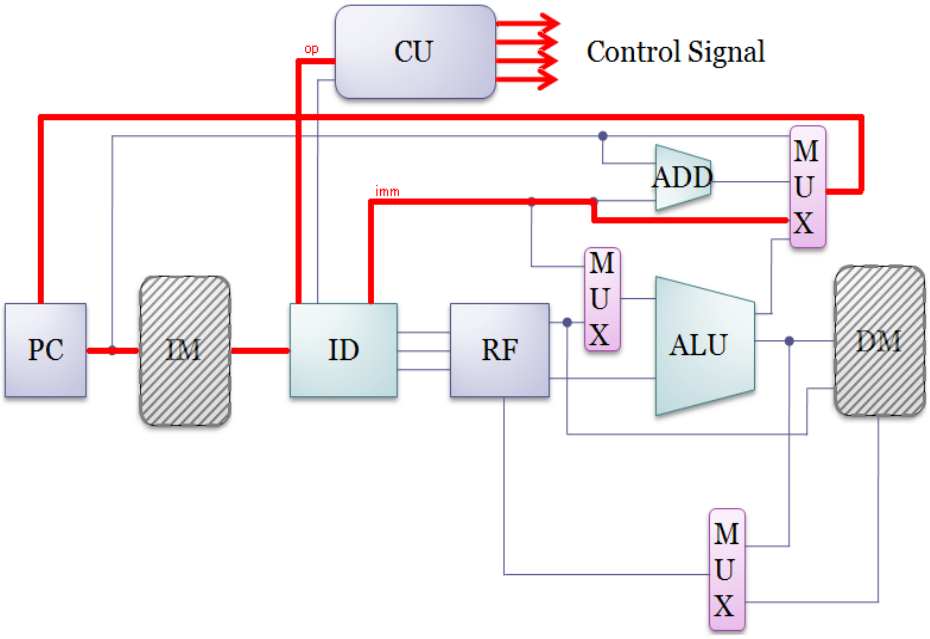

図 10：JUMP 命令の実行過程

JUMP 命令では、ID が IM から命令を受け取り、次の PC 値となるアドレスを即値として出力する。

(b)NOP、HALT

空白命令の NOP は PC+1 となり、プログラム終了命令の HALT は CU が ID からフィールドを受け取り、各モジュールに停止信号を送る。実行過程は他の命令の実行過程の図を参照し、省略する。

4.2　Verilog HDL による設計

SARIS プロセッサの設計には、Verilog HDL を用いた。以下に、一部のモジュールの外部仕様とその詳細について説明する。

(1) ID　…　Instruction Decoder

命令メモリ（IM）から１６ビットの命令を受け取り、命令の種類を判別する op または fn を CU に出力し、また受け取った命令に用いるアドレス、またはデータとなる rs、rt、rd、imm を選出する。図 11 に ID の入出力仕様を示す。

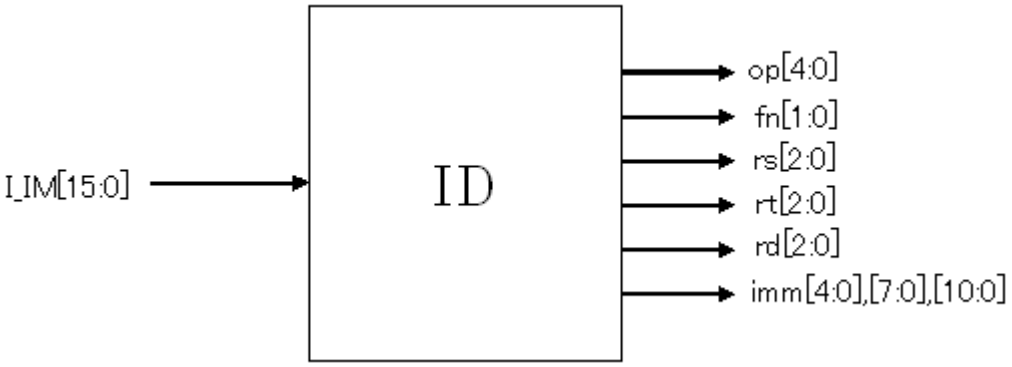

図 11：ID の入出力仕様

ID の設計では、フィールド op の上位 2 bit を形式別に使用することにより効率化を図っている。図 12 に ID のソースの一部を示す。

```
`timescale 1ns / 1ps
module ID(I_IM, O_RS, O_RT, O_RD, O_OP, O_FN, O_IMM);
                              .
                              .
                              .
   assign O_RS = (I_IM[15:14] == 2'b01) ? I_IM[8:6] :
                 (I_IM[15:14] == 2'b11 && I_IM[13:11] == 3'b100) ? I_IM[2:0] : 3'bx;
                              .
                              .
                              .
   assign O_IMM = (I_IM[15:14] == 2'b00) ? I_IM[10:0] :
                  (I_IM[15:14] == 2'b10 && I_IM[10] == 1'b0) ? {6'b000000 , I_IM[10:6]} :
                  (I_IM[15:14] == 2'b10 && I_IM[10] == 1'b1) ? {6'b111111 , I_IM[10:6]} :
                  (I_IM[15:14] == 2'b11 && I_IM[10] == 1'b0) ? {3'b000 , I_IM[10:3]} :
                  (I_IM[15:14] == 2'b11 && I_IM[10] == 1'b1) ? {3'b111 , I_IM[10:3]} : 11'bx;

endmodule
```

図 12：ID の Verilog HDL 記述

(2) RF　…　Register File

ID からデータが格納されているレジスタのアドレス番値を示す rs、rt、rd を受け取り、アドレス番値 rs のデータとアドレス番値 rt のデータを出力する。rd は演算結果や DM からのデータを格納する場所を示すアドレス番値である。図 13 に RF の入出力仕様を示す。

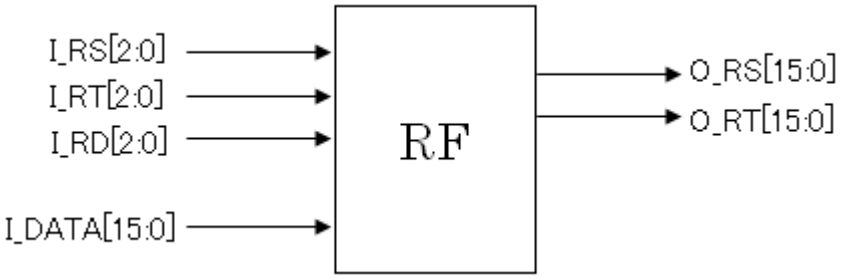

図 13：RF の入出力仕様

(3) CU　…　Control Unit

ID から命令を識別するためのフィールド op または fn を受け取り、命令により PC、RF、ALU、MUX、DM にそれぞれ制御信号を送る。PC には PC 値の更新をするかしないかの制御を行い、RF には演算結果または DM からのデータを格納するのかしないのかの制御を行う。ALU にはどの処理を行うのかという制御を行う。MUX にはどの入力を出力するのかを決める制御を行う。DM には入力データを格納する、また入力アドレスのデータを出力するという処理を制御する。図 14 に CU の入出力仕様を示す。

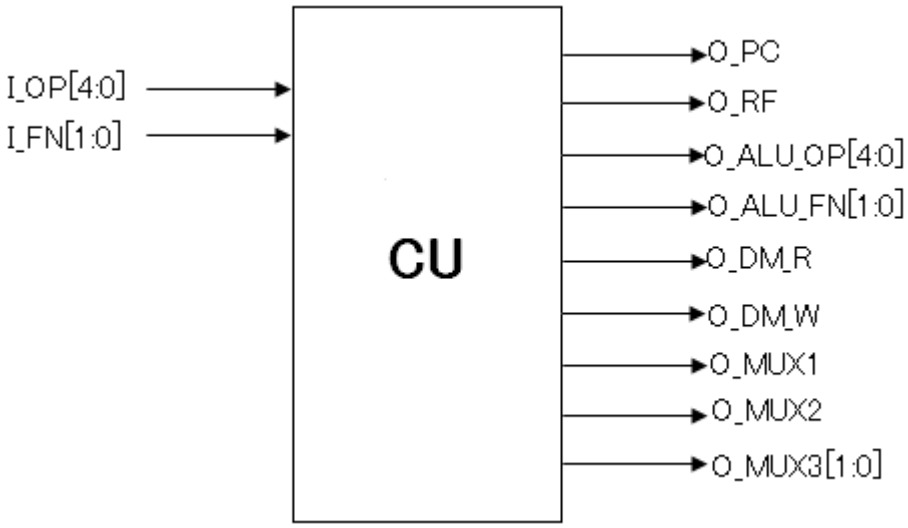

図 14：CU の入出力仕様

4.3 HDL シミュレータでの検証

設計したシングルサイクルプロセッサを、XilinX 社提供の ModelSimXEⅢ6.2g を用いて HDL シミュレーションを行った。シミュレーションに用いたプログラムは N までの和、階乗、二次方程式の根の判別である。各プログラムは、10 までの和、5 の階乗、$X^2+5X+6=0$ の根の判別を実行した。実行結果は全て正しい値が得られた。表 3 に、検証プログラムのシミュレーション結果を示す。

表 3：SARIS のシングルサイクルプロセッサのシミュレーション結果

プログラム	実行命令数	クロックサイクル数
N までの和	44	44
階乗	51	51
二次方程式の根の判別	73	73

検証プログラムの全てが、実行命令数とクロックサイクル数が同じ数値である。これは設計したプロセッサの CPI(clock cycle per instruction)が 1 であることを示している。

4.4 FPGA ボード上での検証

FPGA ボード上に実装するにあたり、開発環境として XilinX 社の ISE9.1i を使用した。論理合成ツールは XST、遅延の計算には Timing Analyzer を用いた。また FPGA ボードには、同じく XilinX 社の Spartan-3 Starter Kit Board を使用した。FPGA 上に実装する際に、設計したプロセッサとプロセッサデバッガを接続した。表 4 に SARIS シングルサイクルプロセッサの設計規模と最大遅延を示す。SARIS シングルサイクルプロセッサでの最大遅延は、ＬＤ命令の場合と想定する。

表 4：SARIS シングルサイクルプロセッサの設計規模と最大遅延

モジュール	スライス数	LUT 数	フリップフロップ数	最大遅延(ns)
SARIS シングル	369	692	140	47.42

SARIS シングルでの FPGA 使用率はスライス数が 19%、LUT 数が 18%、フリップフロップ数が 3%となった。最高動作周波数は 21.18MH z であった。

FPGA ボード上での検証プログラムは、N までの和、階乗、乗算、除算、最大公約数、二次方程式の根の判別を行い、SARIS の全命令を使用した。検証に用いたプログラム全てにおいて正しい結果が得られた。

5　マルチサイクルプロセッサの設計と検証

5.1　アーキテクチャ

図 15 に、マルチサイクルのデータパスを示す。

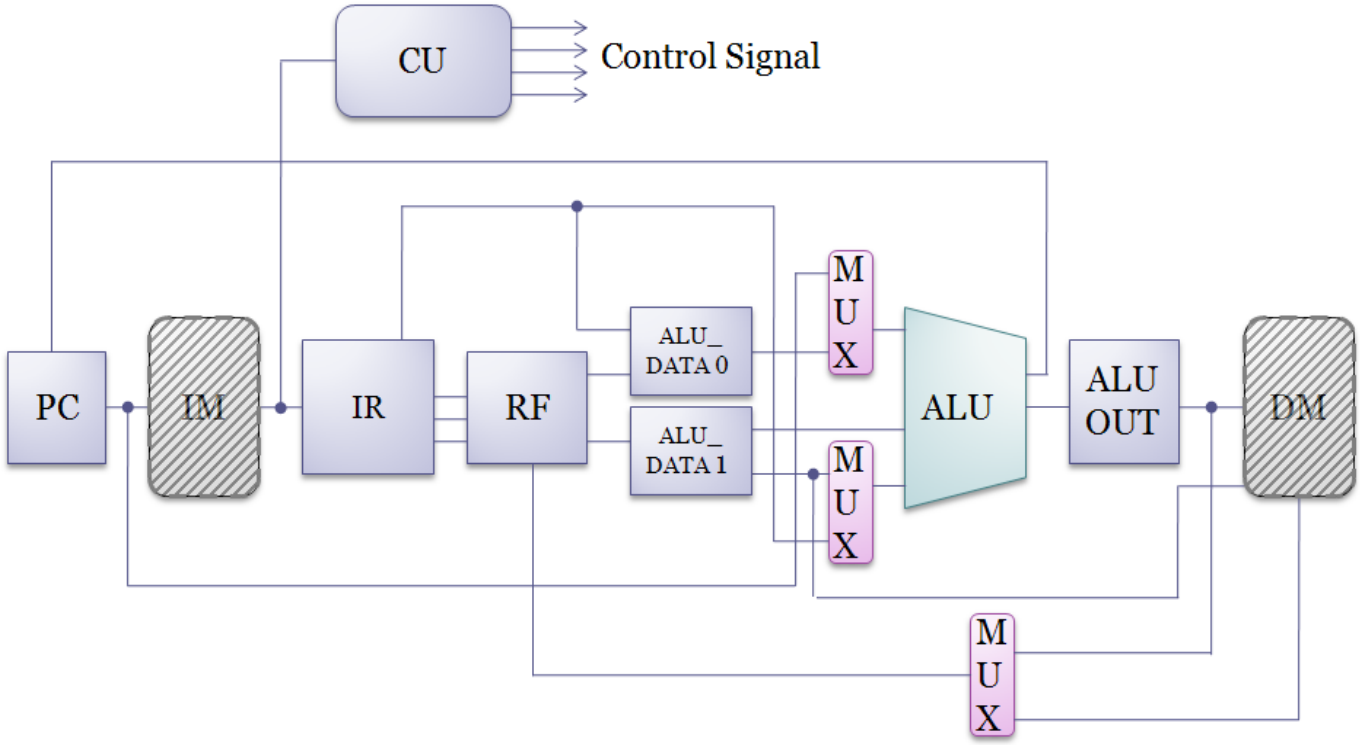

図 15：マルチサイクルデータパス

シングルサイクルとの違いは以下の 4 つの点である。

① IM から読み出した命令を一時保持してから解読（IR）。

② RF から読み出した値と即値を一時保持するモジュール（ALU_DATA0、ALU_DATA1）の追加。

③ PC 値の計算を ALU で行う。

④ 演算処理結果を一時保持するモジュール（ALU_OUT）の追加。

追加したモジュールと、シングルとの動作が違うモジュール IR、ALU_DATA0、ALU_DATA1、ALU、ALU_OUT のそれぞれの動作と意味を以下に示す。

- IR … Instruction Register

  ⇒IM から読み出した命令を一時保持し、命令を解読、そして命令により rs、rt、rd、imm を出力する。

- ALU_DATA0

  ⇒RF から読み出したデータと IR から送られてきた即値を一時保持し、命令によりどちらか一方を選択する。

- ALU_DATA1

  ⇒RF から読み出したレジスタ番地 rt のデータを一時保持する。また分岐条件を ALU に出力する。
- ALU　…　Arithmetic Logic Unit

  ⇒算術演算、PC のアドレス計算を行う。
- ALU_OUT

  ⇒ALU で演算結果出力を一時保持する。次の PC 値となるアドレス計算結果は保持しない。

設計したマルチサイクルの各 Phase における実行と動作を表 5 に示す。J 形式命令と I8 形式の条件分岐命令（BEQZ、BNEZ）と ST 命令は P2 で終了し、その他の R 形式、I5 形式、条件分岐命令と ST 命令以外の I8 形式は P3 で終了となる。全ての命令で P2 で PC 更新となるように、分岐命令以外は ALU での PC 値計算をクロックに同期させている。

表 5：各 Phase における実行と動作

	J 形式、条件分岐、ST 命令	R、I5 形式、I8 形式の LDLI、LD 命令
P0	PC 更新、命令フェッチ	命令フェッチ
P1	命令デコード、RF 読み出し	
P2	ALU 処理	
P3		PC 更新、RF に演算結果格納

SARIS マルチサイクル設計において、4 クロック実行となる R 形式、I5 形式、I8 形式の LDLI、LD 命令と、3 クロック実行となる J 形式と I8 形式の条件分岐命令と ST 命令について、その詳細をそれぞれのデータパスを図に示しながら以下に解説する。

(1)　4 クロック実行

フェーズ 0 で命令フェッチ、フェーズ 1 で命令デコードと RF の読み出しを行う。フェーズ 2 で ALU により次の PC 値の計算と演算処理を行い、フェーズ 3 で PC 更新、RF に演算結果を格納する。R 形式、I5 形式、I8 形式の LDLI、LD 命令のマルチサイクル設計での実行過程をそれぞれ図 16～図 19 に示す。

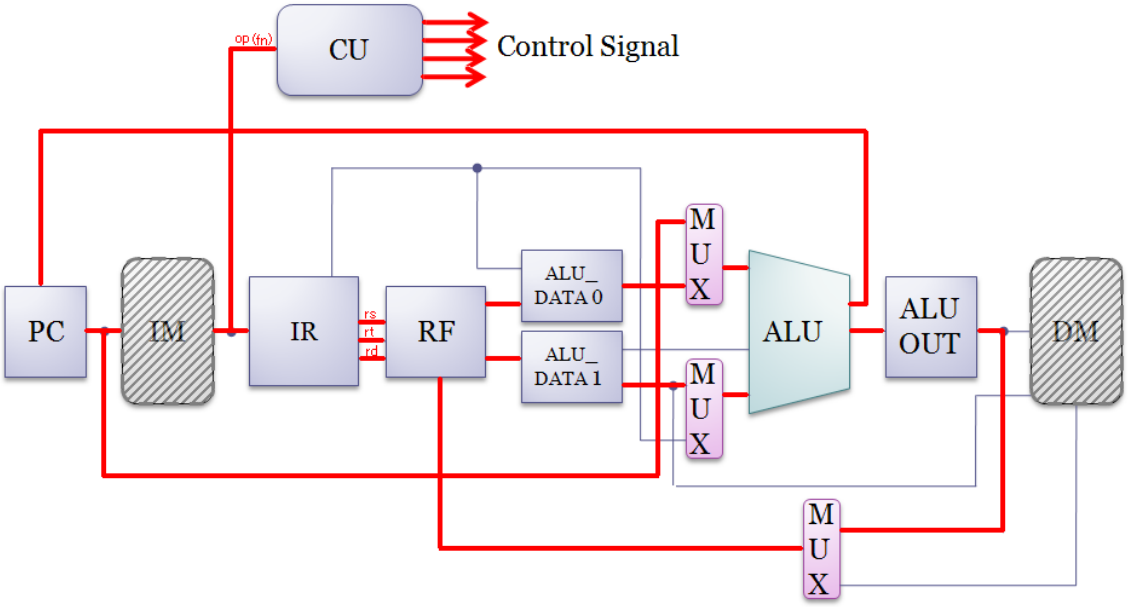

図 16：マルチサイクル設計での R 形式命令の実行過程

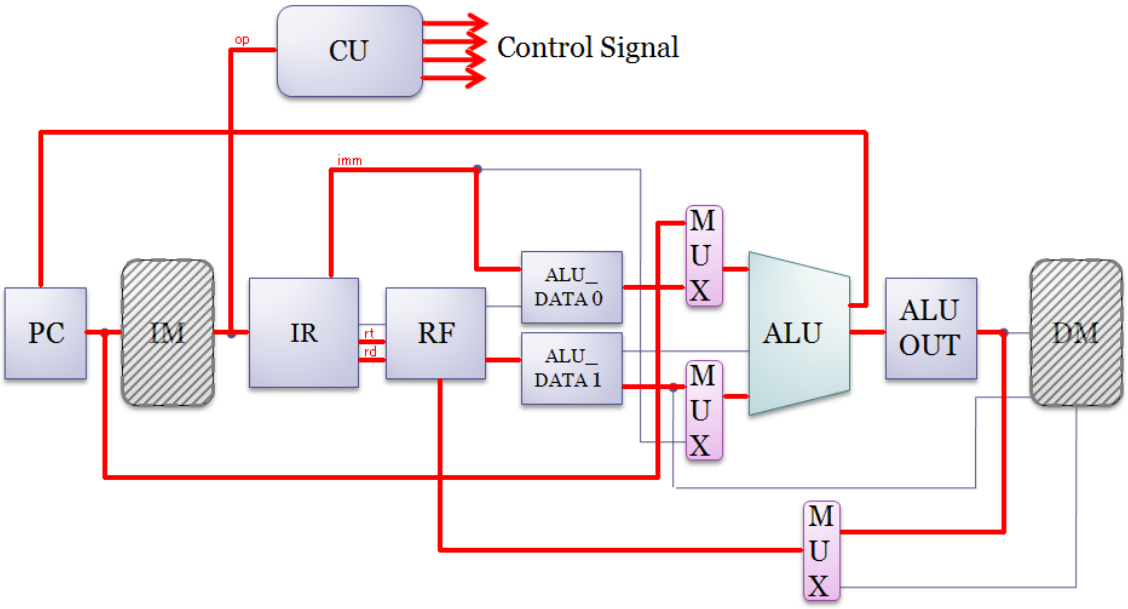

図 17：マルチサイクル設計での I5 形式命令の実行過程

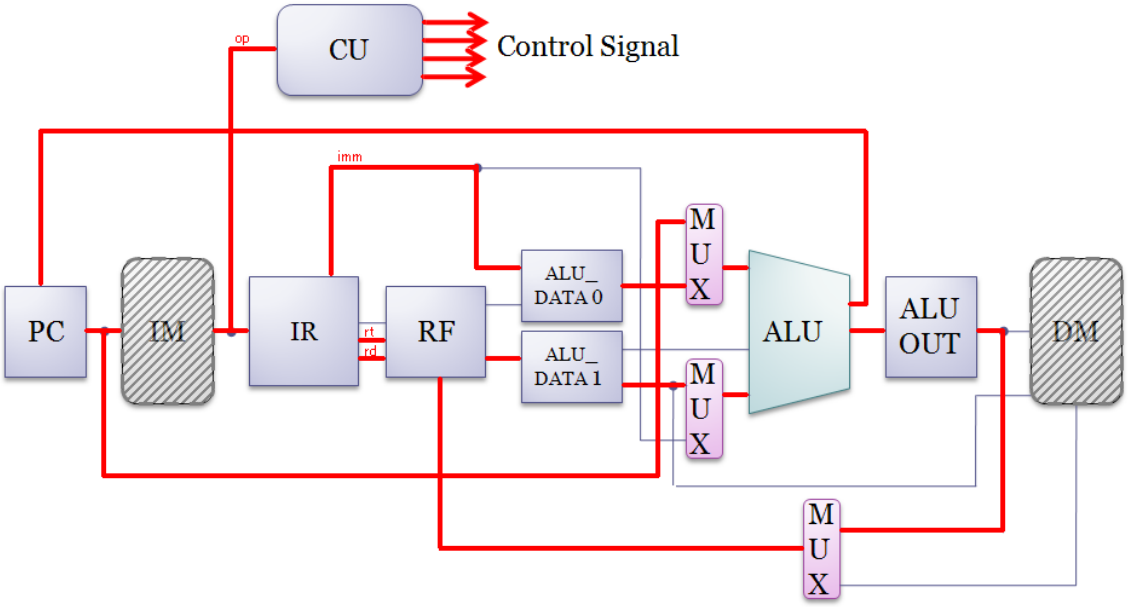

図 18：マルチサイクル設計での LDLI 命令の実行過程

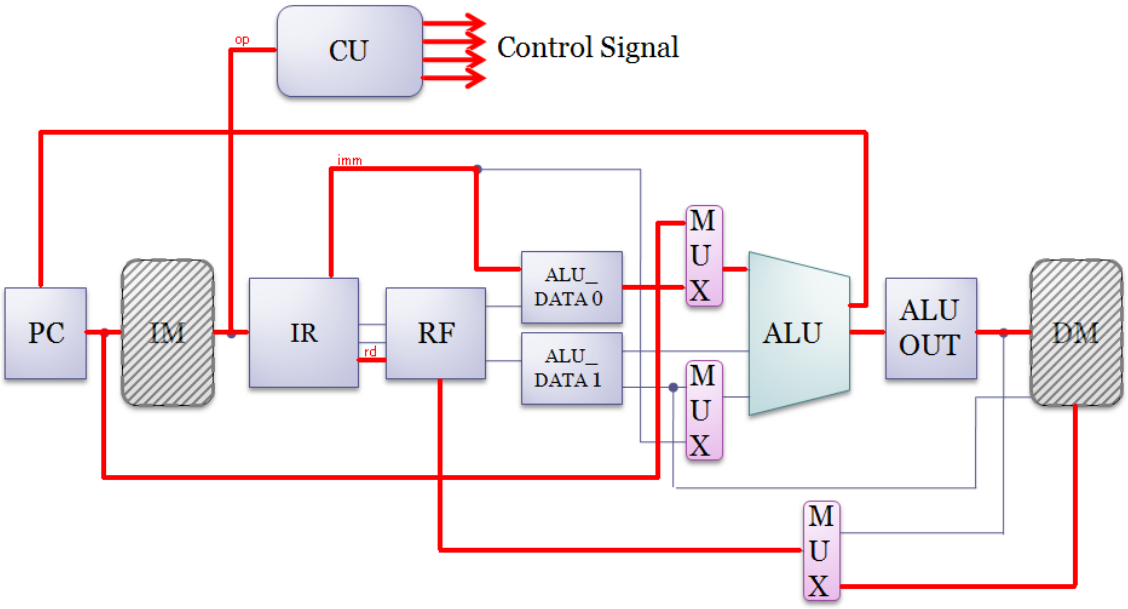

図 19：マルチサイクル設計での LD 命令の実行過程

(2)　3 クロック実行

フェーズ 0 で PC 更新と命令フェッチを行う。フェーズ 1 で命令デコードと RF 読み出しを行い、フェーズ 2 では ALU により次の PC 値の計算、また ST 命令では DM にデータを格納する場所となるアドレスを出力する。フェーズ 2 で終了となるので、次はフェーズ 0 の処理を行う。

条件分岐命令、ST 命令、J 形式の JUMP 命令の実行過程をそれぞれ図 20～図 22 に示す。J 形式の NOP、HALT 命令の実行過程はシングルと同様に他の図を参照し、省略する。

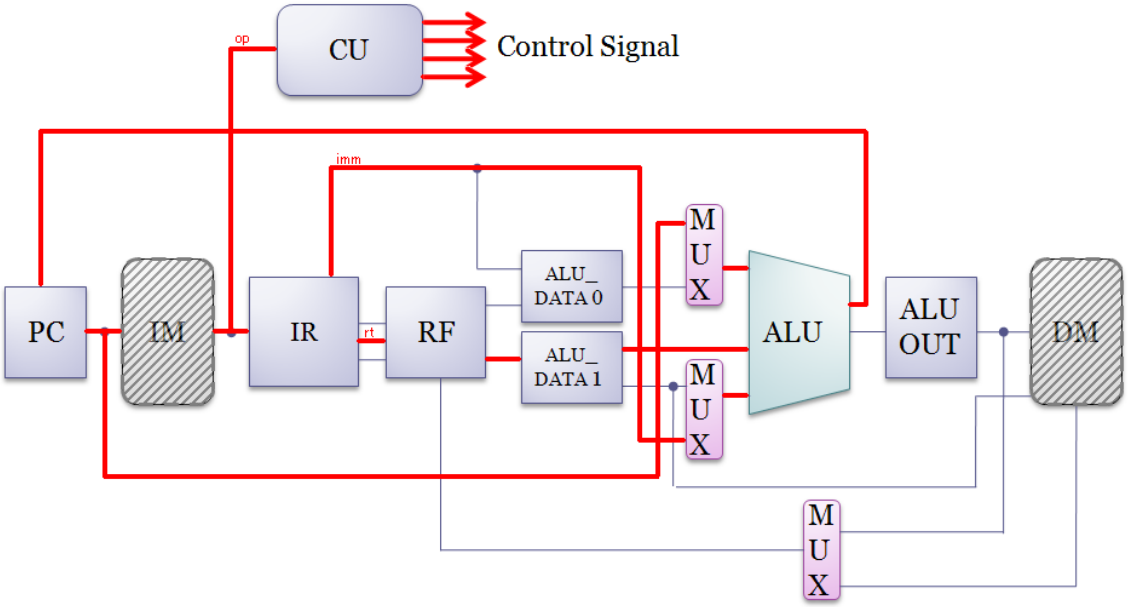

図 20：マルチサイクル設計での条件分岐命令の実行過程

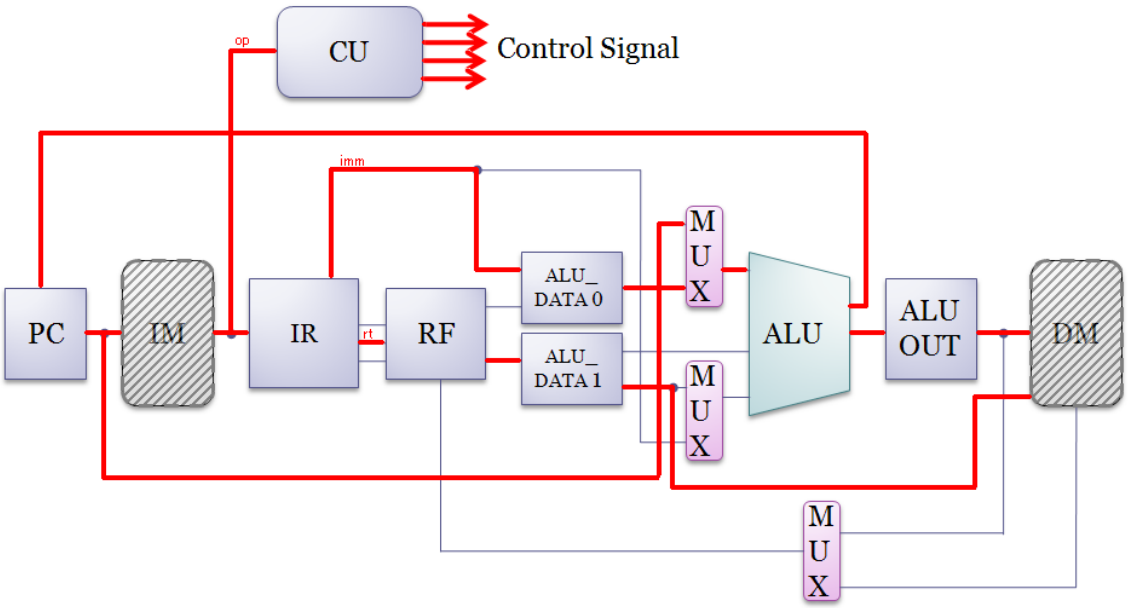

図 21：マルチサイクル設計での ST 命令の実行過程

ST 命令はシングルサイクル設計と違い、DM に格納されるデータが ALU_DATA1 から DM に接続されている。これはクロック間の最大遅延を短くするためである。

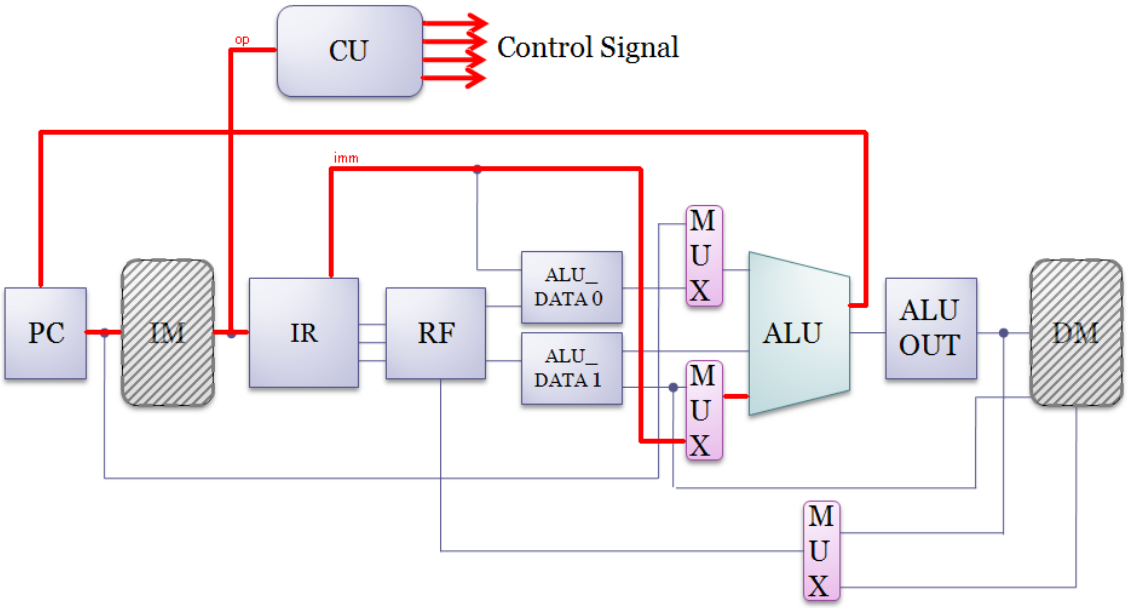

図 22：マルチサイクル設計での JUMP 命令の実行過程

5.2　Verilog HDL による設計

SARIS プロセッサのシングルサイクル設計と同様に、マルチサイクルによる設計でも Verilog HDL を用いた。以下に、モジュール ALU_DATA0、ALU_DATA1、ALU の外部仕様とその詳細について説明する。

(1)ALU_DATA0

RF のアドレス rs の出力データと、IR からの即値を保持する。命令により、どちらか一方のデータを出力する。図 23 に ALU_DATA0 の入出力仕様を示す。

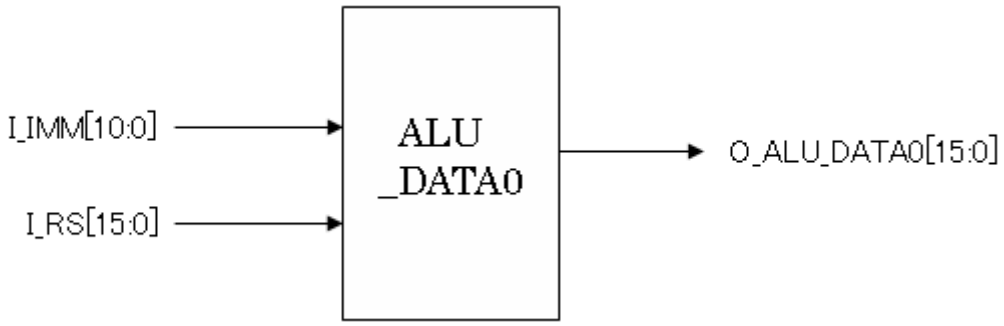

図 23：ALU_DATA0 の入出力仕様

(2)ALU_DATA1

RF のアドレス rt の出力データを保持する。条件分岐命令の場合、保持するデータの値が 0 なら 0 を、0 以外なら 1 を分岐する条件として ALU に出力する。その他の命令では保持しているデータを出力する。図 24 に ALU_DATA1 の入出力仕様を示す。

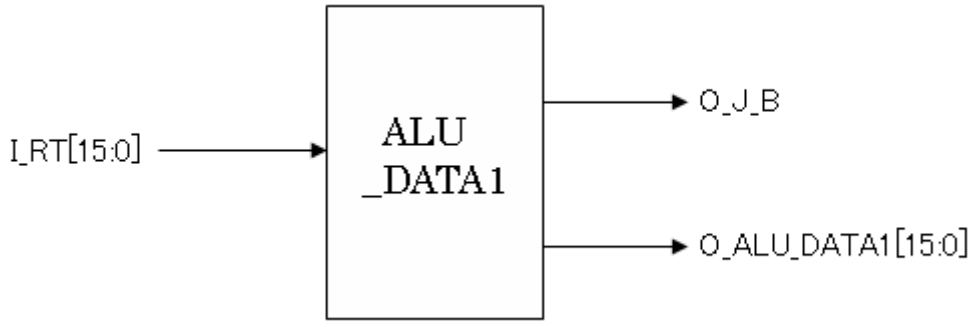

図 24：ALU_DATA1 の入出力仕様

(3) ALU　…　Arithmetic Logic Unit

入力データの I_ALU0 と I_ALU1 は、算術演算データ、または PC のアドレス計算を行うデータとなる。I_J_B により分岐条件が入力される。出力データ O_ALU_PC は PC に接続され、O_ALU は ALU_OUT に接続される。図 25 に ALU の入出力仕様を、図 26 に ALU のソースの一部を示す。

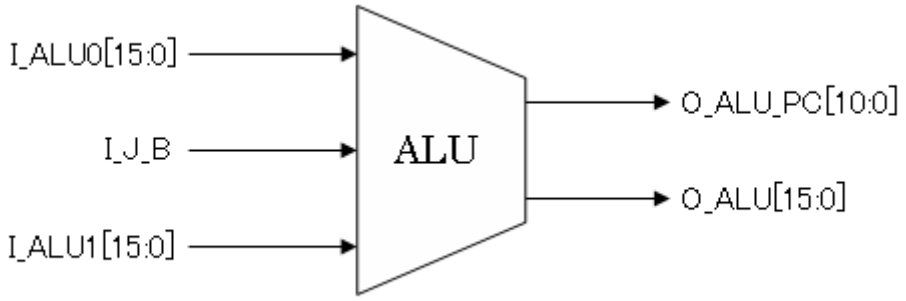

図 25：ALU の入出力仕様

```
`timescale 1ns / 1ps

`define Rr      5'b01000
`define Rrr     5'b01001

`define ADDI    5'b10000
`define SUBI    5'b10001
                    .
                    .
                    .
module ALU(I_ALU0, I_ALU1 , O_ALU, I_CU_OPE, I_CU_FN, I_J_B, O_ALU_PC, CLK ,XRST);
                    .
                    .
                    .
   reg [10:0] o_alu_pc;

   assign O_ALU = (I_CU_OPE == `Rr)&&(I_CU_FN == 2'b00) ? (I_ALU1  + I_ALU0);//ADD
               (I_CU_OPE == `Rr)&&(I_CU_FN == 2'b01 ) ? (I_ALU1  - I_ALU0);//SUB
                    .
                    .
                    .
   assign O_ALU_PC = (I_CU_OPE == `BEQZ) && (I_J_B == 0) ? I_ALU0[10:0] + I_ALU1[10:0] + 1'b1 :
                (I_CU_OPE == `BEQZ) && (I_J_B != 0) ? I_ALU0[10:0] + 1'b1 :
                (I_CU_OPE == `BNEZ) && (I_J_B != 0) ? I_ALU0[10:0] + I_ALU1[10:0] + 1'b1 :
                (I_CU_OPE == `BNEZ) && (I_J_B == 0) ? I_ALU0[10:0] + 1'b1 :
                 o_alu_pc;

      always@(posedge CLK or negedge XRST)
         if(!XRST) begin
            o_alu_pc <= 11'b0;
         end
         else if(I_CU_OPE == `JUMP)begin
            o_alu_pc <= I_ALU1[10:0];
         end
         else if(I_CU_OPE[4:3] == 2'b01  || I_CU_OPE[4:3] == 2'b10 || I_CU_OPE == `NOP
            || I_CU_OPE == `LD || I_CU_OPE == `ST || I_CU_OPE == `LDLI)begin
            o_alu_pc <= I_ALU0[10:0] + 1'b1;
         end
         else begin
            o_alu_pc <= o_alu_pc;
         end

endmodule
```

図 26：ALU の Verilog HDL 記述

5.3 HDL シミュレータでの検証

設計したマルチサイクルプロセッサを、シングルサイクルと同様にして HDL シミュレーションを行った。検証プログラムもシングルと同様に 10 までの和、5 の階乗、$X^2+5X+6=0$ の根の判別を実行した。それぞれ実行結果は全て正しい値が得られた。表 6 に、検証プログラムのシミュレーション結果を示す。

表 6：SARIS マルチサイクルプロセッサのシミュレーション結果

プログラム	実行命令数	クロックサイクル数
N までの和	44	162
階乗	51	189
二次方程式の根の判別	73	268

N までの和、階乗、二次方程式の根の判別での CPI は、順に約 3.68、約 3.7、約 3.67 という結果になった。これは J 形式命令と条件分岐命令、ST 命令が 3 クロック実行、その他の命令が 4 クロック実行であることで CPI が増減していることを表している。J 形式命令や条件分岐命令、または ST 命令が多いほど CPI が小さくなることが考えられる。

5.4 FPGA ボード上での検証

実装環境、開発環境ともに、シングルサイクルでの検証と同様である。表 7 に、SARIS マルチサイクルプロセッサの設計規模と最大遅延を示す。SARIS マルチサイクルプロセッサでの最大遅延は、ＬＤ命令の場合と想定する。

表 7：SARIS マルチサイクルプロセッサの設計規模と最大遅延

モジュール	スライス数	LUT 数	フリップフロップ数	最大遅延(ns)
SARIS マルチ	398	744	239	15.077

SARIS マルチでの FPGA 使用率はスライス数が 20%、LUT 数が 19%、フリップフロップ数が 6%となった。最高動作周波数は 66.32MＨｚであった。シングルと比較すると、最高動作周波数が約 3.1 倍となったが、CPI が条件分岐命令、ST 命令や J 形式命令に依存することを考えても、シミュレーション検証で平均 3.7 という結果を得たので、単純なプログラムではマルチよりシングルの方が処理が早いと判断できる。マルチサイクルでの動作周波数の向上が今後の課題である。

FPGA ボード上での検証プログラムは、シングルサイクルでの検証と同様に 6 つのプログラムを行い、SARIS の全命令を使用した。検証に用いたプログラム全てにおいて正しい結果が得られた。

6　ハード/ソフト協調学習システムの評価

6.1　汎用アセンブラ

実際に自分自身で検証プログラムを機械語に直し、HDL シミュレータ、または FPGA ボード上で検証したところ、エラーが発生すると機械語に直した際の計算間違いなどによるエラーなのか、プロセッサの設計によるエラーなのかわからなくなる。汎用アセンブラを用いることで、検証プログラムを機械語に直す時間が短縮でき、デバッグする際プロセッサの設計を見直せばいいので、全体の検証時間とデバッグ時間の短縮になった。

6.2　プロセッサデバッガ

プロセッサデバッガを用いることで、実機上でのデバッグが容易に可能であった。デバッガとの検証結果とシミュレーション結果を比較することで、デバッグ時間が短縮された。表 8 に主に使用したデバッガコマンドを示す。

表 8：使用したデバッガコマンド一覧

コマンド	実行動作
load im	im バッファに命令ファイルの読み込み
load dm	dm バッファにデータファイルの読み込み
send im	im バッファから命令メモリに書き込み
send dm	dm バッファからデータメモリに書き込み
run all	プログラム全実行
run clk (x)	指定したクロック分実行する
read pc	PC の値を読み出し
read rf	RF の値を読み出し
read dm	dm の値を読み出し
set bp (x)	ブレイクポイント設定
run bp	次のブレイクポイントまで実行
init	全バッファの初期化

コマンド run clk (x)によりシングルとマルチの両方で 1 命令実行が可能であったので、read pc、read rf で PC と RF の値を確認しながら検証を行うことができた。これによりデ

バッグが必要である箇所が容易に発見できた。またシミュレーションで正しい結果が得られていても、実機上では動作しない場合があった。この時のデバッグに、コマンド set bp、run bp によるブレイクポイント実行によりデバッグ個所が特定できた。

6.3　システム全体の評価

本研究で、ハード/ソフト協調学習システムを用いてシングルサイクルとマルチサイクルの 2 つのプロセッサを設計し、検証を行ってきたことによりハードウェアとソフトウェアの両方の学習を進めることができた。ソフトウェア学習の面では、アセンブリプログラミング技術の向上、プロセッサアーキテクチャの理解ができた。ハードウェア学習の面では、HDL プログラミング技術の向上とプロセッサ設計能力が習得できた。

プロセッサ設計支援ツールの汎用アセンブラとプロセッサデバッガは、プログラムの検証時間とデバッグ時間が大きく短縮できたことから、ハードウェア学習において有効性が高いと考えられる。

表 9 に、本研究を通しての学習時間を示す。

表 9：本研究を通しての学習時間

工程	学習時間(時間)	
	シングル	マルチ
アセンブリプログラミング	３０	
命令セット設計	１０	
HDL 設計	６０	８０
HDL シミュレータでの検証	２０	１５
FPGA 上ボードでの 動上作検証	１５	１０

7 おわりに

本論文では、本研究室で開発を進めているハード/ソフト協調学習システムを用いて、独自に命令セットを定義し、シングルサイクルプロセッサ、マルチサイクルプロセッサを設計した。設計したプロセッサを、HDL シミュレータで検証、さらには FPGA ボード上に実装し、検証を行った。最後に、本研究を通して利用したハード/ソフト協調学習システムの評価、さらにはプロセッサ設計支援ツールの汎用アセンブラとプロセッサデバッガの評価を行った。

今後の課題としては、マルチサイクルの動作周波数の向上、パイプライン設計があげられる。また本研究で評価していないプロセッサ設計支援ツールを評価し、ハード/ソフト協調学習システムの拡張を検討していくことがあげられる。

謝辞

本研究の機会を与えてくださり、ご指導を頂きました山崎勝弘教授に深く感謝いたします。また、本研究に関して様々な相談に乗って頂き、貴重な助言を頂いた志水建太氏をはじめ、様々な面で貴重な助言や励ましを下さった研究室の皆様に深く感謝いたします。

～参考文献～

[1] 中森章　著：マイクロプロセッサ・アーキテクチャ入門、CQ 出版、2004.

[2] 大八木睦：ハード/ソフトコラーニング上でのアーキテクチャ可能なプロセッサシミュレータ(MONI シミュレータ)の使用法, 立命館大学理工学研究科, 2004.

[3] 難波翔一郎：FPGA ボード上での単一サイクルマイクロプロセッサの設計と検証、立命館大学理工学部情報学科卒業論文、2005.

[4] 難波翔一郎：プロセッサ設計支援ツールの実装とハード/ソフト協調学習システムの評価、立命館大学理工学研究科修士論文、2007.

[5] 志水建太：ハード/ソフト協調学習システム上でのプロセッサ設計とプロセッサデバッガによる検証、立命館大学理工学部情報学科卒業論文、2007.

[6] 榎本雄太：ARM ライクプロセッサの設計と FPGA ボードへの実装の検討、立命館大学理工学部情報学科卒業論文、2007.

[7] 難波翔一郎・志水建太・山崎勝弘・小柳滋：プロセッサ設計支援ツールの設計・実装とハード/ソフト協調学習システムの評価、FIT2007.LC002.2007.

[8] 大八木睦：ハード/ソフト・コラーニングシステム上でのアーキテクチャ可変なプロセッサシミュレータの設計と試作, 立命館大学理工学研究科修士論文, 2004.

[9] 池田修久：ハード/ソフト・コラーニングシステム上での FPGA ボードコンピュータの設計と実装, 立命館大学理工学研究科修士論文, 2004.

[10] 深山正幸・北川章夫・秋田純一・鈴木政國　著：HDL による VLSI 設計、共立出版、2002.

[11] 堀桂太郎：図解 Verilog HDL 実習、森北出版、2006.

[12] David A.Patterson/John L.Hennessy 著／成田光彰　訳：コンピュータの構成と設計（上）（下)、日経 BP 社、2006.