卒業論文

プロセッサ設計支援ツールを用いた
独自プロセッサの設計

氏　名　：　宮崎 匡史
学籍番号　：　2260050076-0
担当教員　：　山崎 勝弘 教授
提出日　：　2009年2月19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、本研究室が開発を進めているハードウェアとソフトウェアの両方の知識を得ることができるハード/ソフト協調学習システムを用いて Verilog HDL による独自プロセッサ設計を行い、設計した独自プロセッサについて HDL シミュレーションを行い、FPGA ボードに実装して動作検証を行った。さらに独自プロセッサ設計に用いたプロセッサ設計支援ツールについて評価を行った。評価の結果、命令セット定義ツール、命令セットシミュレータ、命令セットアセンブラ、プロセッサモニタに改善が考えられ、一部のツールに改善が行われた。

ハードソフト協調学習システムを用いて独自プロセッサの設計を行う上で、プロセッサアーキテクチャとハードウェア記述言語を学習し、プロセッサ設計支援ツールの検証を行った。独自プロセッサ設計を行う上で用いるプロセッサ設計支援ツールが有効なツールであることを評価することが本研究の目的である。

目次

図目次

表目次

1　はじめに

近年の集積回路の集積技術の発達により、様々な分野で力を発揮し、回路が小型化されている。身近なものでは、テレビ、液晶テレビ、携帯電話、パソコン、デジタルカメラ、自動車、MP3 プレーヤーなどがあり、これらは年を重ねるごとに回路が小型化されて機能が増え、物が小さくなっている。これは組み込みシステム LSI の技術が発達してきたためである。組み込みシステム LSI とは、一枚のチップに複数の機能を持たせて回路規模を小型化することで小さい部品でも複数の機能を持つことができる。そして、そのチップを使った基盤はさらに複数の機能をもたせることができるようになるのである。しかし、ここでソフトウェアとハードウェアの最適分割が問題になり、どこまでをソフトウェアで設計し、どこまでをハードウェアで設計すればよいのかが問題になる。ソフトウェアの設計が多い場合、設計コストはかからないが、処理速度は遅くなる。一方ハードウェアの設計が多い場合、処理速度は速いが、設計コストが大幅にかかってしまう。つまり、ソフトウェアとハードウェアの両方をうまく調和させる必要がある。

そこで、ソフトウェアとハードウェアの両方の知識を持った人材が必要になる。本研究室ではソフトウェアとハードウェアの両方の知識を得るためにハード/ソフト協調学習システムの開発を進めている。ハード/ソフト協調学習システムとは、プロセッサ設計をする上でソフトウェアとハードウェアの知識を学習することができるシステムである。まず、ソフトウェアの学習を行い、次にハードウェアの学習を行うことでアーキテクチャを意識したプログラミングを行うことができる。[10]

以上のような背景より、本研究はハード/ソフト協調学習システムを用いて、まずソフトウェア学習を行い、プロセッサアーキテクチャを理解する。次にプロセッサ設計支援ツールを用いて、命令セット定義ツールを用いた独自プロセッサの命令セットを定義し、命令セットシミュレータを用いて定義した命令セットを用いたアセンブリプログラムを作成し、シミュレーションを行った。そして Verilog HDL による独自プロセッサ設計を行うことでハードウェア学習を行い、命令セットアセンブラを用いてアセンブリプログラムを機械語に変換して Verilog HDL を用いて設計した独自プロセッサをプロセッサデバッガと接続し、HDL シミュレータを用いてシミュレーションを行った。最後に設計した独自プロセッサを FPGA ボード上に実装してプロセッサモニタとプロセッサデバッガを接続し、プロセッサの動作検証を行う。プロセッサ設計を行う上で用いたプロセッサ設計支援ツールの評価を行うことで、今後より有効なツールにすることを目的とする。

本論文では、第 2 章でハード/ソフト協調学習システムとプロセッサ設計支援ツールについて説明する。第 3 章では設計した独自プロセッサの設計方針と命令セットアーキテクチャ、プロセッサアーキテクチャについて説明する。第 4 章では Verilog HDL による独自プロセッサの設計と FPGA ボードへの実装、プロセッサ設計支援ツールを用いた結果により得られたツール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　システムの概要

ハード/ソフト協調学習システムとは、プロセッサの設計を考慮しながらハードウェアとソフトウェアの両方を学習することができるシステムである。ソフトウェア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）を用いてアセンブリ言語の学習とプロセッサのアーキテクチャを学ぶことができる。MONI 仮想シミュレータとは本研究室で 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。ハードウェアでは、シミュレータを動作させて学んだプロセッサのアーキテクチャの知識を用いて HDL によるプロセッサ設計を行う。設計したプロセッサが動作するかシミュレーションを行い、FPGA ボードコンピュータに実装して実機で動作検証を行うことで学習する。このように、アセンブリ言語によるソフトウェア学習と HDL によるプロセッサ設計を行うことで、ソフトウェアとハードウェアの知識をつけることが本システムの目的である。

MONI 仮想シミュレータとは、目で 1 命令の動作を確認することができ、わかりやすくプロセッサのアーキテクチャを理解できるツールである。MONI 仮想シミュレータの動作を理解した上で実際にプロセッサのハードウェア設計を行うので深くプロセッサのアーキテクチャについて理解することができる。また、プロセッサを高速に動作させることを目標にプロセッサ設計を行うなどハード/ソフト協調学習システムはソフトウェアとハードウェアを学べる一方でプロセッサの動作を学習し、新たに独自プロセッサを設計できるツールである。[4] [7]

学習の仕方は、図 1 がハード/ソフト協調学習システムを用いた学習体系である。左にソフトウェアの学習手順があり、まず MONI 仮想シミュレータを用いて事前に作った MONI 用のアセンブリプログラムを動作させる。MONI プロセッサのシングルサイクル、マルチサイクル、パイプライン、スーパースカラを選択して動作を確認できる。プログラムの命令ごとにプロセッサのデータパス動作を確認することができ、プロセッサアーキテクチャとプロセッサのデータパス動作の知識を得られる。

その後、右のハードウェアの学習手順である。まず、MONI プロセッサの設計や命令セット定義ツールを用いた命令セットの定義による独自のプロセッサ設計などの HDL を用いたプロセッサ設計を行う。次に作成したプロセッサを HDL シミュレータでシミュレーションをしてデバッグを行う。そして、デバッグを行ったプロセッサを FPGA ボード上に実装して動作を検証する。作ったプロセッサとプロセッサデバッガを接続し、プロセッサモニタにより FPGA ボード上で検証する。検証にはソフトウェア学習で作成したアセンブリプログラムを用いて動作の検証を行うことができる。このようにハードウェアとソフトウェアの両方を学習することができるのである。

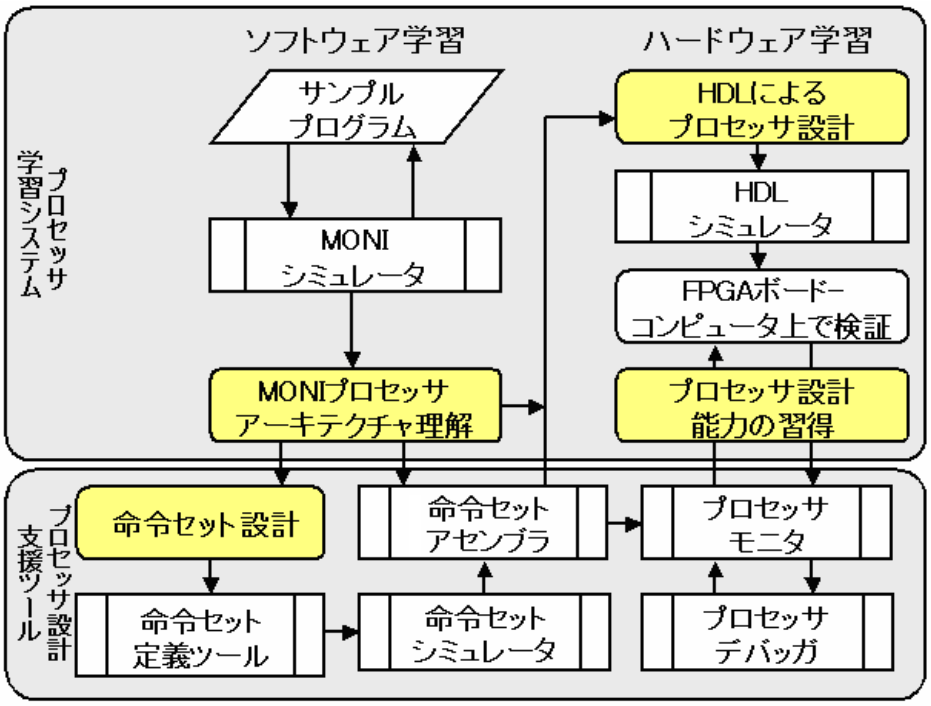

図 1：ハード/ソフト協調学習システムの学習体系

2.2 プロセッサ設計支援ツール

ハードウェアを学習する上で使用するプロセッサ設計支援ツールでは、命令セット定義ツール、命令セットシミュレータ、命令セットアセンブラ、プロセッサモニタ、プロセッサデバッガがある。図 2 は命令セットを定義する流れを示している。[1][2]

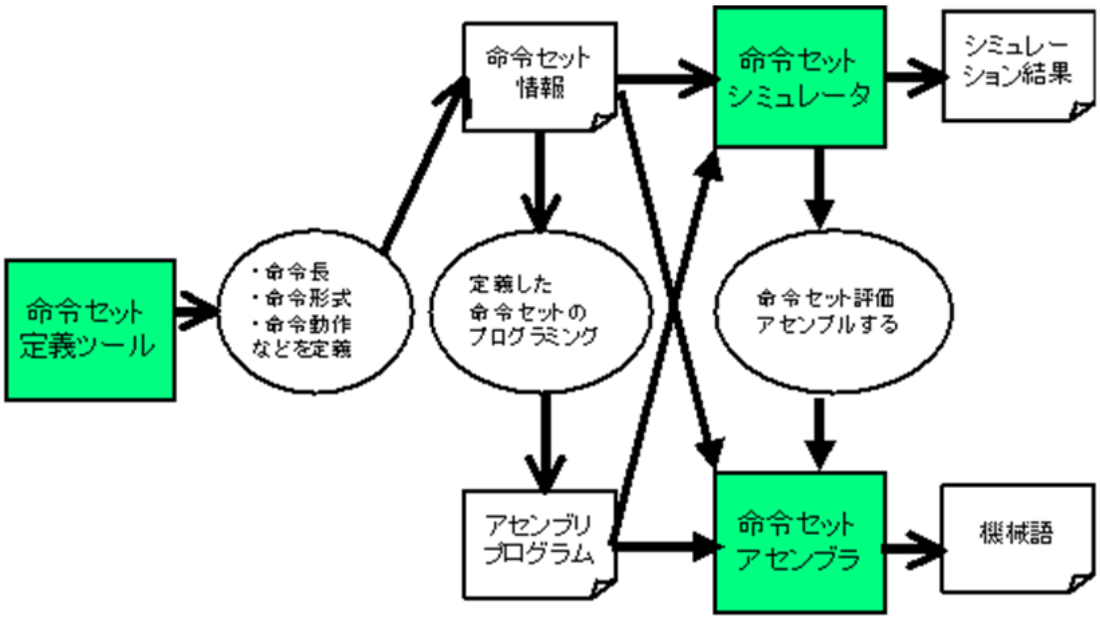

図 2：命令セット定義とシミュレーションの流れ

(1)　命令セット定義ツール

命令セット定義ツールでは、独自プロセッサの命令を定義することができる。まず、学習者は独自プロセッサに用いる命令長、命令形式、命令動作などを考える。作成順序は、まずプロセッサの名前、レジスタとメモリのワード幅を 8bit,16bit,32bit から選択し、メモリの容量、汎用レジスタの個数を設定する。次にフィールドを定義する。オペコード、レジスタ、アドレスなどを定義し、命令形式を定義する。最後に命令名と動作を定義し、擬似命令を定義することもできる。ツールには制限があり、命令語長は 8bit,16bit,32bit から選択し、３オペランドまで対応、アドレス指定モードは絶対アドレス指定、レジスタアドレス指定、即値アドレス指定、PC 相対アドレス指定、インデックスアドレス指定の５つから選択し、命令動作の記述方法は固定であるなど制限がある。制限を考慮して命令セット定義を行う。命令セットを定義すると学習者が考えた命令名と内部での動作をまとめたものを一覧にし、一つのファイルを生成する。このファイルは次の命令セットシミュレータを用いてシミュレーションを行うときに用いる。[2]

(2)　命令セットシミュレータ

表 1：命令セットシミュレータのコマンド

コマンド	引数	意味
load	file name	命令セット/プログラムの読み込み
save	file name	シミュレーション結果の作成
set	pc/bp/stop/rf/dm	レジスタ・メモリの設定
del	bp	ブレイクポイントの削除
run	none/bp/N	通常/ブレイク/N 命令実行
rst	none/pc/bp/rf/sp/im/dm	レジスタ・メモリのリセット
list	none/pc/bp/rf/sp/im/dm/inst	レジスタ・メモリの表示
exit		シミュレータの終了

命令セットシミュレータでは、まず学習者が考えた命令セットを用いてアセンブリプログラムを作成する。作成したアセンブリプログラムを実行して動作確認を行うことができる。このとき各命令の動作は命令セット定義ツールで出力されたファイルを基に動作する。よって命令セット定義ツールを用いて間違って定義していると、命令セットシミュレータでも間違って動作していることが確認できる。命令セットシミュレータでは、入出力、表示、コマンド入力、実行、データ管理の機能がある。入出力とは、命令セット定義ファイルとアセンブリプログラムを入力することでシミュレーション結果ファイルを出力する。全実行命令数、各命令の実行数などが出力される。表示では、アセンブリプログラムを実行する上でレジスタファイル、命令メモリ、データメモリ、プログラムカウンタなどにデ

ータを入出力するコマンドを入力した際に表示されるものである。コマンドは表 1 である。他に表示される内容はブレイクポイント、スタックポインタ、実行した命令と次に実行する命令、無限ループ回避実行命令数、条件分岐の絶対分岐または PC 相対分岐、各命令の実行回数、ステータスレジスタである。これらを見て動作を確認することで命令セット定義ファイルの修正とアセンブリプログラムの修正を行うことができる。[1]

シミュレーション順序は、まず命令セット定義ツールで出力した命令セット定義ファイルを命令セットシミュレータにロードする。さらに定義した命令セットを基に作成したアセンブリプログラムをロードして実行する。データメモリ、命令メモリ、プログラムカウンタなどをそれぞれリセットすることができ、また 1 クロック動作に対応しているため、作成した命令セット定義ファイルやアセンブリプログラムが正しく動作するかを検証することができる。このときのアセンブリプログラムは重要であり、後の独自プロセッサを Verilog HDL で記述した後のシミュレーションによる動作確認の時や、FPGA ボードに実装したときに動作を確認できるものとして用いる。

また、利点として定義した命令セットが正確に動作することを確認しないまま Verilog HDL でのプロセッサ設計をはじめると、HDL シミュレーションのときに命令セットが間違っているのか、HDL が間違っているのかわからなくなり、設計時間が余計にかかってしまう。そこで命令セットシミュレータを用いてシミュレーションすることで定義した命令セットが正確に動作することを検証し、後の HDL によるプロセッサ設計後のシミュレーションでは、HDL によるプロセッサ設計のデバッグを考えるだけで良くなることが命令セットシミュレータの利点である。

(3) 命令セットアセンブラ

さらに命令セットアセンブラでは、作ったアセンブリプログラムを機械語に出力することができる。出力されたファイルは、16 進数で表示されたアセンブリプログラムと入力したアセンブリプログラムが 1 つのファイルとして出力される。このとき条件分岐命令は絶対分岐と PC 相対分岐のどちらかで出力しているため、学習者は 16 進数で表示された分岐先をプロセッサの内部動作を考慮した上で正確な分岐先へ変更する必要がある。機械語に出力されたアセンブリプログラムは、後に Verilog HDL での独自プロセッサの設計を行った後にシミュレーションで動作確認をするときにこの機械語を用いて検証する。

(4) プロセッサモニタとプロセッサデバッガ

最後にプロセッサモニタ、プロセッサデバッガがある。プロセッサモニタとはプロセッサデバッガを用いて FPGA ボードに実装された独自プロセッサを検証するときに使用するものである。プロセッサデバッガとプロセッサを接続し、作成したアセンブリプログラムを用いてプロセッサモニタによって FPGA ボード上に実装されている独自プロセッサを検証する。プロセッサモニタとプロセッサデバッガは図 3 のような構成になっており、プロ

セッサモニタを用いて FPGA ボード上にある独自プロセッサを、各命令が FPGA ボード上でどのような動作をしているかをプロセッサモニタ上でコマンドを入力してデータを FPGA ボードへ入出力することで検証する。プロセッサモニタのコマンドが表 2 である。

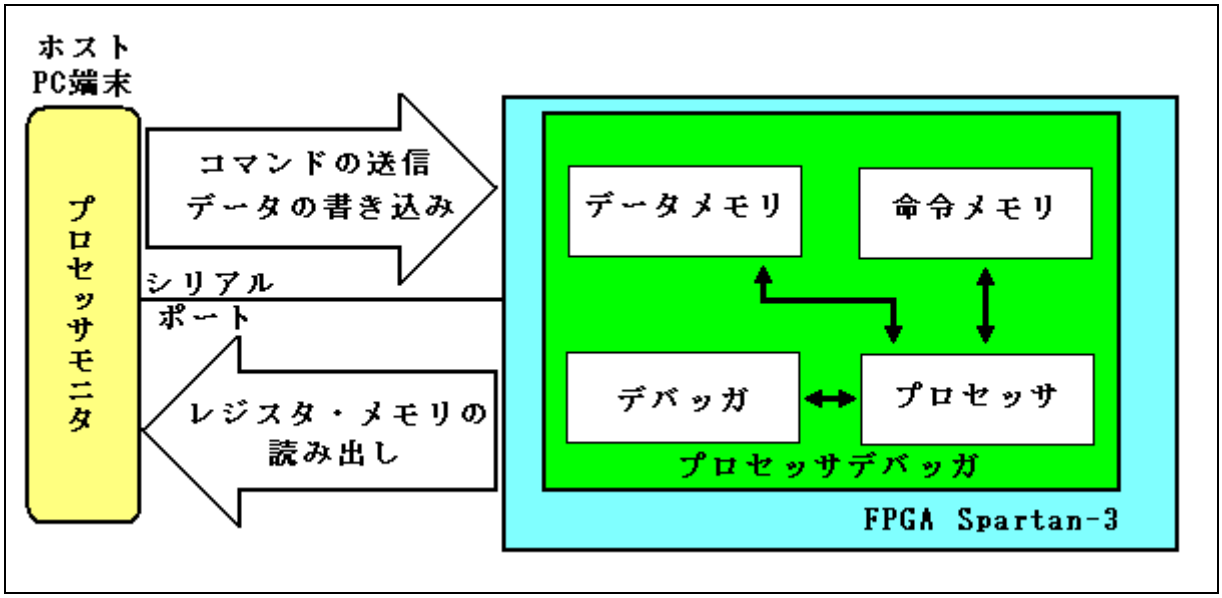

図 3：プロセッサモニタとプロセッサデバッガの構成

表 2：プロセッサモニタのコマンド

コマンド	引数	意味
write/read	dm/im/rf/pc/bp	メモリ・レジスタの書き込み/読み込み
save	dm/im/rf/pc/bp	メモリ・レジスタ内容をファイルに保存
load	dm/im/rf/pc/bp	ファイルからモニタへロード
set	rf/pc/bp/dm	レジスタ・ブレイクポイント・メモリの設定
del	bp	ブレイクポイントの削除
list/init	dm/im/rf/pc/bp	メモリ・レジスタの表示/初期化
run/run bp		通常実行/ブレイク実行
run clk N		1～Nクロック実行
halt/rst		プロセッサの停止/リセット
help/exit		コマンド内容表示/プロセッサモニタ終了

3　プロセッサ設計支援ツールを用いたアーキテクチャの定義

3.1　設計方針

本研究ではプロセッサ設計支援ツール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フラグ命令を導入し、プロセッサ設計を行った。作成したプロセッサはフラグにより計算を簡単に処理できることから、PSCSF(Processor Simply Calculable with Simple Flag)と名付けた。使用可能なフラグは ZF（ゼロフラグ）、NF（ネガティブフラグ）、CF（キャリーフラグ）、VF（オーバーフローフラグ）である。フラグの値によって絶対分岐をする。フラグを用いることで条件分岐命令の命令数を減らすことができるため、フラグを導入したプロセッサを設計した。PSCSF プロセッサの特徴を以下に示した。

- 命令長 16bit 固定
- 4 つの命令形式
- 3 オペランド命令方式
- 全部で 27 命令
- ZF、NF、CF、VF の 4 つのフラグが使用可能

3.2　命令セットアーキテクチャ

PSCSF プロセッサは、4 つの命令形式で動作する。その 4 つとは、R(Register)形式、I5(Immediate 5bit)形式、I8(Immediate 8bit)形式、J(JUMP)形式である。R 形式はレジスタ同士の計算が可能であり、I5 形式では即値演算が可能である。I8 形式では PC 相対分岐命令、レジスタファイルとデータメモリ間の転送命令がある。J 形式では、絶対分岐命令の JUMP、終了命令の HALT の他に、ゼロフラグ、ネガティブフラグ、キャリーフラグ、オーバーフローフラグを用いた絶対分岐命令がある。命令長は 16bit であり、オペコードが 5bit、レジスタ Rs が 3bit、Rt が 3bit、Rd が 3bit、ファンクションコードが 2bit と即値とアドレスによりそれぞれ構成されている。I5 形式では即値が 5bit で、I8 形式では 8bit、J 形式ではジャンプ先アドレスを 11bit で入力しなければならない。PSCSF プロセッサの命令形式をまとめると表 3 のようになる。

表 3：PSCSF プロセッサの命令形式

命令形式	5bit	3bit	3bit	3bit	2bit
R 形式	Opecode	Rs	Rt	Rd	Fn
I5 形式	Opecode	Rs	Rt	Immediate	
I8 形式	Opecode	Rs	Immediate/Address		
J 形式	Opecode	Target absolute address			

各命令形式と各命令の動作の詳細が表 4 の命令セット一覧である。この一覧を基に命令が実行されアセンブリプログラムを実行するのである。PSCSF プロセッサの特徴としてフラグを用いた命令が J 形式に８つある。

表 4：PSCSF プロセッサの命令セット一覧

命令形式	命令	活用例	説明	op	rs	rt	rd	fn
				5bit	3bit	3bit	3bit	2bit
R	ADD	ADD rd rs rt	rd = rs + rt	10000	rs	rt	rd	00
	SUB	SUB rd rs rt	rd = rs − rt	10000	rs	rt	rd	01
	SLT	SLT rd rs rt	(if rs ＜ rt) rd = 1	10000	rs	rt	rd	10
	SGT	SGT rd rs rt	(if rs ＞ rt) rd = 1	10000	rs	rt	rd	11
	SLE	SLE rd rs rt	(if rs ≦ rt) rd = 1	10001	rs	rt	rd	00
	SGE	SGE rd rs rt	(if rs ≧ rt) rd = 1	10001	rs	rt	rd	01
I5	ADDI	ADDI rt rs imm	rt = rs + imm	01000	rs	rt	imm	
	SUBI	SUBI rt rs imm	rt = rs − imm	01001	rs	rt	imm	
	SLTI	SLTI rt rs imm	(if rs ＜ imm) rt = 1	01010	rs	rt	imm	
	SGTI	SGTI rt rs imm	(if rs ＞ imm) rt = 1	01011	rs	rt	imm	
	SLEI	SLEI rt rs imm	(if rs ≦ imm) rt = 1	01100	rs	rt	imm	
	SGEI	SGEI rt rs imm	(if rs ≧ imm) rt = 1	01101	rs	rt	imm	
I8	BEQZ	BEQZ rs imm	(if rs ＝ 0) PC = imm	11000	rs	imm		
	BNEZ	BNEZ rs imm	(if rs ≠ 0) PC = imm	11001	rs	imm		
	LD	LD rs imm	rs = DATAMEM[imm]	11010	rs	imm		
	ST	ST rs imm	DATAMEM[imm] = rs	11011	rs	imm		
J	NOP	NOP	PC++	10010				
	JUMP	JUMP imm	JUMP (PC = imm)	10011	imm			
	HALT	HALT	EXIT	10100				
	BZF	BZF imm	(if ZF = 1) PC = imm	00000	imm			
	BZFZ	BZFZ imm	(if ZF = 0) PC = imm	00001	imm			
	BNF	BNF imm	(if NF = 1) PC = imm	00010	imm			
	BNFZ	BNFZ imm	(if NF = 0) PC = imm	00011	imm			
	BCF	BCF imm	(if CF = 1) PC = imm	00100	imm			
	BCFZ	BCFZ imm	(if CF = 0) PC = imm	00101	imm			
	BVF	BVF imm	(if VF = 1) PC = imm	00110	imm			
	BVFZ	BVFZ imm	(if VF = 0) PC = imm	00111	imm			

3.3　プロセッサアーキテクチャ

PSCSF プロセッサのデータパスアーキテクチャを図 4 に示す。

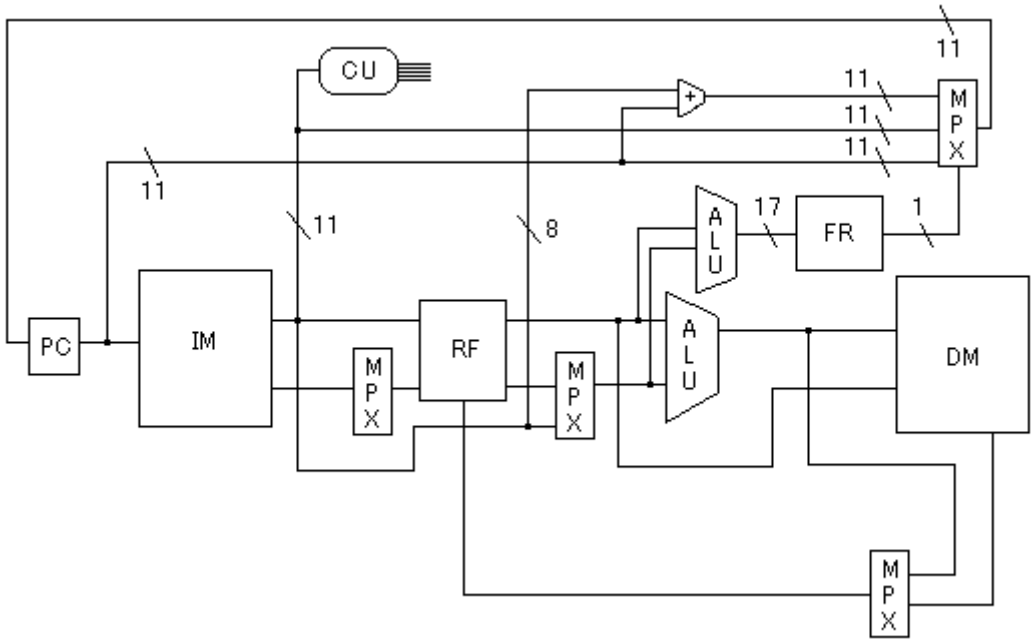

図 4：PSCSF プロセッサの回路図

図 4 の各モジュールの説明は以下である。

・PC（Program Counter：プログラムカウンタ）
　実行する命令の格納アドレスを保持しているレジスタ
・IM（Instruction Memory: 命令メモリ）
　実行する命令を保持しているメモリ
・RF（Register File：レジスタファイル）
　データを一時記憶しておくモジュール
・DM（Data Memory：データメモリ）
　データを保持しておくメモリ
・FR（Flag Register：フラグレジスタ）
　フラグのデータを保持しておくモジュール
・MPX（multiplexer：マルチプレクサ）
　複数の入力から 1 つを選択するモジュール
・ALU（Arithmetic Logic Unit：演算装置）
　2 入力 1 出力の算術演算を行うモジュール
・CU（Control Unit：コントロールユニット）
　IM からの命令によって各モジュールを制御する制御ユニット

次に各命令形式について回路図を用いて説明する。

(1) R 形式（ADD、SUB、SLT、SGT、SLE、SGE）

図 5 が R 形式の実行過程であり、レジスタを 3 つ用いて実行する R 形式では 2 つのレジスタファイルの値を ALU で演算を行うと、もう一つ指定されたレジスタファイルに書き込みを行う回路になっている。まず IM から命令が出力され、RF からデータが出力されて ALU にて演算が行われる。演算結果を RF へ再び戻して格納する。最後に PC が 1 追加されて次の命令を読み込む。また ADD、SUB での実行ではフラグ用の ALU によって 17bit で演算を行い、演算結果によって FR の ZF、NF、CF、VF に格納している。

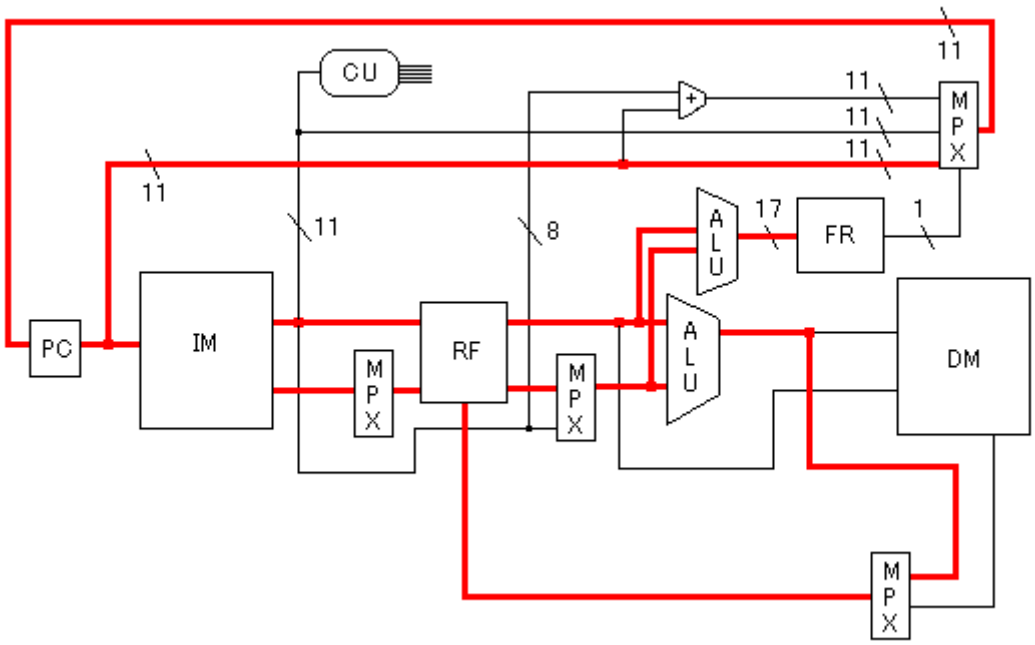

図 5：R 形式の実行過程

(2)　I5 形式（ADDI、SUBI、SLTI、SGTI、SLEI、SGEI）

図 6 が I5 形式の実行過程であり、レジスタの 2 つと 5bit の即値が指定されて実行される。まず IM から命令が出力され、RF からデータを一つ読み取る、そして ALU にて 5bit の即値と演算が行われてもう一つのレジスタの値へ出力される。最後に PC を 1 追加して次の命令を読み込む。ADDI、SUBI の演算ではフラグが発生する可能性があるのでフラグ用の ALU にて 17bit の演算を行い、演算結果を FR に出力することで FR 内でフラグをチェックし、必要があればフラグレジスタに格納する。ADDI と SUBI のときにのみ FR にそれぞれのフラグが格納されるようになっているため、ADDI、SUBI、ADD、SUB 以外のときにはフラグレジスタにデータが入力されることは無い。

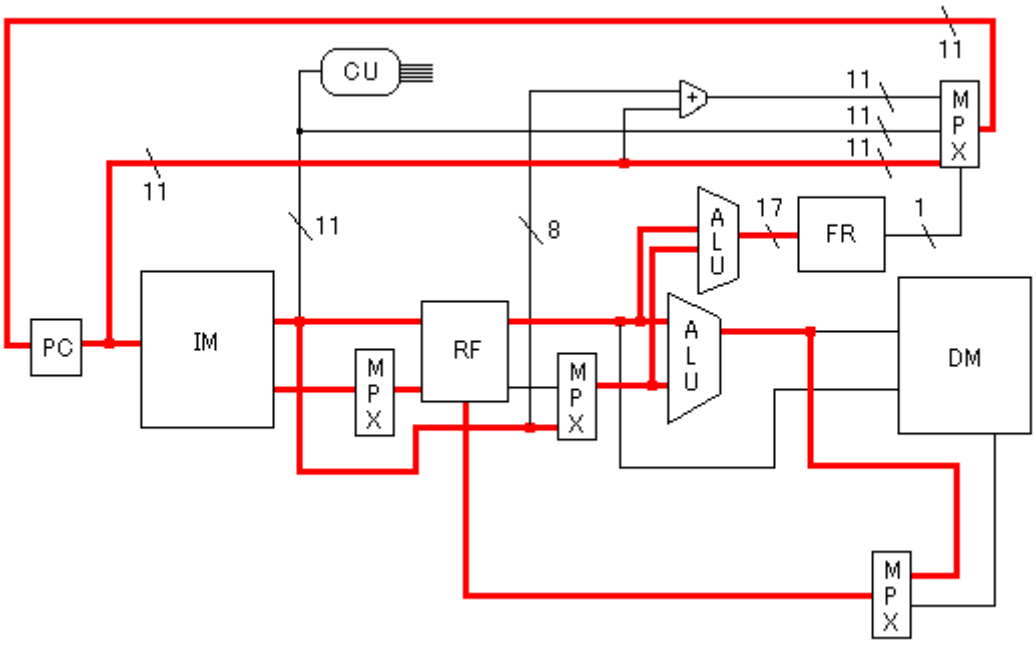

図 6：I5 形式の実行過程

(3)　I8 形式（BEQZ、BNEZ、LD、ST）

図 7 は I8 形式の BEQZ と BNEZ の実行過程であり、レジスタ内の値が 1 か 0 かによって PC 相対分岐する命令である。まず IM から命令が出力され、0 か 1 かを計るレジスタを ALU に入れる。8bit のアドレスは PC の 11bit と加算されて MUX へ入力され、ALU の出力によって MUX が PC＋1 で出力するのか、PC+アドレス 8bit の分岐先アドレスへ分岐するのかを選択する。

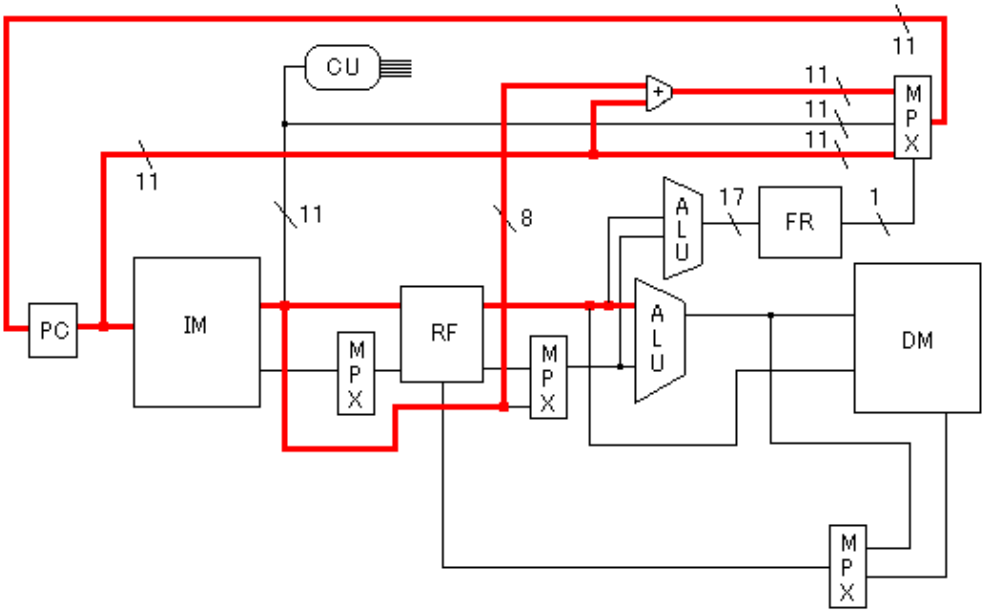

図 7：I8 形式の BEQZ と BNEZ の実行過程

図 8 は LD と ST の実行過程であり、LD と ST は PSCSF プロセッサでは特徴があり、直接アドレス指定した先のデータメモリに格納、読み込みを行う。そのためアドレス指定のレジスタのデータを読み込む必要がなく、アドレス指定用のレジスタを使う必要がない。

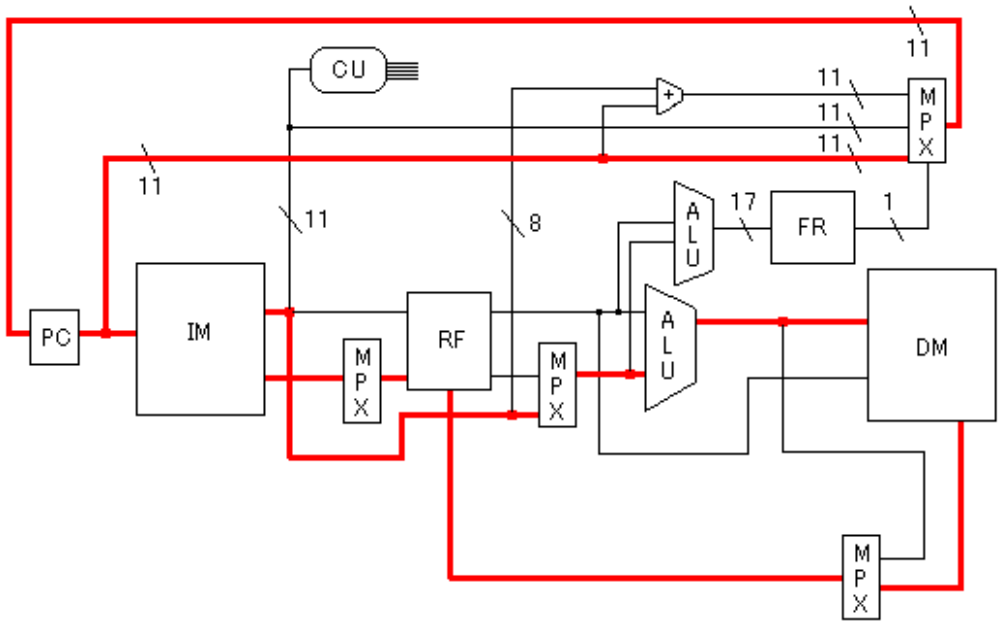

図 8：I8 形式の LD と ST の実行過程

(4)　J 形式（JUMP,NOP,HALT,BZF,BZFZ,BNF,BNFZ,BCF,BCFZ,BVF,BVFZ）

図 9 は J 形式の実行過程であり、PSCSF プロセッサの特徴であるフラグを用いた命令を実行することと、JUMP、NOP、HALT 命令を実行する形式である。IM から命令が出力され、フラグを用いた命令のときのみ、MPX は FR からデータを読み取り、ZF、NF、CF、VF のそれぞれの値によって絶対分岐先のアドレスを出力するのか、PC＋１を出力するのかを選択する。このとき、FR には読み込みだけが行われるようになっており、FR のフラグレジスタの内容が変わることは無い。それ以外の NOP 命令では PC＋１が出力され、JUMP 命令では絶対分岐の分岐先が出力され、HALT 命令は HALT 命令が出力されるようになっている。

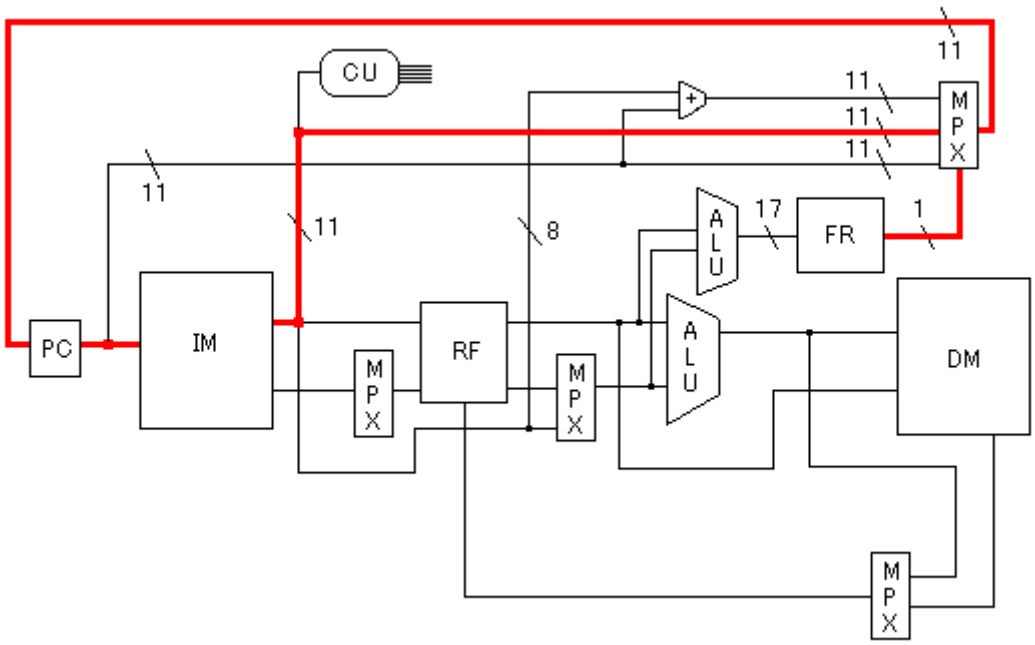

図 9：J 形式の実行過程

4　プロセッサ設計

4.1　Verilog HDL によるプロセッサ設計

PSCSF プロセッサの回路設計を Verilog HDL で行ったモジュールは PC、MUX、PLUS、RF、ALU、FR、CU、top モジュールである。これらのモジュールの一部を外部仕様の図と Verilog HDL 記述を用いて説明する。

(1)　ALU:フラグ用の演算モジュール

IM から命令が出力され、ADD、SUB、ADDI、SUBI の演算を行うときのみ演算が行われるモジュールである。図 10 に外部仕様を示す。FALU_ALU_OPE と FALU_ALU_FN にはオペコードとファンクションコードが入力されるようになっており、ADD、SUB、ADDI、SUBI 命令かどうかを判断するために入力している。ADD、SUB、ADDI、SUBI 命令のときのみ FALU_ALU_IN0 と FALU_ALU_IN1 より演算元のデータが入力されて演算結果を 17bit として出力している。

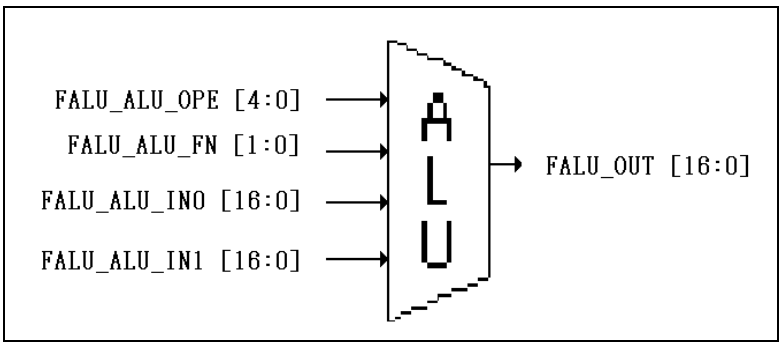

図 10：フラグ用演算モジュールの入出力仕様

(2)　FR:フラグレジスタ

フラグレジスタに読み込み信号が入ると、信号の値に応じたフラグを出力するようになっている。また、書き込み時には CU から書き込みイネーブル信号が入力され、フラグ用演算モジュールから入力された値を各フラグ別にフラグのデータを保持する仕様になっている。図 11 に外部仕様を示す。

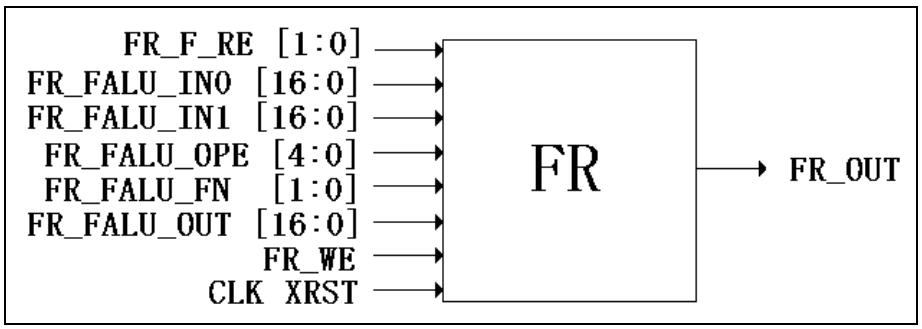

図 11：FR の入出力仕様

(3)　CU:コントロールユニット

CU とは IM から出力されたオペコードとファンクションコードが入力されると PSCSF プロセッサに入力された命令に合った動作を行うように各モジュールを制御するモジュールである。プログラムカウンタのリセット信号の制御、レジスタファイルの書き込みイネーブル信号と読み込み信号の制御、フラグレジスタへの書き込みイネーブル信号と読み込み信号、ALU へのオペコードとファンクションコードの制御、データメモリの書き込みイネーブル信号と読み込み信号の制御、各マルチプレクサの制御がある。図 12 に CU の入出力仕様と図 13 に CU の Verilog HDL 記述を示す。

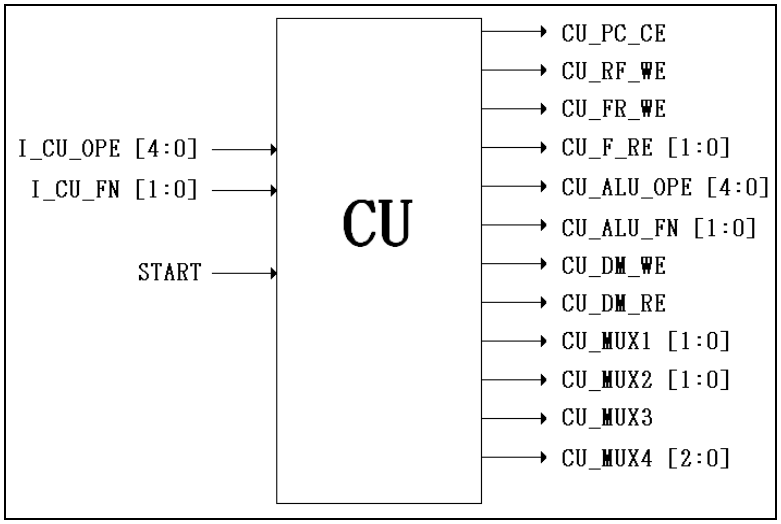

図 12：CU の入出力仕様

```
`timescale 1ns / 1ps
`define NOP     5'b10010
`define JUMP    5'b10011
`define HALT    5'b10100
`define BZF     5'b00000
`define BZFZ    5'b00001
`define BNF     5'b00010
`define BNFZ    5'b00011
`define BCF     5'b00100
`define BCFZ    5'b00101
`define BVF     5'b00110
`define BVFZ    5'b00111
          :
          :
module CU(I_CU_OPE,I_CU_FN,CU_PC_CE,CU_RF_WE,CU_FR_WE,CU_F_RE,...,CU_MUX4,START);
                                                    :
                                                    :
    //MUX4:BEQZ BNEZ Rset I5 I8 NOP JUMP
    assign CU_MUX4 = (START == 1'b1 && I_CU_OPE == `BZF) || (START == 1'b1 && I_CU_OPE == `BNF)
                  || (START == 1'b1 && I_CU_OPE == `BCF) || (START == 1'b1 && I_CU_OPE == `BVF) ? 3'b011:
                     (START == 1'b1 && I_CU_OPE == `BCFZ)|| (START == 1'b1 && I_CU_OPE == `BNFZ)
                  || (START == 1'b1 && I_CU_OPE == `BZFZ) || (START == 1'b1 && I_CU_OPE == `BVFZ) ? 3'b100:
                     (START == 1'b1 && I_CU_OPE == `JUMP) ? 3'b010:
                     (START == 1'b1 && I_CU_OPE == `Rr) || (START == 1'b1 && I_CU_OPE == `Rrr)
                  || (START == 1'b1 && I_CU_OPE == `ADDI) || (START == 1'b1 && I_CU_OPE == `SUBI)
                  || (START == 1'b1 && I_CU_OPE == `SLTI) || (START == 1'b1 && I_CU_OPE == `SGTI)
                  || (START == 1'b1 && I_CU_OPE == `SLEI) || (START == 1'b1 && I_CU_OPE == `SGEI)
                  || (START == 1'b1 && I_CU_OPE == `LD) || (START == 1'b1 && I_CU_OPE == `ST)
                  || (START == 1'b1 && I_CU_OPE == `NOP) ? 3'b001:
                     (START == 1'b1 && I_CU_OPE == `BEQZ) || (START == 1'b1 && I_CU_OPE == `BNEZ) ? 3'b000:
                     3'bx;

endmodule
```

図 13：CU の Verilog HDL 記述

オペコードとファンクションコードに合わせて MUX を制御しているのがわかる。他の MUX やフラグレジスタへの書き込みイネーブル信号と読み込み信号なども CU で記述している。また、このようにわかりやすい記述方法を用いて各モジュールを記述することで後のデバッグがしやすいように記述をしている。

また、上記のように Verilog HDL 記述を用いて各モジュールを設計し、XilinX 社提供の ModelSimXEⅢ6.3c を用いて HDL シミュレーションを行った。シミュレーションを行う際に、N までの和、最大値、素数判定、BCD 加算、一次方程式のアセンブリプログラムを用いて検証を行った。検証した結果、正しく動作していることを確認することができた。また、正しく動作することを確認し表 5 のような結果を得た。

表 5：PSCSF プロセッサのシミュレーション結果

プログラム名	実行命令数	クロックサイクル数
N までの和	24	24
最大値	11	11
素数判定	28	28
BCD 加算	40	40
一次方程式	23	23

表 5 より実行命令数とクロックサイクル数がすべてのアセンブリプログラムにおいて等しい。これは設計した PSCSF プロセッサの CPI(clock cycle per instruction)が 1 であることを指している。つまり、1 クロック内に 1 命令をこなしているのである。

4.2 FPGA ボードへの実装

PSCSF プロセッサを FPGA ボードへ実装するために論理合成を行った。論理合成ツールは、XilinX 社提供の総合開発環境 ISE9.2i を用いて論理合成を行った。その結果、表 6 のような結果を得た。また、最高動作周波数 36.814MHz、最大遅延は 27.163ns であった。

表 6：PSCSF プロセッサの論理合成の結果

	スライス数	LUT 数	フリップフロップ数
PSCSF	369	607	144
FPGA 使用率	19%	15%	3%

論理合成を行った結果、正確に動作していることが確認できた。その後 FPGA ボードに書き込みを行い、プロセッサデバッガとプロセッサを接続し、プロセッサモニタを用いてアセンブリプログラムを実行した。N までの和、最大値、素数判定、BCD 加算、一次方程式のアセンブリプログラムを実行し、BCD 加算以外のアセンブリプログラムは問題なく動

いた。アセンブリプログラムを動作させるためにプロセッサモニタの「set」コマンドを主に用いてデータメモリへデータを入力していたのだが、「set」コマンドは 9 ビット以上には対応しておらず、フラグレジスタを動作させるために 16 ビットのデータをデータメモリへ保持しておく必要があった BCD 加算は正確に動作しなかったのである。そこで、「load」のコマンドを用いて 16 ビットのデータをあらかじめ入力したファイルをデータメモリへロードして実行すると正確に動作することを確認できた。

4.3　プロセッサ設計支援ツールの評価

PSCSF プロセッサを設計する上で命令セット定義ツールでは、命令の１つ１つを手で定義する必要がなく、手順どおりに入力を行うと簡単に命令セットを定義することができる。また命令セットの定義を行う際に途中でファイルを保存する機能により、途中でファイルを保存することができる。しかし、次に途中で保存したファイルをロードするときに正確にロードすることができなかったため、27 命令ある PSCSF プロセッサの命令を最後まで失敗せずに入力する必要があった。MONI プロセッサでは倍以上の 70 命令あるため命令を定義するだけで大幅に時間がかかってしまう。今回の評価の結果より、あらゆるプロセッサの命令セットを定義できるようにするためであることと、失敗をしても途中から始められるように、命令セット定義ツールの途中で保存したファイルを正確にロードできる機能を追加させる改善が行われた。ツールの改善によってプロセッサ設計時間の短縮につながると考えられる。

命令セットシミュレータでは、命令セット定義ファイルとアセンブリプログラムを読み込み、データメモリ、命令メモリ、レジスタなどの動作が詳しく表示されるので使いやすく、またデバッグを簡単に行うことができるツールであった。PSCSF プロセッサはフラグを用いたプロセッサなので、フラグレジスタの値が表示されていることはさらにデバッグを簡単に行うことができるツールであった。しかし、各レジスタの値をリセットできるような仕様になっているのだが、１回実行すると再びアセンブリプログラムを読み込み、実行する必要があり、プログラムカウンタをリセットしても読み込んだアセンブリプログラムを再実行することができないというところは改善の必要があると考えられた。今回の評価の結果より、プログラムカウンタをリセットするだけで再実行できるように改善が行われた。またツールの改善によってさらにデバッグ時間を短縮し、プロセッサ設計時間を短縮することができると考えられる。

命令セットアセンブラでは、設計者が１つ１つの命令を手で機械語に直すことがなく、設計時間を短縮できるため良いツールであった。しかし、現在のツールでは出力されたファイルは PC 相対分岐と絶対分岐のアドレス指定先がすべて PC 相対分岐またはすべて絶対分岐のどちらかに統一されてしまうという仕様がある。今回では機械語に変換する際に 160 行あるアセンブリプログラムの中で条件分岐命令の PC 相対分岐と絶対分岐のアドレス指定先を変更する必要があり時間がかかってしまった。今後の課題としてツールに定義した

命令セットの条件分岐命令の動作を正確に読み込み、アドレス先を正確に出力するように改善する必要があると考えられる。さらにツールの改善によって機械語への変換の時間短縮になり、更なるプロセッサの設計時間の短縮につながると考えられる。

プロセッサデバッガでは Verilog HDL で記述した PSCSF プロセッサとアセンブリプログラムを接続してシミュレーションすることができる。アセンブリプログラムに原因があるのか、プロセッサに原因があるのかなど、どこで異常が起きているのかをすぐに発見することができるので使いやすく、独自プロセッサを設計する上で必要不可欠なツールである。

プロセッサモニタでは FPGA ボード上の PSCSF プロセッサとデータの入出力を行うことで、プロセッサデバッガを用いた HDL シミュレータでのシミュレーションによる検証だけでなく、FPGA ボードに実装された PSCSF プロセッサの動作を検証することができる。ソフトウェア上でのシミュレーションだけでなく、ハードウェアに実装した検証が可能であり、またプロセッサデバッガとの接続が可能であればプロセッサモニタと接続し、設計したプロセッサを検証することが可能であるため、今後も独自プロセッサを設計する上で必要なツールであると考えられる。しかし、「set」命令は 8 ビットまでしか対応していないため、データメモリへ 9 ビット以上のデータを入力する場合にはデータメモリへロードするために、あらかじめ入力したいデータを入力させたファイルを作成しておく必要があった。今回の評価の結果より、「set」命令は 16 ビットまで対応できるような機能に改善が行われた。今後の更なるツールの改善によって、より便利なツールとなりプロセッサ設計時間の短縮につながると考えられる。

5　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たプロセッサ設計支援ツールを用いて独自プロセッサ PSCSF プロセッサの命令セットを定義し、命令セットシミュレータによってシミュレーションを行った。さらに Verilog HDL を用いてプロセッサ設計を行い、HDL シミュレーションを行って、FPGA ボードに実装を行って動作を検証した。また、用いたプロセッサ設計支援ツールについて評価を行った。評価の結果、プロセッサ設計においてプロセッサ設計支援ツールのどれもが有効性のあるツールであった。しかし今回の評価の結果によりツールの更なる有効性の向上として、命令セット定義ツールでは途中セーブのファイルを正確にロードする改善、命令セットシミュレータではプログラムカウンタのリセットによるプログラム再実行を可能にする改善、命令セットアセンブラでは機械語への変換機能の改善、プロセッサモニタでは「set」コマンドによる動作の改善が考えられ、一部のツールで改善が行われた。

今後の課題として、プロセッサ設計支援ツールの命令セット定義ツール、命令セットシミュレータ、命令セットアセンブラ、プロセッサデバッガ、プロセッサモニタをよりプロセッサ設計に便利なツールに改善を行い、プロセッサ設計支援ツールの有効性の向上を行うことがあげられる。

謝辞

本研究の機会を与えてくださり、また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、貴重な助言を頂きました志水建太氏、井手純一氏をはじめ、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] 志水建太：プロセッサ設計教育のための命令セット・スーパースカラシミュレータの試作と評価、立命館大学理工学研究科修士論文、2009.

[2] 志水建太，井手純一，山崎勝弘：プロセッサ設計教育における命令セット定義ツールと命令セットシミュレータの試作、情報処理学会関西支部、支部大会講演論文集、2008.

[3] 志水建太：プロセッサアーキテクチャ学習のためのスーパースカラシミュレータの開発、情報処理学会、第 71 回全国大会講演論文集、2009.

[4] 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.

[5] 井手純一：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、立命館大学理工学部電子情報デザイン学科卒業論文、2007.

[6] 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.

[7] 難波翔一朗：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.

[8] 難波翔一朗：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.

[9] 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.

[10] 大八木睦，池田修久，山崎勝弘，小柳滋：ハード/ソフト・コラーニングシステムにおけるアーキテクチャ選択可能なプロセッサシミュレータの設計、情報処理学会第 66 回全国大会論文集、2004.

[11] 池田修久，中村浩一郎，大八木睦，Hoang Anh Tuan，山崎勝弘，小柳滋：ハード/ソフト・コラーニングシステムにおける FPGA ボードコンピュータの設計， 情報処理学会第 66 回全国大会論文集、2004.

[12] 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.

[13] 池田修久：ハードウェア記述言語による単一サイクル/パイプラインマイクロプロセッサの設計、立命館大学理工学部情報学科卒業論文、2002.

[14] David A.Patterson，John L.Hennessy 著，成田光彰 訳：コンピュータの構成と設計第 3 版 (上)(下)，日経 BP 社，2006.

[15] 桜井至：HDL 設計入門 改訂版，テクノプレス，2000.

[16] 並木秀明、前田智美、宮尾正大：実用入門ディジタル回路と Verilog-HDL、技術評論社、1996.

[17] IT 用語辞典 e-Words ： http://e-words.jp/.