修士論文

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

氏　　名	:	志水　建太
学籍番号	:	6162070066-4
指導教員	:	山崎　勝弘　教授
提 出 日	:	2009 年 2 月 13 日

立命館大学大学院　理工学研究科　創造理工学専攻

内容梗概

本研究では，ハードウェアとソフトウェアの両面の知識を持つ人材育成を目的に，命令セットシミュレータとスーパースカラシミュレータの設計と試作を行い，本研究室で開発しているハード/ソフト協調学習システムに実装した．本論文では，学習者が定義した命令セットの評価やアセンブリプログラミングを行うことができる命令セットシミュレータと，2 命令同時実行のシングルサイクルアーキテクチャの理解とハザードの発生を回避するようなアセンブリプログラミング能力を身に付けることができるスーパースカラシミュレータについて述べる．さらに，これまでに行ってきたシステムを利用したプロセッサ設計実習から，プロセッサ設計支援ツールやシステム全体の評価について述べる．

命令セットシミュレータは，コマンドユーザインタフェースで設計し，データのセットや通常実行，ブレイク実行，N 命令実行などのシミュレーションコマンドを用いることによって，アセンブリプログラムのデバッグができる．アセンブリプログラムの記述には算術演算や等号演算など 70 命令が用意されており，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，インデックスアドレス指定，PC 相対アドレス指定と組み合わせると，205 パターンの記述ができる．MONI 命令セットや，学生が設計した SOAR，SARIS，SCSFP 命令セットを用いた複数のプログラムのシミュレーションも正しく動作した．

スーパースカラシミュレータは，グラフィカルユーザインタフェースで設計し，マウス操作による実行やデータパスの表示によって，スーパースカラプロセッサの学習ができる．シングルサイクルプロセッサでは，9 個のプログラムを用いてシミュレーションを行い，全て正しく動作した．パイプラインプロセッサでは，データハザードや制御ハザードが発生した場合のフォワーディングとストール，条件分岐命令の遅延分岐など，どういう動作をするのか検討した．

命令セット定義ツールと命令セットシミュレータを用いて，独自の命令セット設計やアセンブリプログラムの作成ができた．ハード/ソフト協調学習システムを用いて，学生が複数のアーキテクチャの設計や独自プロセッサの設計も行っており，プロセッサ設計支援ツールを用いることによって，プロセッサ設計時間の短縮もできた．命令セットシミュレータを用いてアセンブリプログラムを作成することで，さらにプロセッサ設計時間の短縮ができた．

目次

図目次

表目次

1. はじめに

近年の半導体の高集積化に伴って，LSI の軽量化，高速化，省電力化，高機能化や多機能化が可能になった．テレビ，デジタルカメラ，携帯電話，自動車や自動販売機などの組み込み機器の普及がどんどん広がっていくなかで，開発者には開発期間の短縮化が求められ，多様化したニーズに合わせて様々な機能を実現しなければならない．組み込みシステムの開発において，性能や資源に制約がある中でハードウェアとソフトウェアの協調設計を進めていくには，両面からシステム全体を考えて設計していく人材が必要になってきた．この問題を解決するためには，早期の教育が必要である．ハードウェアとソフトウェア両方の知識を持つ人材を育成するために，大学での計算機アーキテクチャ教育の重要性が大きくなってきた．慶應義塾大学の PICO^2，広島市立大学の City-1，九州工業大学の Super KITE など，教育用マイクロプロセッサを用いたシステムの開発が多くの大学で行われている[1]-[7]．また，プロセッサ設計において，命令の追加や構成をカスタマイズすることが多くなってきたことで，プロセッサにおける命令セットとアーキテクチャの知識も必要不可欠になってきた．プロセッサアーキテクチャについても，処理を高速化させるために，スーパーパイプライン，スーパースカラ，マルチコアなど考案されてきた．これらのアーキテクチャや技術を理解することでプロセッサの発展に対応し，また，要求を満たせる組み込みシステムの開発を行っていかなければならない．

これらの背景から，本研究室では，プロセッサ設計を通してハードウェアとソフトウェア両方の知識を持つ人材を育成するために，ハード/ソフト協調学習システム（Hardware / Software Co-learning System : HSCS）の開発を行ってきている[8]-[30]．また，本システムに，プロセッサ設計時間の短縮を目的としてプロセッサ設計支援ツールの設計と実装を行った[14][27]．本システムの特徴は，プロセッサアーキテクチャを段階的に学習でき，学習者が独自の命令セットを定義して，そのプロセッサを設計できることにより，ハードウェアとソフトウェアのトレードオフを理解できることである．本研究室では，2004 年度から 4 回生の卒業研究の導入実習として，本システムを利用した MONI アセンブリプログラミングや，プロセッサ設計と実機上での検証を行い，システムの評価を行っている[15][28][29][30]．

現在のシステムでは，学習者はシングルサイクル，マルチサイクル，パイプラインアーキテクチャの学習ができ，設計したプロセッサを FPGA 上で検証ができる．本研究では，学習者がよりソフトウェアとハードウェアについて学習を進めるように，学習者が考案した独自プロセッサの設計時間を短縮し，命令を並列実行するスーパースカラプロセッサの学習を目的として，二つのシミュレータの設計と試作を行う．与えられた課題をこなすのではなく，学習者自身が考えてプロセッサを設計することで，学習効果はより大きくなる．そこで一つ目は，学習者が独自の命令セットを定義し，そのアセンブリプログラムのデバッグと命令セットの評価を行える，命令セットシミュレータ[16]である．本シミュレータを用いることで独自の命令セットを設計し，そのプロセッサの設計を行うことで，ハードウェアとソフトウェアのトレードオフの学習をより行うことができる．二つ目は，学習者が

並列実行するスーパースカラプロセッサについて理解し，ハザードの発生をおさえるアセンブリプログラミングを学習する，スーパースカラシミュレータである．本シミュレータを使用することで，どのようなアセンブリプログラムが効果的かを考える力や，プロセッサアーキテクチャについて理解を進めることができる．さらに，これまで行ってきたハード/ソフト協調学習システムを利用したプロセッサ設計実習の結果から，プロセッサ設計支援ツールとシステム全体の評価，及び本研究で試作したシミュレータの評価を行う．

本論文では，第 2 章でハード/ソフト協調学習システムとプロセッサ設計支援ツールについて説明する．本研究で試作を行った，命令セットシミュレータを第 3 章で，スーパースカラシミュレータを第 4 章でそれぞれ詳しく述べる．最後に，これまで行ってきたプロセッサ設計支援ツールやハード/ソフト協調学習システムの評価について第 5 章，第 6 章で述べる．

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，学習者がソフトウェアとハードウェアの学習を通してプロセッサ設計を行い，ソフトウェアとハードウェアについての知識を習得しながら両面のトレードオフを理解していくことを目的とした教育システムである．本システムは，本研究室で MIPS のサブセットとして定義した MONI 命令セットを用いたアセンブリプログラミングとプロセッサアーキテクチャの理解，さらに，プロセッサ設計を行うプロセッサ学習システム，及び学習者が独自に命令セットを定義でき，設計したプロセッサを FPGA ボード上で検証できるプロセッサ設計支援ツールから構成される．

図 1 に本システムの学習フローを示す．学習者は，3 つのステップで学習を進めていく．

(1) ソフトウェア学習

サンプルプログラムや MONI 命令セットを用いたアセンブリプログラムを作成し，MONI シミュレータで実行する．MONI シミュレータでは，シングルサイクル，マルチサイクル，パイプラインアーキテクチャについて学習ができる．プログラムカウンタ，レジスタ・メモリ，ハザードなどのデータや，各命令の動作ごとによるデータの流れを強調表示したデータパスから，アセンブリプログラミング技術を身に付け，さらに，プロセッサアーキテクチャを段階的に学習していくことで，どのようにアーキテクチャが発展してきたか知識を習得する．

(2) ハードウェア学習

実際にシングルサイクルの MONI プロセッサを設計する．HDL シミュレータは Xilinx 社の ModelSim を用いてプロセッサのデバッグを行う．シミュレータ上と実機上での動作の違いも確認するため，FPGA ボードやプロセッサ設計支援ツールのプロセッサモニタとプロセッサデバッガを使用して実機検証し，プロセッサの完成を目指す．これにより，プロセッサ設計における基本的な能力を習得する．

(3) 独自の命令セットプロセッサの設計

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，独自の命令セットを定義する．命令セットシミュレータを用いて，定義した命令セットのアセンブリプログラミングや命令セットの評価を行う．命令セットの設計が完成すれば，命令セットアセンブラを用いて検証プログラムを作成し，プロセッサ設計を進めていく．

これらのソフトウェアとハードウェア学習を融合したシステムを使用して両面のトレードオフを学習していくことで，命令セットを理解した上でのプロセッサアーキテクチャの設計や，プロセッサアーキテクチャを理解した上でのアセンブリプログラミングと命令セットの設計能力を身に付けることができる．

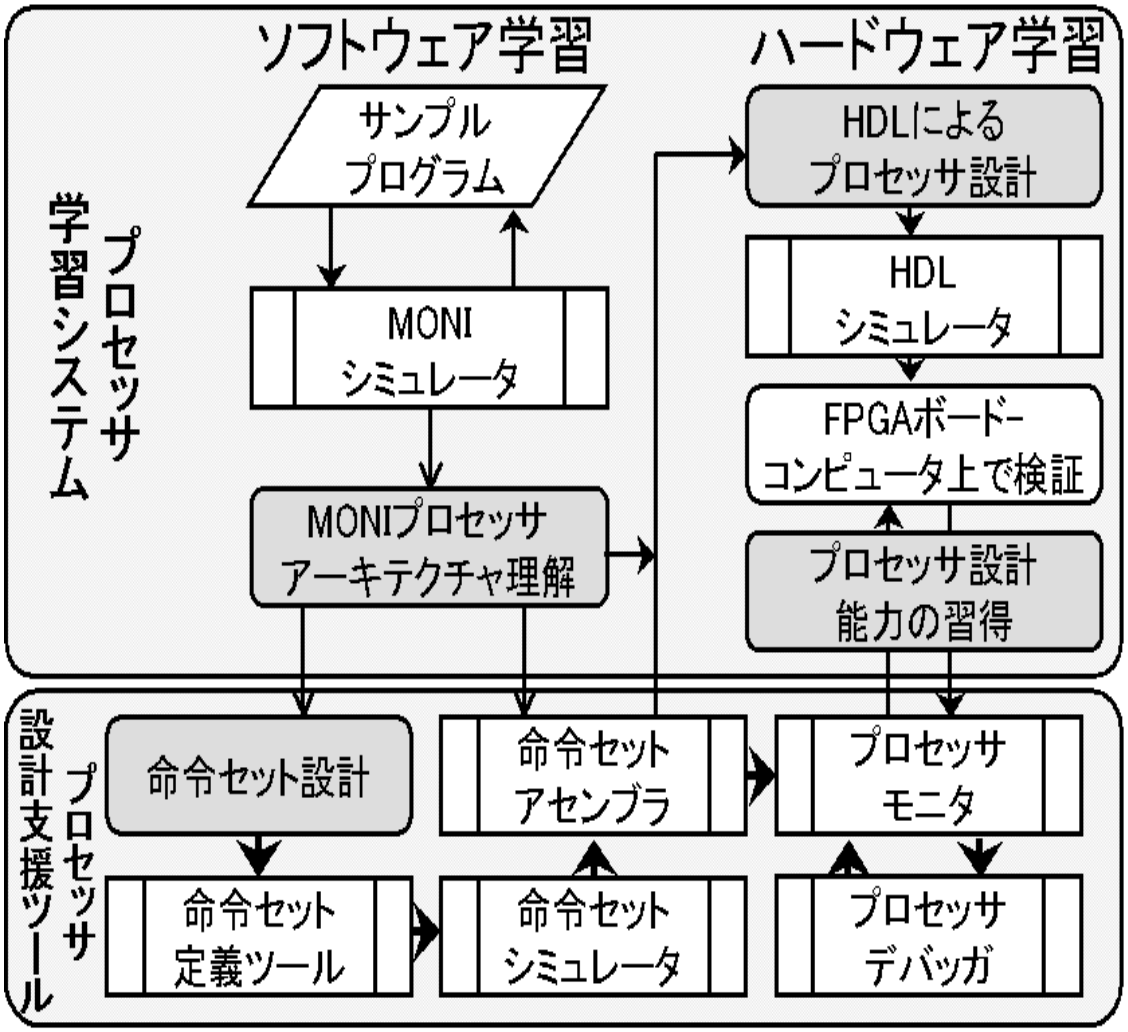

図 1：ハード/ソフト協調学習システムの学習フロー

2.2. プロセッサ設計支援ツール

学習者がプロセッサ設計をスムーズに進めていくことを目的に,プロセッサ設計支援ツールを開発した．本ツールの役割は，命令セットの定義とその命令セットのアセンブリプログラムの作成，FPGA ボード上でプロセッサのデバッグのサポートである．命令セットの定義とアセンブリプログラムの作成には，命令セット定義ツール，命令セットシミュレータ，及び命令セットアセンブラを用い，FPGA ボード上でのプロセッサ設計には，プロセッサデバッガとプロセッサモニタを用いる．

学習者が独自のプロセッサ設計をするためには，命令セットの定義と検証を繰り返し行える環境が必要であり，それらを提供するために以下の 3 つのツールを開発した．図 2 に命令セットの設計と評価からアセンブルまでのフローを示す．まず，命令セット定義ツールを用いて，命令セット設計においてビット長などの必要な項目を定義する．次に，定義した命令セットのアセンブリプログラミングを行い，命令セットシミュレータにプログラ

ムと作成された命令セット情報を入力する．最後に，命令セットの評価を行い命令セットアセンブラでアセンブルする．

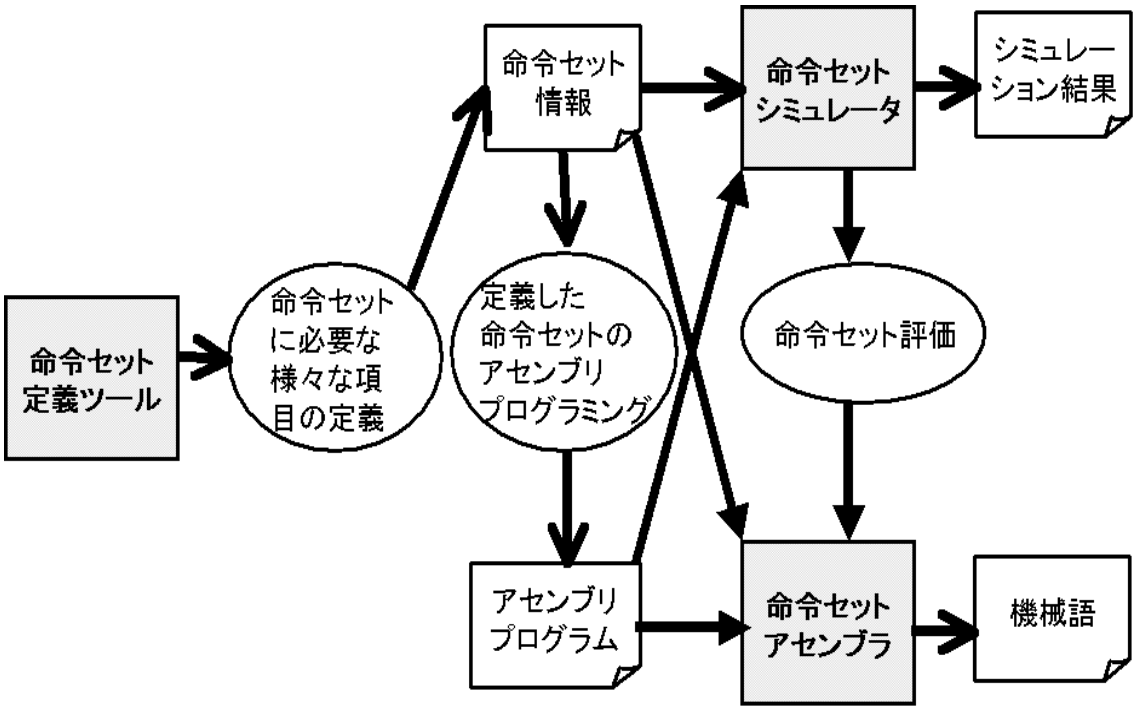

図 2：命令セットの設計と評価からアセンブルまでのフロー

(1)　命令セット定義ツール

本ツールで定義できる項目を表 1 に示す．命令セット定義ツールはこれらの情報を一括管理する．作成されるファイルは，このツールを使用することで容易に命令の追加や各項目の修正が可能であるため，スムーズに命令セットの評価を繰り返すことができる．

表 1：命令セット定義ツールで定義できる項目

項目	詳細
アーキテクチャ	命令セット名
	メモリのビット幅と容量
	レジスタファイルのビット幅と容量
	プログラムカウンタのビット幅
フィールド	フィールド名
	タイプ(定数，変数)
	ビット幅
フォーマット	フォーマット名
	フィールド群と略名
命令	命令名
	フィールド値
	引数値
	動作内容
疑似命令	疑似命令名
	命令選択
	引数値

(2) 命令セットシミュレータ

命令セットシミュレータでは，学習者が定義した命令セットのアセンブリプログラミングのデバッグを行いながら，シミュレーション結果から命令セットの評価を行う．命令セット情報のファイルを用いることで，定義した様々な命令セットを評価できる．詳細は第 3 章で述べる．

(3) 命令セットアセンブラ

命令セットアセンブラでは，命令セットシミュレータを用いてデバッグしたアセンブリプログラムのアセンブルを行う．命令セット情報のファイルを用いることで，定義した様々な命令セットのアセンブリプログラムをアセンブルできる．これにより，スムーズに独自プロセッサ設計に入ることができる．

HDL シミュレータ上で検証し，正しい動作をしたからといってプロセッサは完成ではない．実機上でも確実に動作してこそ完成といえる．そこで，実機上でも容易に検証が出来るように，プロセッサモニタとプロセッサデバッガを開発した．図 3 にプロセッサモニタとプロセッサデバッガの構成を示す．FPGA 上にプロセッサデバッガを搭載し，ホスト PC 上にあるプロセッサモニタから FPGA ボードにデバッグコマンドを送り，プロセッサの実行やデータの送受信を行ってデバッグする．

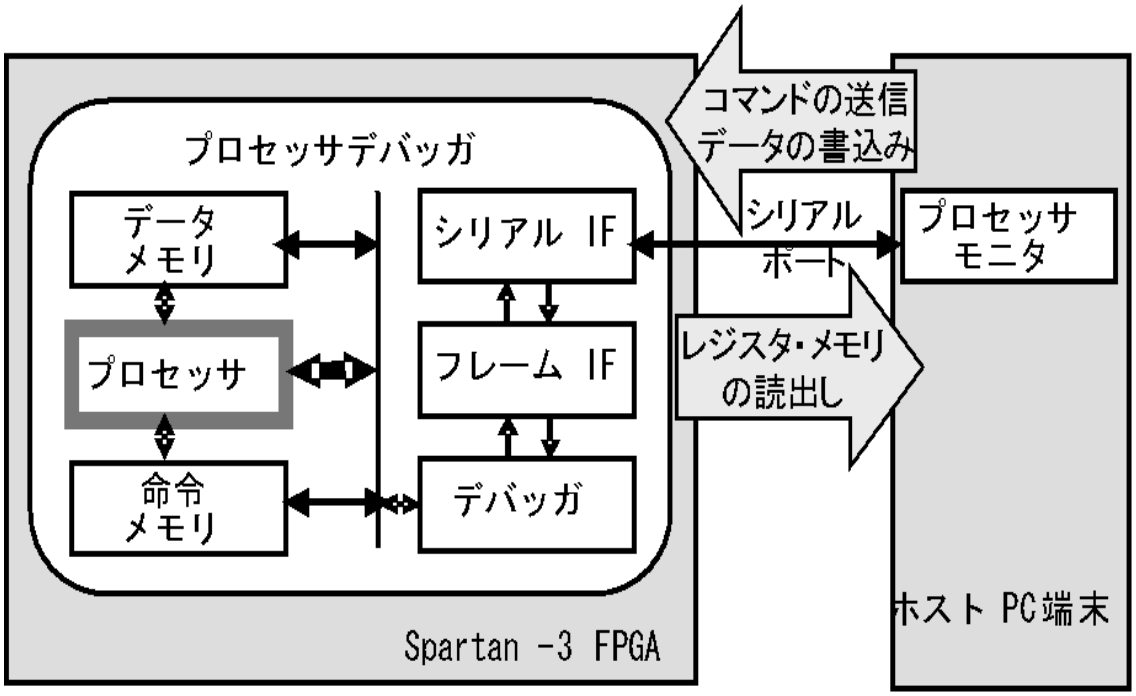

図 3：プロセッサモニタとデバッガの構成

(4)　プロセッサモニタ

プロセッサモニタは, ホスト PC 上で実機上のプロセッサの制御とデバッグを行うために, 学習者が入力したデバッグコマンドを解釈し, プロセッサデバッガへ指示を与える. また, 本ツールにより，プログラムカウンタ，レジスタ・メモリなどのデータを観測することが可能になる．表 2 にプロセッサのデバッグコマンドを示す．これらのコマンドを用いて，様々な操作を行う．

表 2：プロセッサのデバッグコマンド

コマンド	ターゲット	意味
write	dm / im / rf	モニタからデバッガへメモリ・レジスタの書き込み
read	dm / im / rf / pc	デバッガからモニタへメモリ・レジスタの読み出し
save	dm / im / rf / pc / bp	モニタのメモリ・レジスタの内容を保存
load	dm / im / rf / bp	モニタへファイルをロード
set	pc / bp / dm	モニタとデバッガのプログラムカウンタと ブレイクポイントの設定
del	bp	モニタとデバッガのブレイクポイントの削除
list	dm / im / rf / pc / bp	モニタのメモリ・レジスタの表示
init	dm / im / rf / pc / bp	モニタのメモリ・レジスタの初期化
run all		通常実行
rum clk N		N クロック実行
run bp		ブレイク実行
rst		デバッガのメモリ・レジスタの初期化
halt		プログラム実行の停止
help		コマンドの内容表示
exit		モニタの終了

dm：データメモリ，im：命令メモリ，rf：レジスタファイル
pc：プログラムカウンタ，bp：ブレイクポイント

(5) プロセッサデバッガ

学習者が設計したプロセッサをプロセッサデバッガと接続して FPGA 上へ搭載し，プロセッサモニタから送られたデバッグコマンドを解釈することで，プロセッサの実行や停止，メモリやレジスタなどのデータを更新することができる．また，設計するプロセッサのインタフェースを統一しているため，一つのプロセッサデバッガで様々なプロセッサが検証可能である．

3. 命令セットシミュレータの設計と試作

3.1. 命令セットの定義方法

命令セットシミュレータの目的は，学習者が命令形式を定義するとこから学習ができること，定義した命令セットのアセンブリプログラムのデバッグを容易に行うこと，及び，定義した命令セットの評価を行うことである．シミュレーションを行うにあたり，ある程度条件をつけることにする．条件を以下に示す．これらの条件は命令セット定義ツールで定義するものである．

(1) 命令語長は，8，16，32 ビットから選択
(2) 3 オペランドまで対応
(3) 命令動作は提供されている中から選択
(4) アドレス指定モードは 5 つから選択
    (ア) 絶対アドレス指定
    (イ) レジスタアドレス指定
    (ウ) 即値アドレス指定
    (エ) インデックスアドレス指定
    (オ) PC 相対アドレス指定

表 3 に命令セット定義で選択できる動作内容を示す．詳細は付録の表 10 に示す．全部で 70 命令選択でき，アドレス指定モードと組み合わせると 205 パターンの記述が可能である．動作内容の列は命令セット定義ツールで定義する内容である．この部分でシミュレータはどんな命令なのか判断する．$から始まるものはレジスタ，imm は即値をそれぞれ表す．LABEL は分岐先を指す。

ZF，NF，CF，VF のステータスフラグを用意する。算術演算，ビット演算，シフト演算を行った場合にセットされる．ZF は，演算結果が 0 になる場合に 1 がセットされ，NF は，演算結果の先頭ビットが 1 になる場合に 1 がセットされる．CF は，演算結果に桁上げが発生した場合に 1 がセットされる．VF は，ビットで表される数値の範囲を超えた計算の場合に 1 がセットされる．例えば 8 ビットでは-128～+127 までであり，その範囲を超えた計算結果のときである．

表 3：命令セット定義で選択できる動作内容とアドレス指定モード

種類	動作内容	アドレス指定モード
算術演算子	加算，減算，乗算，除算，剰余算	ア，イ，ウ，エ
ビット演算子	ビット反転，論理積，論理和，排他的論理和，排他的論理和の否定	ア，イ，ウ，エ
単項演算子	各桁ビットの論理積，否定論理積，論理和，否定論理和，排他的論理和，排他的論理和の否定	ア，イ，ウ，エ
等号演算子	二つの数値が等しいか等しくないか	ア，イ，ウ，エ
関係演算子	二つの数値の比較が成り立つか成り立たないか	ア，イ，ウ，エ
シフト演算子	左論理シフト，右論理シフト，右算術シフト	ア，イ，ウ，エ
データ代入	下位半分，上位半分に即値を設定	ウ
ロード，ストア	データメモリからレジスタへロード，レジスタからデータメモリへストア	ア，イ，ウ，エ
ポップ	スタックからレジスタへポップ	ア，イ，エ
プッシュ	レジスタデータをスタックへプッシュ	ア，イ，ウ，エ
コール，ジャンプ	サブルーチンへ分岐，ラベルへ分岐	ウ
リターン	サブルーチンから復帰	
条件分岐	レジスタで絶対分岐，PC 相対分岐 フラグで絶対分岐，PC 相対分岐	ウ，オ
キャリー	キャリーフラグのセット，リセット	
その他	何もしない，停止	

アドレス指定モード

(ア)　絶対アドレス指定	[imm]
(イ) レジスタアドレス指定	$rt
(ウ) 即値アドレス指定	imm
(エ) インデックスアドレス指定	[$rt + imm]
(オ) PC 相対アドレス指定	

ステータスフラグ

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

図 4 にアセンブリプログラムの記述例を示す．これらの条件を満たさないとシミュレータはエラーを出力する．

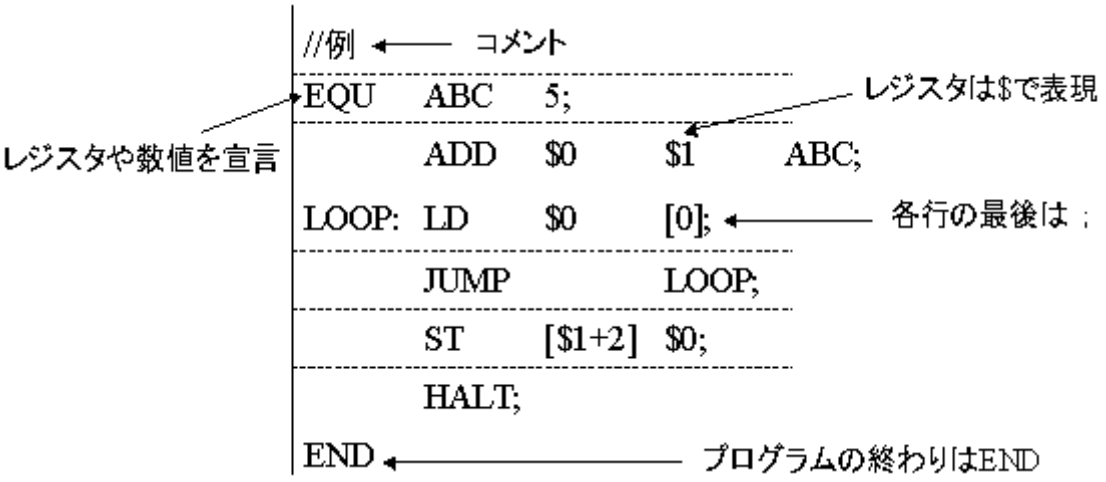

図 4：アセンブリプログラムの記述例

3.2. 機能と構成

本シミュレータの機能は，コマンド入力，入出力，表示，実行，及びデータ管理の 5 つに大別される．図 5 に命令セットシミュレータの構成を示す．コマンド入力クラスをトップとして，入出力クラス，表示クラス，実行クラス，データクラスがある．各クラスは上から，クラス名，データ，メソッドとなっている．矢印はクラスの関連を示しており，矢印の先のクラスを呼び出すことはできるが，逆はできない．まず，学習者がシミュレーションコマンドを入力する．コマンドの種類によって，実行クラス，表示クラス，入出力クラスを呼び出す．それぞれのクラスは必要に応じてデータクラスを呼び出し，データの送受信を行う．実行クラスと入出力クラスは，処理を行うと表示クラスを呼び出し，コマンド入力に戻る．

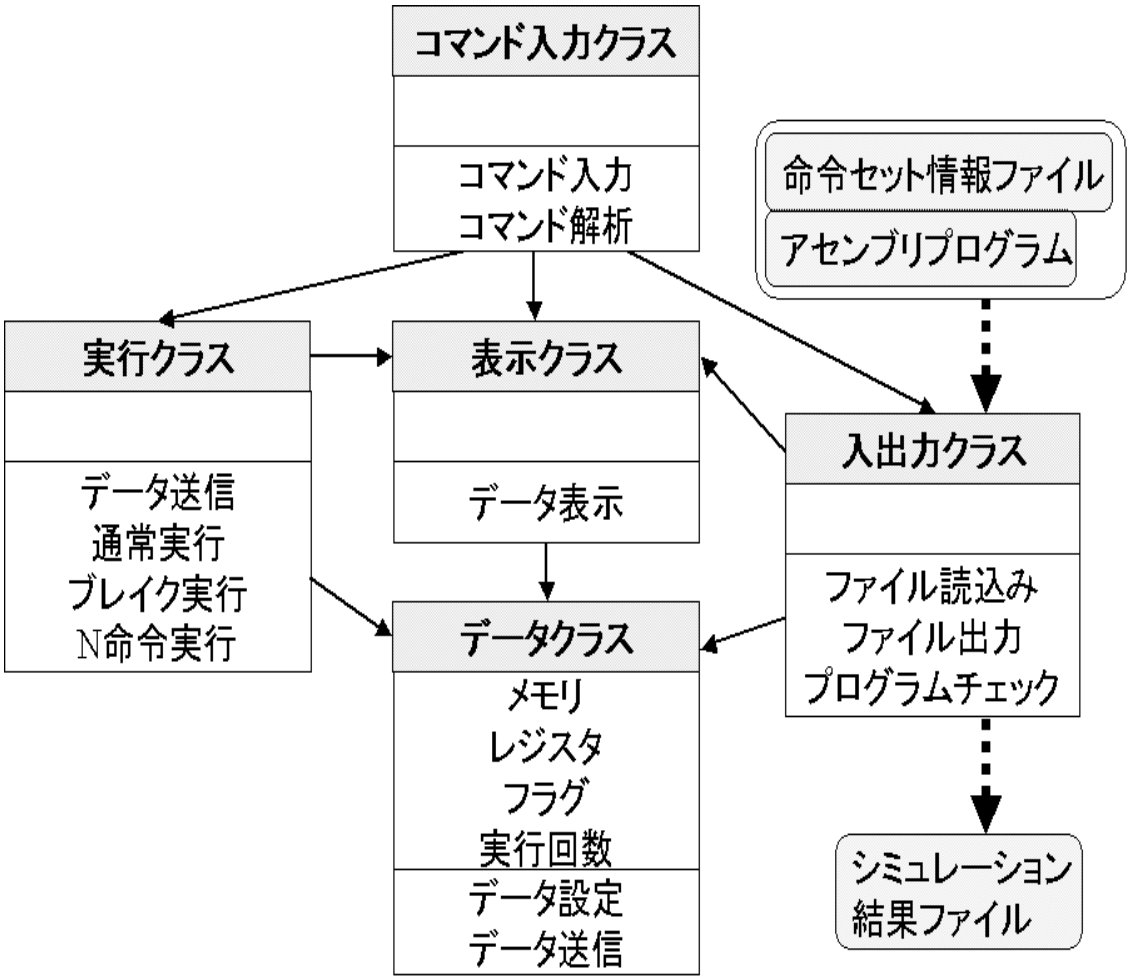

図 5：命令セットシミュレータの構成

(1)　コマンド入力クラス

学習者が入力したシミュレーションコマンドが正しいかどうか解釈し，各機能を呼び出す．シミュレーションコマンドには，ファイルアクセス，シミュレータの実行，データの設定などが用意されている．同じコマンドでも，引数を変更することでターゲットを指定できる．シミュレータ起動後，初めに命令セット情報ファイルとアセンブリプログラムを読み込ませて，シミュレーションを行っていく．表 4 にシミュレーションコマンドの一覧を示す.

表 4 : シミュレーションコマンド

コマンド	入力例	意味
load	load　xyz.txt	xyz.txt ファイルの読み込み
save	save　xyz.txt	xyz.txt(シミュレータ結果)ファイルの作成
set	set　pc　4	プログラムカウンタに 4 を設定
	set　bp　6	ブレイクポイントに 6 を設定
	set　stop　10000	最大実行命令数を 10000 に設定
	set　rf　2　6	レジスタファイルの 2 番地に 6 を設定
	set　dm　0　5	データメモリの 0 番地に 5 を設定
del	del　bp　4	ブレイクポイントの 4 を削除
run	run	通常実行
	run　bp	ブレイク実行
	run　7	7 命令実行
rst	rst	全データを初期化
	rst　pc	プログラムカウンタを初期化
	rst　bp	ブレイクポイントを初期化
	rst　rf	レジスタファイルを初期化
	rst　sp	スタックポインタを初期化
	rst　im	命令メモリを初期化
	rst　dm	データメモリを初期化
list	list	全データを表示
	list　pc	プログラムカウンタを表示
	list　bp	ブレイクポイントを表示
	list　rf	レジスタファイルを全表示
	list　sp	スタックポインタを表示
	list　im	命令メモリを全表示
	list　im　10　15	命令メモリの 10～15 番地を表示
	list　dm	データメモリを全表示
	list　dm　0　3	データメモリの 0～3 番地を表示
	list　inst	inst のデータを表示
	list　pc　rf　dm	pc rf dm のデータを表示
exit	exit	シミュレータの終了

pc：プログラムカウンタ，bp：ブレイクポイント，rf：レジスタファイル

sp：スタックポインタ，im：命令メモリ，dm：データメモリ，inst：各命令の実行回数と全実行命令数

load コマンドで，命令セット情報とアセンブリプログラムを読み込む．run コマンドで無限ループをしてしまった場合を回避するため，最大実行命令数を用意し，初期値は 5000 命令実行である．ブレイクポイントは 8 個まで設定可能とする．list コマンドを使用しなくても，pc，rf，sp，dm，実行命令，ステータスフラグ，及び条件分岐命令の場合に絶対分岐か PC 相対分岐かの値は常に表示させる．なお，rf は最大で 32 個まで表示させ，dm は 0～63 番地までと最後の番地から 10 番地目までを常に表示させる．sp は dm の最後の番地から指していく．また，im，dm は範囲の指定もできる．list の引数は複数選択できる．引数を指定しない場合は，run の場合は通常実行，rst の場合は全データの初期化，list の場合は全データを表示する．ただし，im，dm の範囲指定はできない．

(2)　入出力クラス

本シミュレータは，命令セット情報ファイルとアセンブリプログラムを読み込み，シミュレーション結果ファイルを出力する．シミュレーション結果ファイルの内容は，各命令の実行数と全命令実行数である．また，命令セット情報ファイルとアセンブリプログラムを読み込んだ際に，ファイルの記述が正しいかどうかのチェックも行う．記述のチェックは，まず分岐命令などで複数の同じ LABEL がある場合や，分岐先の LABEL がないなどのチェックを行う．次に，各命令の記述のチェックを行う．オペランドの数やレジスタの範囲内の記述など，細かくチェックを行う．エラー出力は，記述が間違っている行を出力する．

(3)　表示クラス

表示クラスでは，以下のデータを確認することができる．

・プログラムカウンタ
・レジスタファイル
・命令メモリ
・データメモリ
・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
・全命令実行数
・条件分岐命令の場合の，絶対分岐か PC 相対分岐か
・各命令の実行数
・最大実行命令数
・ステータスフラグ

これらのデータを確認しながら，プログラムのデバッグや定義した命令セットの検証を行う．条件分岐命令の場合，絶対分岐では分岐先のアドレスを表示し，PC 相対分岐では現在の PC 値からどれくらい離れているかを＋，－で表示する．

(4)　実行クラス

実行クラスでは，通常・ブレイク・N 命令実行を行う．算術演算，ビット演算，シフト演算の場合はゼロフラグ，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のセットも行う．また，データのセットやリセットも行う．まず，記述のチェックを行いながら，算術演算やビット演算などを判別し，レジスタやメモリからデータを読み出す．命令語長が 8 ビット，16 ビット，32 ビットのどれかを判別し，フラグのチェックと各命令の実行を行う．実行する際には，境界値の変化を調べ，正しい出力に調整する．また，命令の実行回数も増やす．分岐命令はプログラムの最初から分岐先のラベルが見つかるまで調べていき，プログラムカウンタの更新を行う．

(5)　データクラス

データクラスでは，表示させるデータに加え，レジスタファイルやメモリのビット幅や容量，命令の個数のデータを保持する．間違いでデータの書き換えを行わないように，一括管理する．入出力クラスや実行クラスから送られたプログラムカウンタやレジスタなどのデータの上書きや，入出力クラス，表示クラスや実行クラスなどから呼び出されたデータを出力する．

3.3.　実現方法

命令セットシミュレータはコマンドユーザインタフェースであり，開発言語は C++で設計した．図 6 に本シミュレータの使用例を示す．コマンドユーザインタフェースにすることでキーボード操作のみになり，シミュレーションコマンドを入力するだけでプログラムのデバッグなどが行え，スムーズに新たなプログラム作成やプロセッサ設計に進める．使いやすさに関しては，MONI プロセッサ設計の際に，実機上の検証でコマンド入力による操作に慣れるので，本シミュレータも容易に操作ができる．各機能をクラスとして設計していくことで，管理がしやすく C 言語より簡単に記述ができるため C++を利用した．各クラスは，メソッドを記述するだけでよく，誤ってデータを上書きすることがない．また，機能の追加も簡単に行える．アセンブリプログラムも条件を付けることで，エラーの発見が簡単に行えるようにしている．記述も MONI アセンブリプログラムに似せており，プログラミングが容易に行えるようにしている．

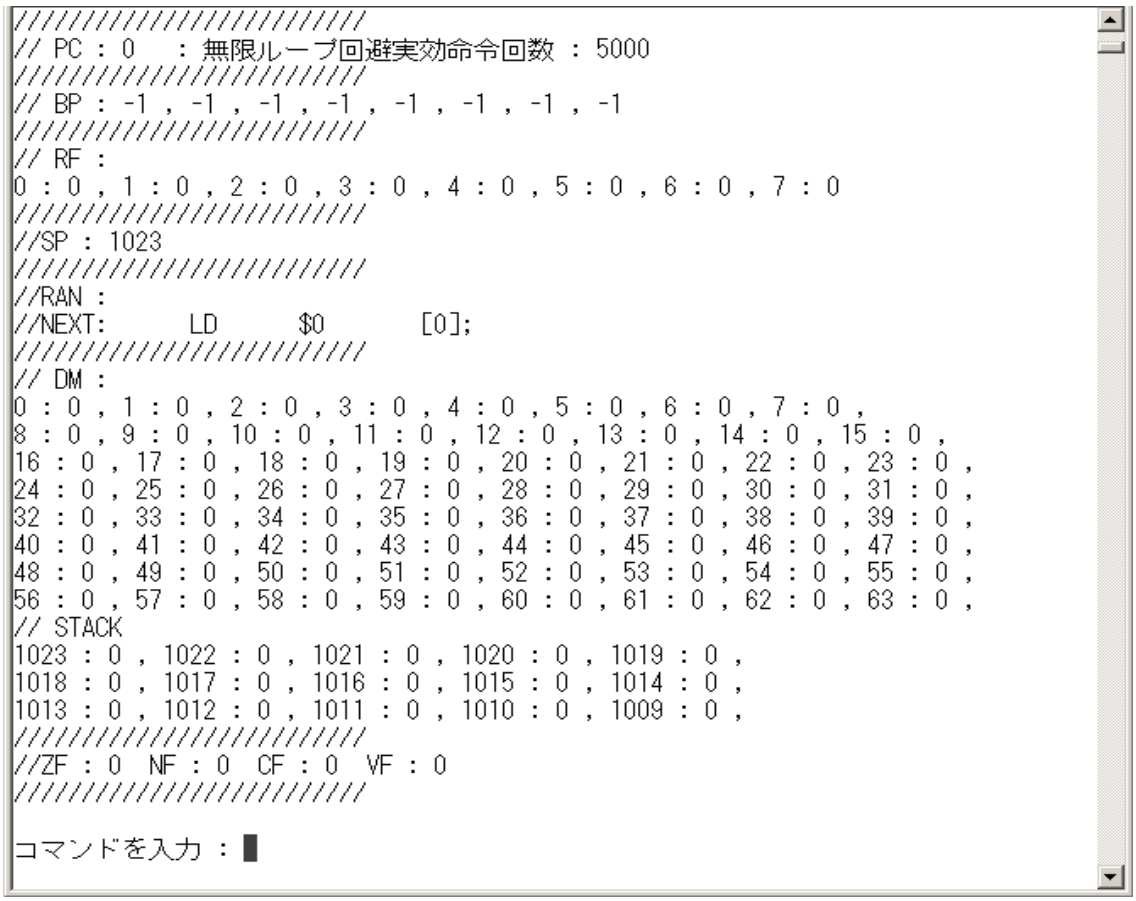

図 6：命令セットシミュレータの使用例

3.4. テスト

まず，表 3 に示した動作内容の 70 命令とアドレス指定モードを組み合わせたアセンブリプログラム全 205 パターンの記述が正しく動作することを確認した．次に，ステータスフラグの変化と，最大値や最小値の境界値の変化も正しく動作することを確認した．最後に，3 つの命令セットを用いたシミュレーションのテストを行った．使用した命令セットは，本システムで用いる MONI 命令セット[19][20], 2004 年度の四回生が設計した SOAR 命令セット[22]，2007 年度の四回生が設計した SARIS 命令セット[29]，2008 年度の四回生が設計した PSCSF 命令セット[30]である．それぞれを，表 5，表 6，表 7，表 8 に示す．各命令セットの命令語長は 16 ビット固定，3 オペランド方式で，4 つの命令形式を持つ．命令数は，MONI が 43，SOAR が 25，SARIS が 22，PSCSF が 27 である．Register 形式は，レジスタ間の演算命令を定義している．Immediate から始まる形式は，レジスタと即値の演算，条件分岐，メモリ・レジスタへのデータ転送，及びスタックポインタを使用する命令を定義している．Transfer 形式は，条件分岐とメモリ・レジスタへのデータ転送命令，JUMP 形式は，ジャンプ，スタックポインタを使用した命令，フラグを用いた条件分岐命令，及び終了命令を定義している．

表 5：MONI 命令セット

ビット長 / 命令形式	5	3	3	3	2
Register	OP	RS	Rt	Rd	Fn
Immediate5	OP	RS	Rt	Imm	
Immediate8	OP	RS	Imm		
JUMP	OP	Imm			

表 6：SOAR 命令セット

ビット長 / 命令形式	5	3	3	3	2
Register	OP	Fn	Rs	Rt	Rd
Immediate	OP	Imm		Rt	Rd
Transfer	OP	Imm			Rd
JUMP	OP	Imm			

表 7：SARIS 命令セット

ビット長 / 命令形式	5	3	3	3	2
Register	OP	Fn	Rs	Rt	Rd
Immediate3	OP	Fn	Imm	Rt	Rd
Immediate8	OP	Imm			(Rd)
JUMP	OP	Imm			

表 8：PSCSF 命令セット

ビット長 / 命令形式	5	3	3	3	2
Register	OP	RS	Rt	Rd	Fn
Immediate5	OP	RS	Rt	Imm	
Immediate8	OP	RS	Imm		
JUMP	OP	Imm			

テストプログラムは, MONI 命令セットでは 8 個のプログラム（N までの和, N の階乗, 除算，素数判定，二次方程式の根の判別，三角形の判別，一次方程式），SOAR 命令セットでは 4 個のプログラム（N までの和，最大値検出，最大公約数，バブルソート），SARIS 命令セットでは 5 個のプログラム（N までの和，N の階乗，最大公約数，除算，二次方程式の根の判別），PSCSF 命令セットでは 5 個のプログラム（N までの和，最大値検出，素数判定，BCD 加算，一次方程式）を用いた．全てのプログラムが正しく動作することを確認した．アセンブリプログラムに対して何行目がエラーかを出力するので，命令セット情報の修正やプログラムのデバッグが容易に行うことができる．

4. スーパースカラシミュレータの設計と試作

4.1. 設計方針

スーパースカラシミュレータの目的は，スーパースカラアーキテクチャと，ハザードやその対策を学習することである．本シミュレータで学習できるアーキテクチャは 2 命令を同時実行するシングルサイクルプロセッサとパイプラインプロセッサの 2 種類である．使用する命令セットは MONI 命令セットである．設計方針を以下に述べる．

(1) 2 命令同時実行のシングルサイクルプロセッサ

まず，学習者はシングルサイクルアーキテクチャについて学習する．シングルサイクルでは，2 命令を同時に処理する際に発生するデータハザードと制御ハザードについて理解する．ハザードが発生した場合のデータの流れを学習し，さらに，ハザードを発生させないようなアセンブリプログラミングの学習も行う．

(2) 2 命令同時実行のパイプラインプロセッサ

次に，学習者は本来のスーパースカラであるパイプラインアーキテクチャについて学習する．複数の命令をパイプライン処理していく際に発生するデータハザードと制御ハザードについて理解し，それぞれのハザードに対するフォワーディング，ストールや遅延分岐を行うアーキテクチャについて学習を進めていく．さらにアセンブリプログラミングの能力も高めていく．

(3) ハザードの理解とその対策

2 つのアーキテクチャでシミュレーションを行うことで，2 命令間に発生するデータハザードや制御ハザード，パイプライン処理による複数の命令間に発生するデータハザードや制御ハザードについて理解していく．シングルサイクルでは，ハザードがどのような場合に発生するのかを理解する．パイプラインでは，各ステージで実行している複数の命令間に発生するデータハザードは，フォワーディングによって解決し，メモリ読み出し命令はストールすることを理解する．1 つのステージで実行している 2 命令間のデータハザードや制御ハザードは，ストールすることを理解する．さらに，条件分岐命令は遅延分岐を行うことを理解する．

図 7 に本シミュレータの機能と構成を示す．機能は，表示，リセット，メモリ書き込み，実行の 4 つに大別される．表示では，実行した命令，プログラムカウンタ，スタックポインタ，メモリ・レジスタ，データ・制御ハザードやデータパスなど，アセンブリプログラミングとスーパースカラアーキテクチャの学習に必要な情報を表示する．リセットでは，プログラムカウンタやデータメモリなどを初期化することで繰り返しテストが行え，また，エディタに記述した内容も初期化する．メモリ書き込みでは，命令メモリやデータメモリの書き込みを行う際に，記述が正しいかチェックを行う．また，パイプラインアーキテクチャを選択時は，遅延分岐用に命令の順序を並べ替えて書き込みを行う．実行では，1 サイクル・通常・ブレイク・N サイクル実行の 4 つの実行モードがあり，データハザードや制御ハザードの検出も行う．

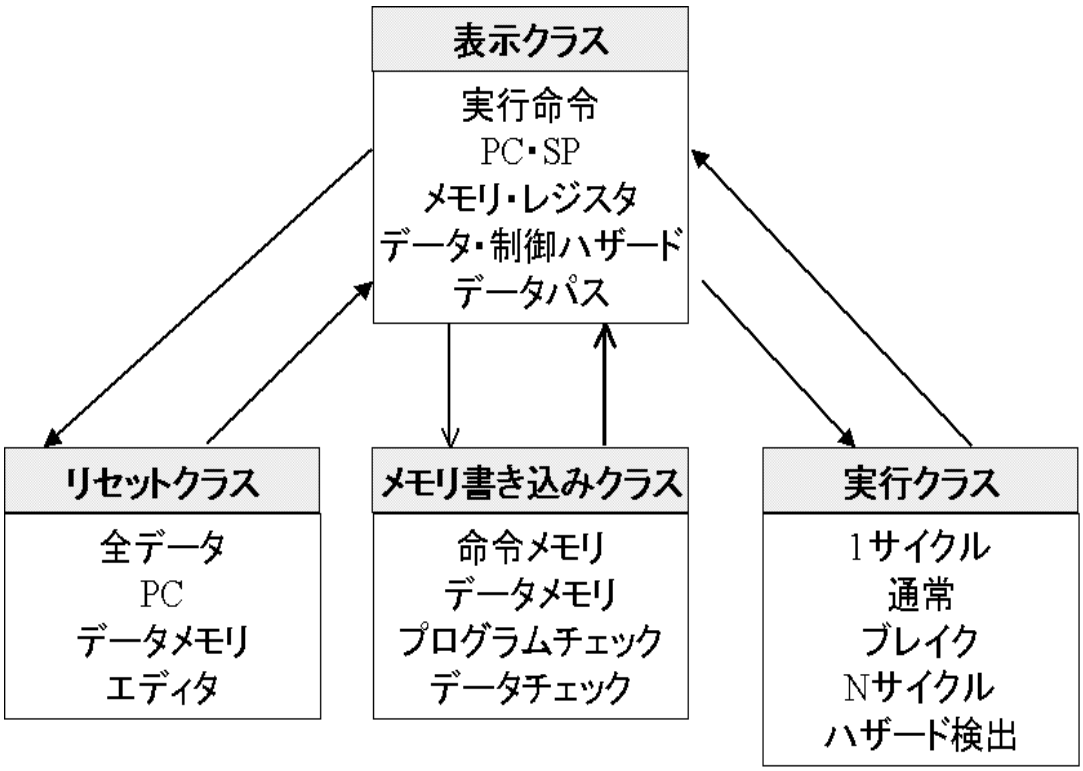

図 7：スーパースカラシミュレータの構成

コマンドの種類によって，リセット，メモリ書き込みや実行を行い，結果のデータを表示する．プログラムをエディタに書き，命令メモリに書き込む．エラーがなければ，データメモリにデータを書き込み，実行を行う．シミュレーション後は，各命令の実行回数などを確認し，プログラムや命令セットの評価を行う．学習者は，データパスが実行ごとにデータの流れを協調表示されるため，アーキテクチャについて理解を進めながら，ハザードを発生させない最適なアセンブリプログラミングの学習ができる．

図 8 に本シミュレータのインタフェースを示す．メニューバーは，ファイルの作成や保存，エディタの編集，全命令と各命令の実行回数を表示するシミュレーションデータ，及びシミュレータのバージョンを表示する．ツールバーは，メニューバーのファイルの作成や保存やエディタ編集の機能とリセットや実行のボタンが用意されている．データパスはインタフェースの画像のサイズを変更することで大きさが変えられる．

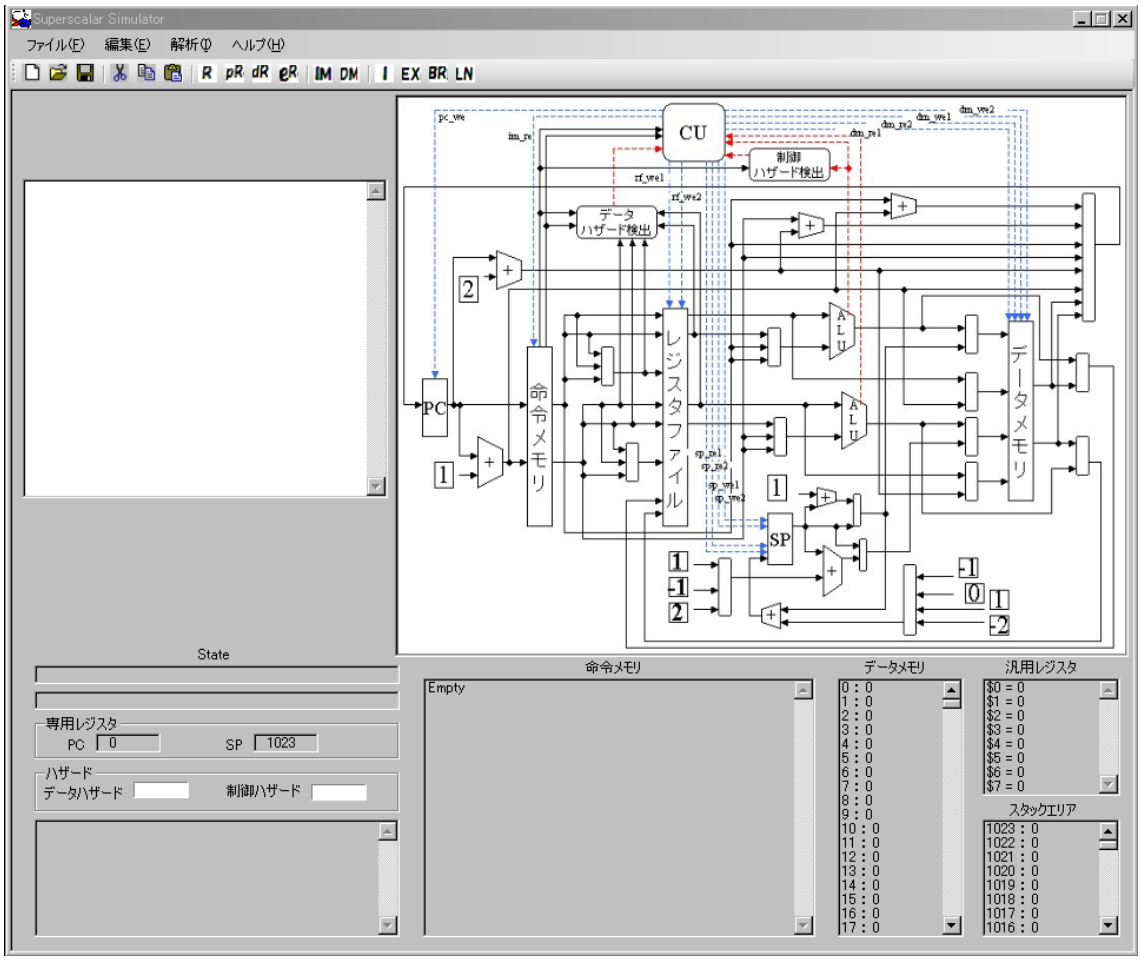

図 8：スーパースカラシミュレータのインタフェース

4.2. シングルサイクルプロセッサの動作

図 9 に 2 命令同時実行のシングルサイクルプロセッサのデータパスを示す．2 命令同時実行なので，ALU やマルチプレクサなどのモジュール，レジスタとメモリの入出力ポートなどを 2 個ずつ用意している．実線の矢印はデータの流れ，CU からの点線の矢印は書き込みや読み出しの制御信号，ハザード検出と ALU からの点線の矢印はハザード検出や条件分岐が成り立つかの信号である．

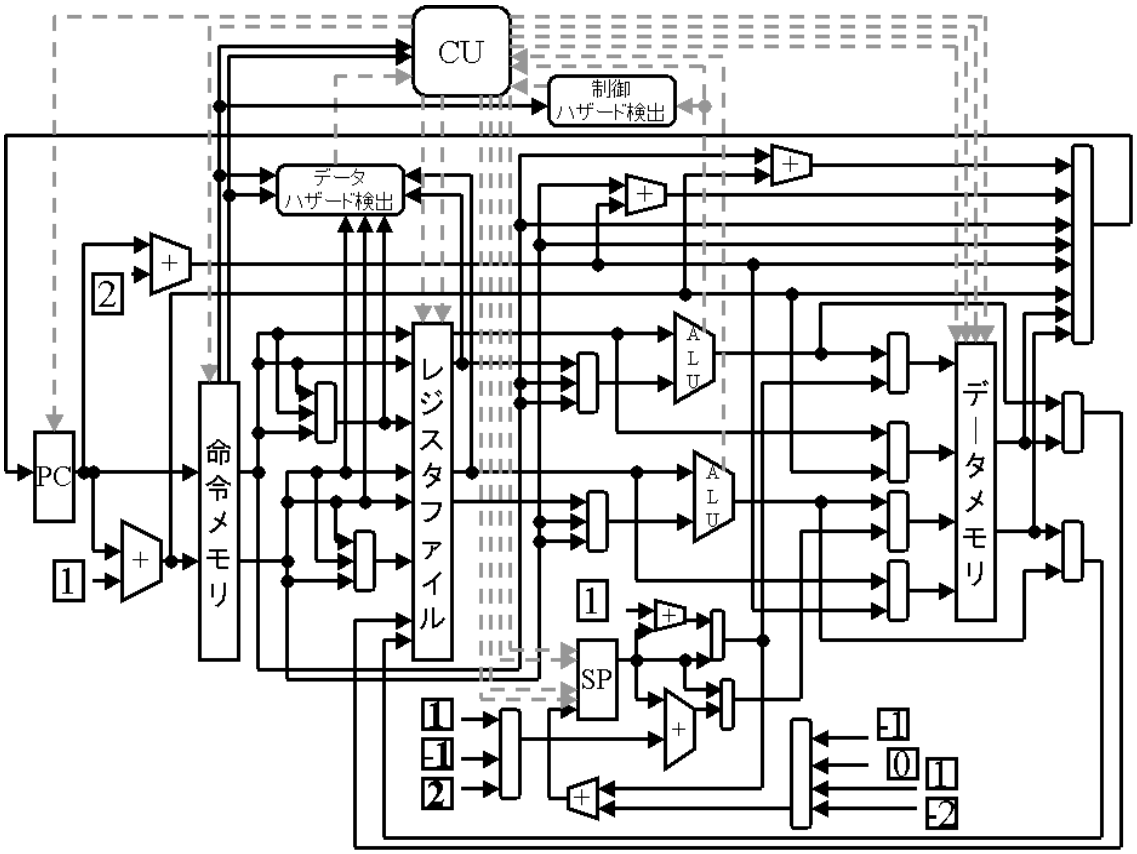

図 9：2 命令同時実行のシングルサイクルプロセッサ

シングルサイクルプロセッサでは，ハザードの発生時 1 命令目のみを実行する．ハザードが発生する状況は，1 命令目の書き込み先を 2 命令目が参照する際に発生するデータハザード，1 命令目が分岐命令の際に発生する制御ハザードである．いずれの場合も 2 命令目は実行しない．1 命令目が条件分岐の場合は 2 命令目も実行し，条件が成り立てば 2 命令目の結果を破棄する．

ハザードが発生していないデータパスを図 10 に，データハザード発生時のデータパスを図 11 に，制御ハザード発生時のデータパスを図 12 に，条件分岐命令で条件が成り立たない場合のデータパスを図 13 にそれぞれ示す．図 10 は，ADD $0 $1 $2 と LD $3 MEM[$4]命令を実行した結果である．2 命令間に依存がないためハザードが発生していない．この場合はプログラムカウンタを＋2 する．ハザードが発生しない場合は，このようなデータパスが表示される．図 11 は，ADD $0 $1 $2 と SUB $3 $0 $1 命令を実行した結果である．1 命令目の結果である$0 を 2 命令目が参照しているため，2 命令目からのデータの流れは表示しない．1 命令しか実行していないので，プログラムカウンタは＋1 する．図 12 は，BNEZ $0 LABEL と SUB $2 $3 $3 を実行した結果である．1 命令目の条件分岐の条件が成り立ったため，制御ハザードが発生した．ハザードが発生したため，2 命令目の処理は最後まで実行されず，データの流れは表示しない．図 13 は，BNEZ $0 LABEL と BEQZ $1 LABEL を実行し，2 命令

とも条件が成り立たなかったデータパスである．1 命令目が成り立たなかったため制御ハザードが発生せず，2 命令目からのデータの流れも表示されている．これらの動作を理解し，ハザードがどのような場合に発生するのかを学習する．

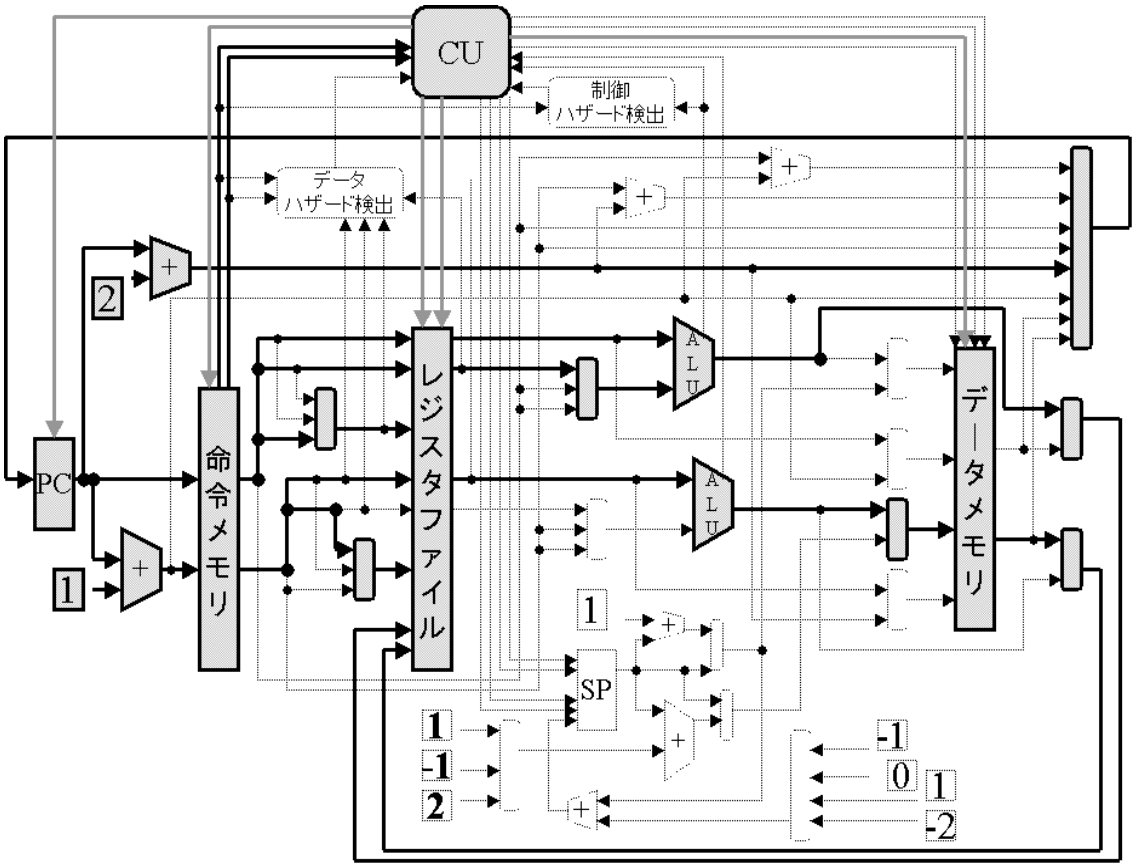

図 10：ハザードが発生していないデータパス

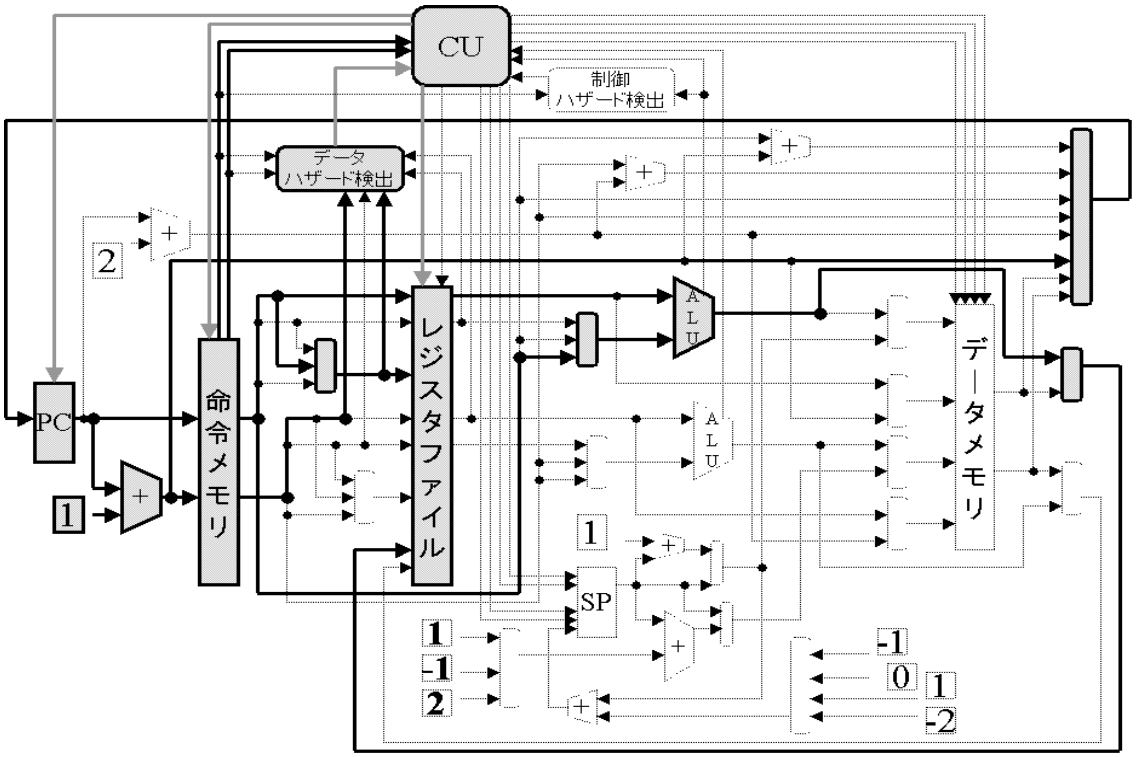

図 11：データハザード発生時のデータパス

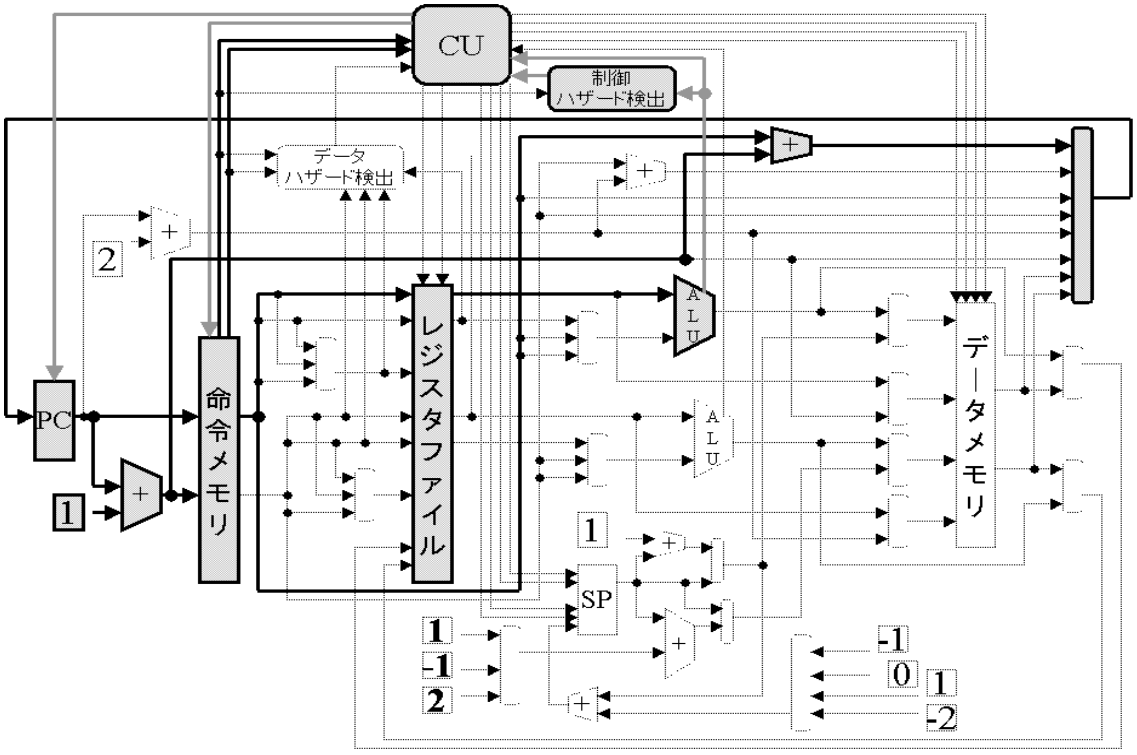

図 12：制御ハザード発生時のデータパス

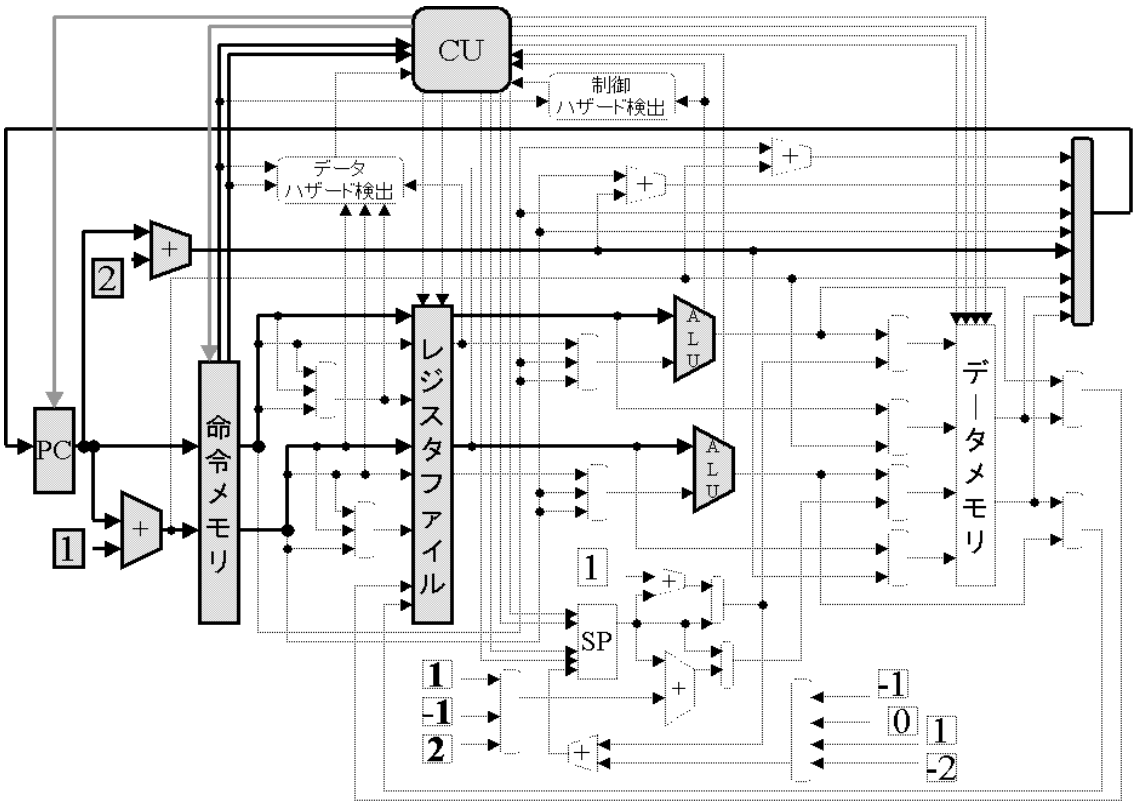

図 13：条件分岐命令のデータパス

4.3. パイプラインプロセッサの動作

図 14 に 2 命令同時実行のパイプラインプロセッサのデータパスを示す．パイプラインは 2 命令を，命令読出し，命令解読，実行，メモリアクセス，レジスタ書き込みの 5 サイクルでインオーダー実行する．ALU やマルチプレクサなどのモジュール，レジスタとメモリの入出力ポートなどを 2 個ずつ用意し，読み出し/命令解読，命令解読/実行，実行/メモリアクセス，メモリアクセス/レジスタ書き込みにレジスタを用意している．実線の矢印はデータの流れ，CU からの点線の矢印は，書き込みや読み出しの制御信号，及び各ステージへの制御信号である．ハザード検出とフォワーディングのモジュールからの点線の矢印は，制御ハザード発生やフォワーディングを行う制御信号である．

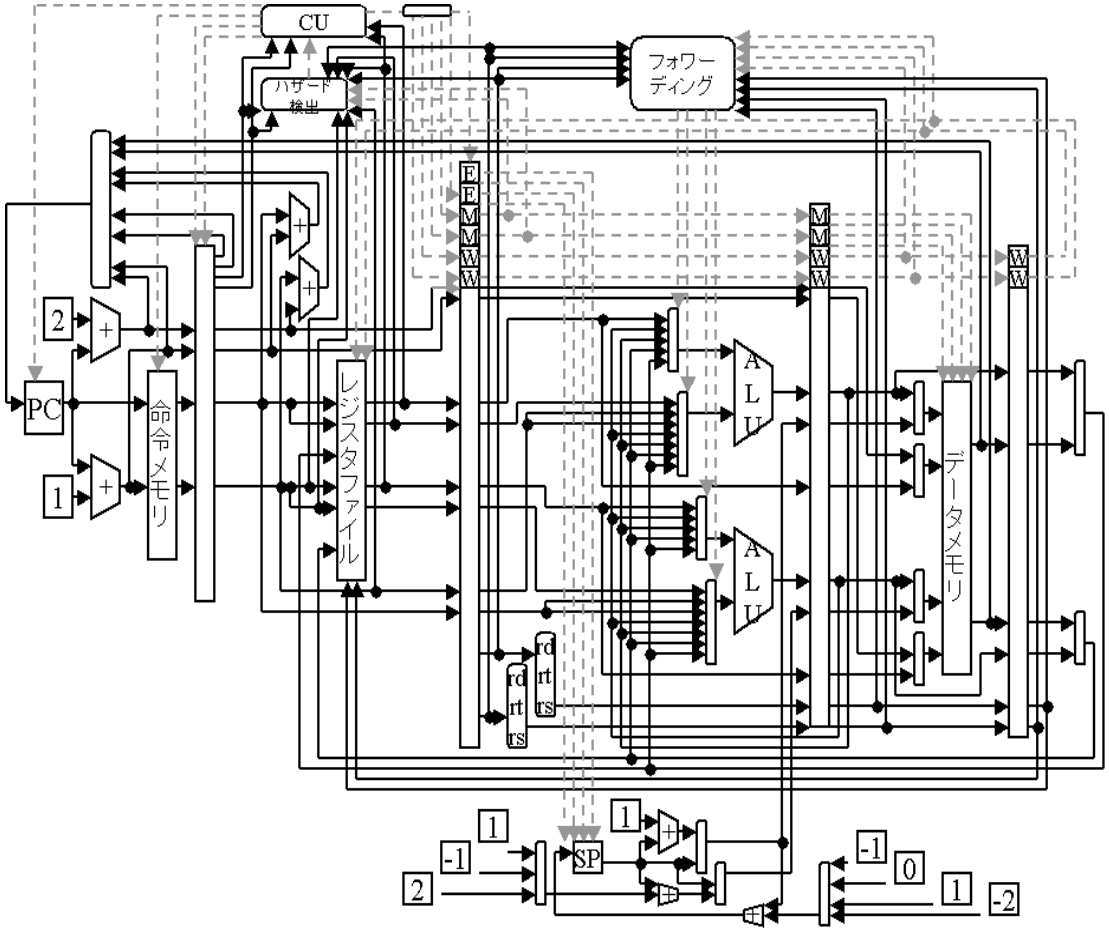

図 14：2 命令同時実行のパイプラインプロセッサ

データハザードが発生した場合は，実行とメモリアクセスのデータを利用するフォワーディングでハザードを解消する．1 命令目がロード命令で，その結果を 2 命令目や次のサイクルの命令が参照する場合は，メモリアクセスでストールする．1 命令目の結果を 2 命令目が参照する場合は，実行でストールする．制御ハザードが発生した場合は，無条件分岐命令は命令解読でストールし，条件分岐の命令は，遅延分岐を行う．

フォワーディングを行っている動作を図 15 に，LD 命令の結果を参照する場合に行うストールを図 16 に，1 命令目の結果を 2 命令目が参照する場合に行うストールを図 17 に，無条件分岐命令の場合に行うストールを図 18 に，条件分岐命令の場合に行う遅延分岐を図 19 にそれぞれ示す．図 15 は，フォワーディングによってデータハザードを解消している動作である．ADD $1 $2 $3 で計算される結果の$1 を，1 サイクル後の SUB $3 $1 $1 は実行からのフォワーディングで，2 サイクル後の SLL $3 $3 $1 はメモリアクセスからのフォワーディングで参照している．図 16 は，LD $0 MEM[$1]でデータメモリから読み出す結果の$0 を，ADD $0 $1 $0 が参照する場合にストールを行い，メモリアクセスからフォワーディングによりデータを参照している．LD $2 MEM[$2]の結果の$2 も，SUB $3 $2 $4 が参照するため，

ストールしてからフォワーディングによって参照している．図 17 は，ADD $1 $0 $1 の結果の$1 を，ADD $2 $0 $1 が参照する場合にストールを行い，実行からのフォワーディングによりデータを参照している．ADD $3 $0 $1，SUB $4 $4 $3，ADD $0 $0 $4 ではそれぞれフォワーディングによってデータを参照している．図 18 は，JUMP 命令の場合に発生する制御ハザード発生時の動作である．命令解読で分岐先のアドレスを得られるため，命令解読でストールし分岐している．図 19 は，遅延分岐を行うために条件分岐命令があるプログラムを，並べ替えて実行している．遅延スロットは 2 命令分である．BNEZ $0 LABEL が参照する$0 と ADD $3 $3 $0，ADD $2 $1 $2 の結果である$3 と$2 に依存関係がないため，条件分岐命令の下に並べ替えている．依存関係があり，遅延スロットに命令を並べ替えられない場合は，NOP 命令を入れる．これらの動作を理解することで，パイプライン処理をしていくなかでハザードが発生した場合，どのような対策を行うのかを学習していく．

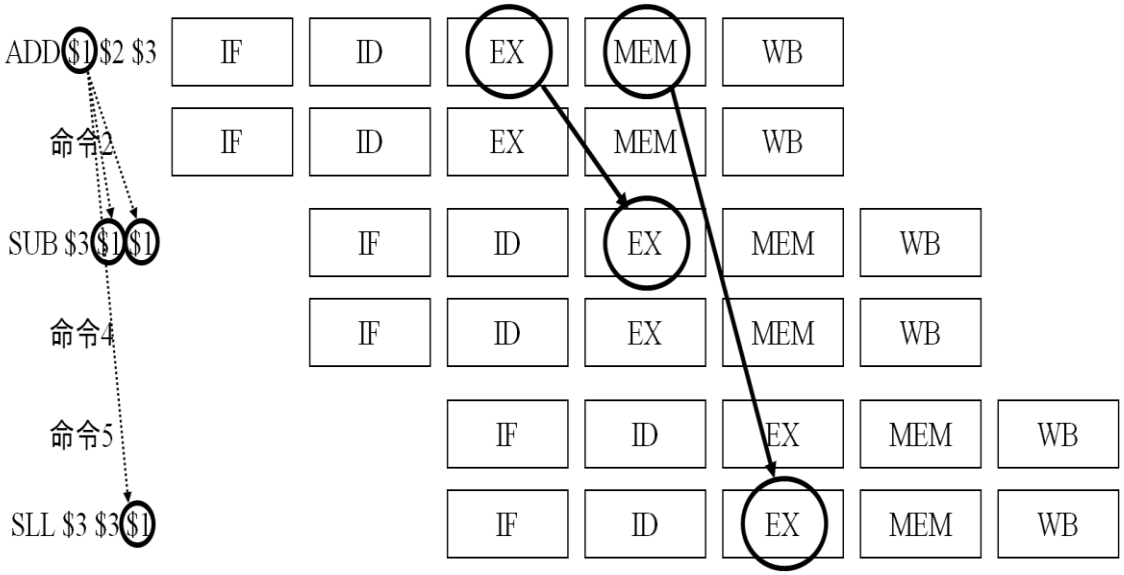

図 15：EX と MEM からのフォワーディング

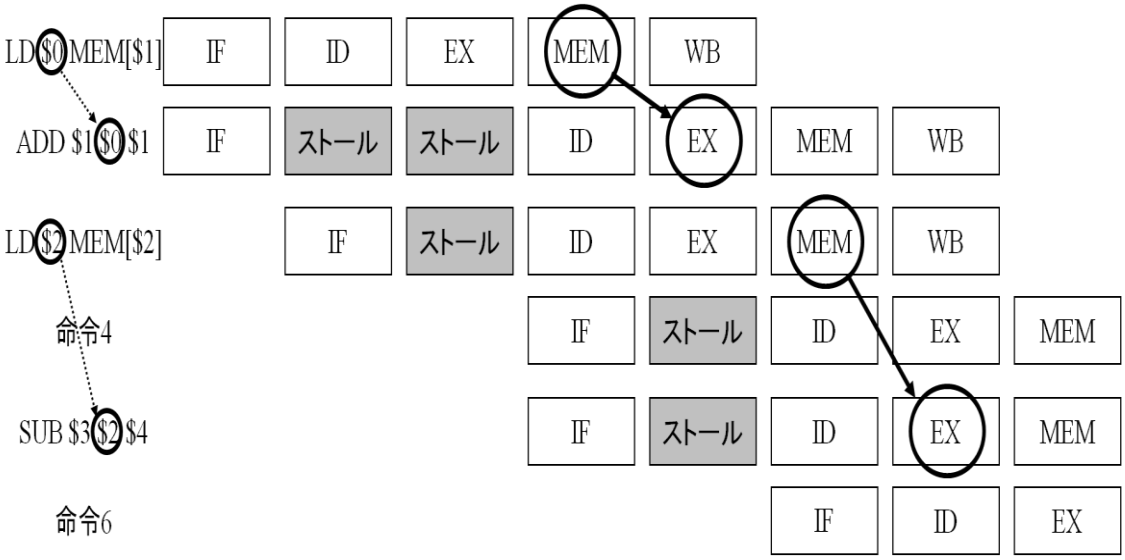

図 16：LD 命令の結果を参照するストール

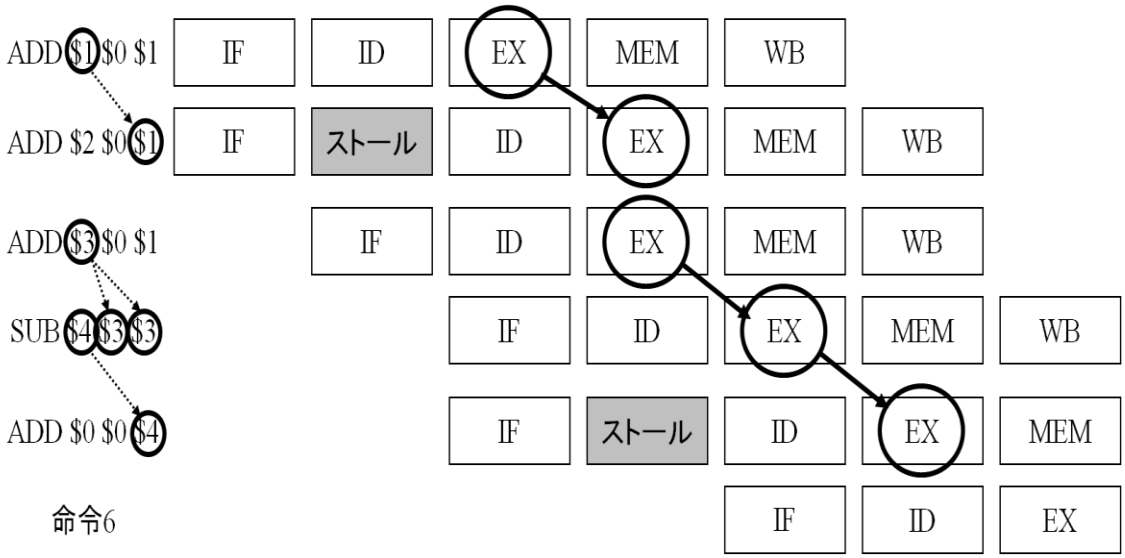

図 17：1 命令目の結果を 2 命令目が参照するストール

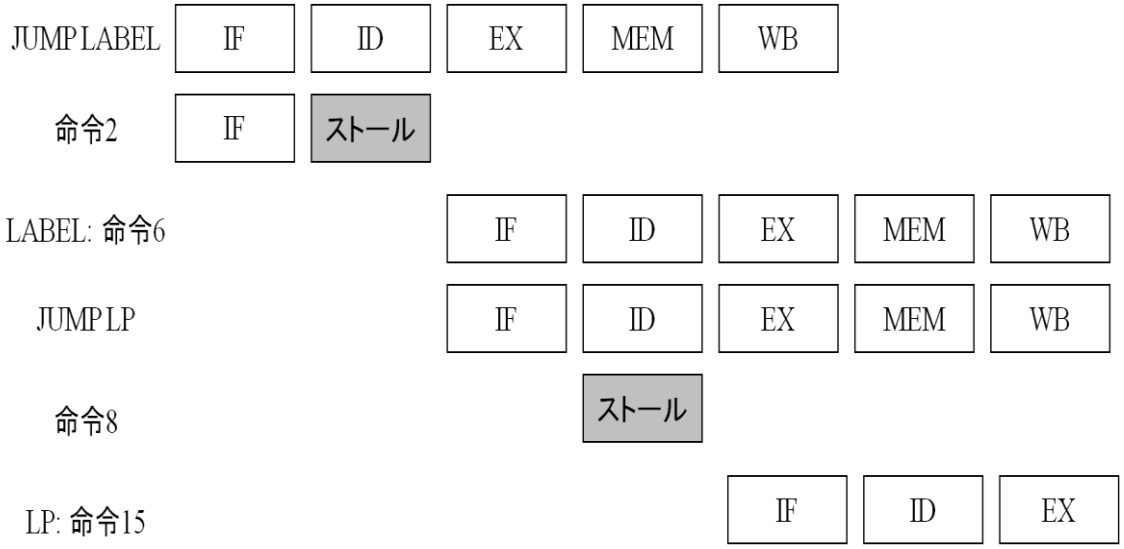

図 18：無条件分岐命令のストール

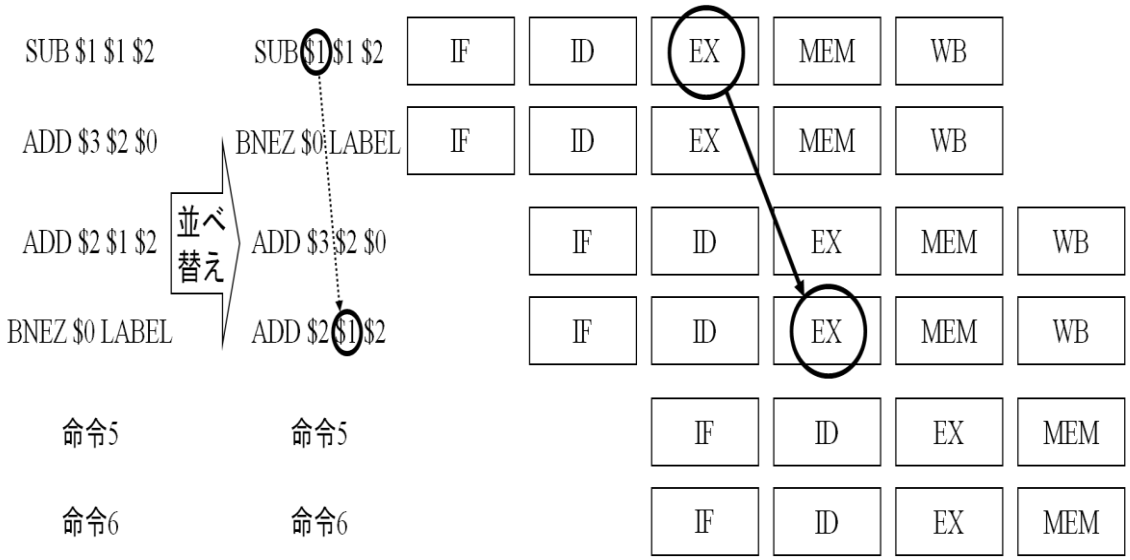

図 19：条件分岐命令の遅延分岐

4.4. シングルサイクルプロセッサのテスト

まず，メニューバー，ツールバーやデータパスなどの表示やエラー出力が正しく動作することを確認した．次に，各命令の処理，数値の最大値や最小値の境界値の変化も正しく動作することを確認した．最後に，テストプログラムを用いたテストを行った．テストプログラムは，9 個のプログラム（N までの和，最大値検出，乗算，除算，素数判定，根の判別，三角形判定，一次方程式，最大公約数）を用いた．これらのテストプログラムで，複雑なプログラムの動作も正しく動作することを確認できた．

5. プロセッサ設計支援ツールの評価

5.1. プロセッサデバッガ・モニタ

プロセッサデバッガ・モニタを用いた実機上での検証では，デバッグコマンドが豊富にあり，効率よくバグの発見ができるようになったため，デバッグ時間の短縮ができた．また，表 2 に示す 3 種類の run コマンドを用いることで，HDL シミュレータ上と実機上での動作の違いの箇所を的確に発見することができるようになった．set pc や set dm コマンドで容易に繰り返し様々なデータのテストができる．以前は，プロセッサデバッガと接続するプロセッサのトップモジュールは全て学習者が記述しなければならなかったが，現在は配線作業だけとしているので，すぐに検証に入ることができる．

5.2. 命令セット定義ツール

命令セット定義ツールによって，様々な命令セットを設計することが可能になった．命令セット情報の修正が簡単に行えるため，アセンブリプログラムの作成や命令セットの評価を繰り返し行える環境ができた．命令セット情報のファイルが，命令セットシミュレータや命令セットアセンブラの入力となっているため，重要なツールである．

5.3. 命令セットアセンブラ

検証プログラムを手書きで機械語に変換し，HDL シミュレータや実機上の検証でバグが発生すると，変換した際の間違いかプロセッサ設計の間違いによるエラーか分からなかった．命令セットアセンブラを用いることで，検証プログラムを手書きで機械語に変換する時間がなくなり，プロセッサ設計のみに集中できるようになった．また，命令セット情報ファイルを変更することで，様々な命令セットのプログラムをアセンブルすることが可能である．HDL シミュレータ上と実機上での検証用の機械語を出力するので，両者の検証時間やデバッグ時間を短縮することができた．

6. ハード/ソフト協調学習システムの評価

6.1. プロセッサ学習システム

ソフトウェア学習では，簡単な課題からアセンブリプログラミングを始め，より多くの課題を作成していくことにより, MONI シミュレータ上に表示されるデータパスのイメージを持つことができる. イメージを持つことで, MONI プロセッサ設計時間の短縮に繋がった. また，実機上での検証もプロセッサデバッガとプロセッサモニタを使用するデバッグコマンドが豊富にあるため，MONI プロセッサ設計時間を短縮することができる．一度，MONI プロセッサの設計を行うため，独自のプロセッサ設計もスムーズに入ることができる.

6.2. 命令セットシミュレータ

シミュレーションできる命令を 70 命令用意し，205 パターンの記述ができるため，学習者が使用したい命令の命令セットの設計ができる．表示している項目が多いため，デバッグがスムーズに行うことができる．以前は，HDL シミュレータ上でアセンブリプログラミングとプロセッサ設計を同時に行っていたため，バグが発生するとアセンブリプログラムかプロセッサかどちらのバグなのか見極めが難しく，プロセッサ設計時間が長くなる原因であった．しかし、本シミュレータを使用することで，プロセッサ設計のための検証が先にできるため，プロセッサ設計を行うだけになり，デバッグもしやすく，プロセッサ設計時間の短縮にもなる．シミュレータのテストで，MONI，SOAR，SARIS 命令セットは，あらかじめ設計していた命令セットを用いたが，SCSFP 命令セットは一から命令セットの設計やアセンブリプログラムの作成をすることができた.

6.3. スーパースカラシミュレータ

シングルサイクルプロセッサを組み込んだことで, 2 命令同時実行のアーキテクチャに関してや, 2 命令間に発生するデータハザードや制御ハザードがいつ起こるのか理解できるようになった．ハザードを発生させないようなアセンブリプログラミング能力も身につけることが可能になった．パイプラインプロセッサも組み込んでいくことで，様々なハザードの発生とその対策や，よりプロセッサアーキテクチャについて知識を身につけることができる．独自プロセッサ設計においても命令セットに最適なプロセッサの設計が可能になり，また，より高速なプロセッサ設計ができると考えられる.

6.4. システム全体

表 9 に学習者の本システムを利用した結果を示す．斜線は，まだその環境が整っていないことを示す．2006 年度に MONI プロセッサを設計した学生は，複数のアーキテクチャで設計することができた．2007 年度に SARIS プロセッサを設計した学生は，MONI のシングルサイクルを設計し，独自のプロセッサで複数のアーキテクチャの設計ができた．2008 年度に PSCSF プロセッサを設計している学生は，MONI シングルサイクルを設計し，命令セット定義ツールや命令セットシミュレータを用いてアセンブリプログラミングを行い，プロセッサ設計まで行うことができた．

ソフトウェア学習とハードウェア学習に着目する．2006 年度から 2007 年度には，プロセッサデバッガと接続するプロセッサのトップモジュールであるインタフェースの部分を用意することによって，HDL を用いた MONI プロセッサ設計や HDL シミュレータ上のデバッグ時間の短縮に繋がった．また，プロセッサデバッガ・モニタにブレイク実行や N クロック実行などのデバッグコマンドを追加したため，実機上でのデバッグ時間が減少した．2008 年度の学生は MONI プロセッサ設計に時間がかかっている．他の二人の学生と比較すると，MONI シミュレータ上でアセンブリプログラムを多く作成したほうが，データパスのイメージが持ちやすく，MONI プロセッサ設計時間は短縮に繋がることがわかる．アセンブリプログラミングも大事だが，プロセッサアーキテクチャの学習を目的としていることを意識させることが重要だと考えられる．

独自プロセッサ設計に着目する．本研究で試作した命令セットシミュレータを用いることで, プロセッサ設計に入る前に検証プログラムが作成でき, PSCSF プロセッサ設計では, HDL を用いたプロセッサ設計, HDL シミュレータ上のデバッグ, 及びプロセッサデバッガ・モニタを用いた実機上のデバッグ時間が短縮できた．これは，命令セットシミュレータによって先に検証プログラムを作成でき，プロセッサ設計ではプロセッサのデバッグだけ行うことが大きいと言える．2008 年度から命令セット定義ツールを使用しているが，使用時間が長い結果になった．命令セット定義ツールに関して，いくつか使用に関してやバグの報告を受けた．まず，仕様書がないため，先輩から使用方法を聞かなければいけないこと，次に，正しい操作をしないと正しく動作しないようになっており，エラー処理などによる対策が不十分であることが問題である．仕様書を作成し，エラー処理も行うことによって，本ツールの使用やプロセッサ設計時間の短縮ができると考えられる．

表 9：ハード/ソフト協調学習システムを利用した学習時間

設計年度	2006		2007		2008
プロセッサ	MONI		SARIS		PSCSF
サイクル	シングル	4 段マルチ	シングル	4 段マルチ	シングル
MONI アセンブリプログラミング	13		12		17
MONI シミュレータ上のデバッグ	12		6		11
作成したプログラム	階乗 三角形判別 根の判別 素数判定 剰余算 一次方程式 8×8 行列加算/減算/乗算 BCD 加算/減算		N までの和 乗算 除算 階乗 素数判定 根の判別 最大公約数 バブルソート		N までの和 4bitBooth BCD 加算
ソフトウェア学習合計	25		18		28
HDL を用いた MONI プロセッサ設計	24	41	13		27
HDL シミュレータ上のデバッグ	14	30	7		21
プロセッサデバッガ・モニタを用いた実機上のデバッグ	25	48	11		2
ハードウェア学習合計	183		31		50
命令セット定義			10		10
命令セット定義ツール使用					14
命令セットシミュレータ使用					29
作成したプログラム			N までの和 階乗 除算 根の判別		N までの和 最大値 素数判定 BCD 加算 一次方程式
アーキテクチャ設計			5	20	11
HDL を用いたプロセッサ設計			35	60	17
HDL シミュレータ上のデバッグ			40	35	34
プロセッサデバッガ・モニタを用いた実機上のデバッグ			15	10	7
独自プロセッサ設計合計			230		122

7. おわりに

本論文では，独自の命令セット設計とそのプロセッサ設計時間を短縮する命令セットシミュレータの設計と試作を行い，ハード/ソフト協調学習システムに実装した．さらに，学習者が並列実行するアーキテクチャを学習するスーパースカラシミュレータの設計と試作を行った．

命令セットシミュレータを利用することで，命令セットの評価を繰り返し行える環境が整った．また，検証プログラムを作成できるため，プロセッサ設計時間の短縮が可能になった．本シミュレータを用意することで，よりソフトウェアとハードウェアのトレードオフを考察することができる．シミュレーションコマンドも，データのセットや複数の実行モードを用意することで，容易にプログラムのデバッグが可能である．スーパースカラシミュレータでは，2 命令同時実行のシングルサイクルアーキテクチャの学習が可能になった．レジスタやデータパスの強調表示によりデータの流れが学習でき，処理のイメージを持つことができる．

学生によるプロセッサ設計では，複数のプロセッサの設計や独自のプロセッサの設計ができるようになった．MONI シミュレータで多くのアセンブリプログラムを作成し，プロセッサアーキテクチャのイメージを持つことで，MONI プロセッサ設計時間を短縮でき，プロセッサデバッガ・モニタを用いた実機検証でもコマンドが豊富になったため，デバッグ時間も短縮することができた．SCSFP プロセッサ設計では，命令セットシミュレータを用いて検証プログラムを作成することで，プロセッサ設計に集中することが可能になり，プロセッサ設計時間が短縮できた．命令セット定義ツールの使用時間が長い結果となり，改善していく必要がある．

今後の課題としては，スーパースカラシミュレータにパイプラインプロセッサを追加すること，命令セット定義ツールと命令セットシミュレータを用いた命令セット設計すること，及び各ツールやシステムの評価を多くの学生に行ってもらうことが挙げられる．システムの評価から改善を行い，本システムの特徴である，学習者がプロセッサにおけるハードウェアとソフトウェアの両面の境界をより学習できるシステムを目指す．

謝辞

本研究の機会を与えてくださり，ご指導いただきました山崎勝弘教授に深く感謝します．また，本研究に関して様々な相談に乗っていただき，貴重な助言，ご意見をいただきました高性能計算研究室の皆様に深く感謝いたします．

参考文献

[1] 片山博誠，石川知雄：COMET 互換プロセッサによる CPU 設計演習環境の開発，情報処理学会，第 58 回全国大会講演論文集，No.1，pp.1-147-1-148，1999.

[2] 西村克信, 額田多政, 天野英晴:教育用パイプライン処理マイクロプロセッサ PICO^2 の開発，情報処理学会研究報告，Vol.2000，No.2，pp.141-148，2000.

[3] 桜井祐一，長澤龍，宮内新，石川知雄 : 教育用 RISC 型マイクロプロセッサ MITEC-II を用いた演習環境の開発及び MITEC-II を用いた演習の実施，情報処理学会研究報告，Vol.2001，No.101，pp.47-54，2001.

[4] 末吉敏則，久我守弘，紫村英智：KITE マイクロプロセッサによる計算機工学教育支援システム，電子情報通信学会論文誌，Vol.J84-D-1，No.6，pp.917-926，2001.

[5] 高橋隆一，大岩元：FPGA を用いたスーパースカラ設計教育に関する一考察，情報処理学会研究報告，Vol.2003，No.49，pp.43-50，2003.

[6] 山口直人，塩見彰睦 : Handel-C による教育用マイクロプロセッサ SEP-3 の設計と工数評価，情報処理学会研究報告，Vol.2003，No.103，pp.7-12，2003.

[7] 重村哲至，守川和夫，力規晃，新田貴之，原田耕治，山田健二：教育用マイコンボードを用いた HDL 演習環境の実現，情報処理学会研究報告，Vol.2005，No15，pp.43-48，2005.

[8] 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 回全国大会講演論文集，5T-5，2004.

[9] 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 回全国大会講演論文集，5T-6，2004.

[10] 中村浩一郎，池田修久，山崎勝弘，小柳滋：プロセッサアーキテクチャ教育用 FPGA ボードコンピュータシステムの開発，情報科学技術レターズ，FIT2004，LC-008，2004.

[11] 難波翔一朗，中村浩一郎，山崎勝弘，小柳滋：マイクロプロセッサの設計と検証に基づいたハード/ソフト・コラーニングシステムの拡張，情報科学技術レターズ，FIT2005，LC-001，2005.

[12] Hoang Anh Tuan，Nakamura Koichiro，Namba Shoichiro，Yamazaki Katsuhiro，Oyanagi shigeru : A FPGA Based Hardware / Software Co-learning System，IEICE，Technical Report，RECONF 2005-48，pp.43-48，2005.

[13] 難波翔一朗，中村浩一郎，Hoang Anh Tuan，山崎勝弘，小柳滋：ハード/ソフト協調学習システムのための命令セット定義ツールとプロセッサデバッガの開発，情報科学技術レターズ，FIT2006，N-009，2006.

[14] 難波翔一朗，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，情報科学技術レターズ，FIT2007，LC-002，

2007.

[15] 井手純一，志水建太，山崎勝弘，小柳滋：学生によるプロセッサ設計実験に基づいたハード/ソフト協調学習システムの評価，FIT2008，C-009，2008.

[16] 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, A-05, 2008.

[17] 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卒業論文，2002.

[18] 大八木睦：ハードウェア記述言語によるマルチサイクル/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卒業論文，2002.

[19] 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.

[20] 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.

[21] 藤原淳平：ハード/ソフト協調学習でのプロセッサアーキテクチャ可変な最適化コンパイラの設計，立命館大学理工学研究科修士論文，2005.

[22] 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卒業論文，2005.

[23] 中村浩一郎：命令定義可能なハード/ソフト・コラーニングシステム上でのプロセッサデバッガの設計と実装，立命館大学理工学研究科修士論文，2006.

[24] 富樫望：命令セット定義可能な汎用アセンブラの設計と実装，立命館大学理工学部卒業論文，2006.

[25] 中川裕樹：ハード/ソフト協調学習のための汎用アセンブラのユーザインタフェースの設計と実装，立命館大学理工学部卒業論文，2006.

[26] 西田範行：ハード/ソフト協調学習のための汎用シミュレータの設計，立命館大学理工学部卒業論文，2006.

[27] 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.

[28] 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卒業論文，2007.

[29] 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卒業論文，2008.

[30] 宮崎匡史：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，立命館大学理工学部卒業論文，2009.

[31] 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) 第 2 版，日経ＢＰ社，2003.

[32] 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，2006.
[33] 高橋麻奈 著 : やさしい C++ 第 3 版，ソフトバンク クリエイティブ，2007.
[34] 青木淳夫 著，山田祥寛 監修 : プログラムを作ろう！パソコン教科書 Visual C++ 2005 Express Edition 入門，日経 BP ソフトプレス，2007.
[35] 林晴比古 著 : 明快入門 Visual C++ 2005 ビギナー編，ソフトバンク クリエイティブ，2007.
[36] 林晴比古 著:明快入門 Visual C++ 2005 シニア編, ソフトバンク クリエイティブ, 2007.
[37] プログラミングを書こう！のページ：http://www.asahi-net.or.jp/~yf8k-kbys/

付録

命令セットシミュレータで使用できる命令の種類と，その記述例を表 10 に示す．動作内容が命令セット定義ツールで定義する項目で，実際にアセンブリプログラミングを行う際は，記述例のように記述する．

表 10：命令の種類と記述例

種類	動作内容	動作説明	記述例	意味
算術 演算子	ADD	加算	ADD　$0　$1　$2	レジスタ 1 番地と 2 番地の加算結果を 0 番地に格納
	SUB	減算	SUB　$3　$4　$5	レジスタ 4 番地と 5 番地の減算結果を 3 番地に格納
	MUL	乗算	MUL　$0　$1　[0]	レジスタ 1 番地とデータメモリ 0 番地の 乗算結果を 0 番地に格納
	DIV	除算	DIV　$3　$2　[4]	レジスタ 2 番地とデータメモリ 4 番地の 除算結果を 0 番地に格納
	REM	剰余算	REM　$1　$2　5	レジスタ 2 番地と即値 5 の剰余算結果を 1 番地に格納
ビット 演算子	NOT	ビット反転	NOT　$3　[$1]	レジスタ 1 番地が指すデータメモリの データのビット反転を 3 番地に格納
	AND	論理積	AND　$0　$0　[$0]	レジスタ 0 番地と 0 番地が指すデータメモリの データの論理積結果を 0 番地に格納
	OR	論理和	OR　$1　$2　$1	レジスタ 2 番地と 1 番地の論理和結果を 1 番地に格納
	XOR	排他的論理和	XOR　$2　$0　2	レジスタ 0 番地と即値 2 の排他的論理和結果を 2 番地に格納
	XNOR	排他的論理和の否定	XNOR　$1　$4　1	レジスタ 4 番地と即値 1 の排他的論理和の 否定剰結果を 1 番地に格納
単項 演算子	RAND	各桁ビットの論理積	RAND　$1　[1]	データメモリ 1 番地の各桁ビットの 論理積結果をレジスタ 1 番地に格納
	RNAND	各桁ビットの否定論理積	RNAND　$2　$2	レジスタ 2 番地の各桁ビットの論理積の否定結果 否定論理積結果を 2 番地に格納
	ROR	各桁ビットの論理和	ROR　$3　3	即値 3 の各桁ビットの論理和結果を レジスタ 3 番地に格納
	RNOR	各桁ビットの否定論理和	RNOR　$4　[$4]	レジスタ 4 番地が指すデータメモリのデータの 各桁ビットの否定論理和結果を 4 番地に格納

	RXOR	各桁ビットの排他的論理和	RXOR $5 [$5+5]	レジスタ 5 番地と即値 5 の加算が指す データメモリのデータの各桁ビットの排他的 論理和結果を 5 番地に格納
	RXNOR	各桁ビットの排他的論理和の否定	RXNOR $6 $6	レジスタ 6 番地の各桁ビットの排他的論理和の 否定結果を 6 番地に格納
等号 演算子	SEQ	等しいと 1 等しくないと 0	SEQ $0 $1 [10]	レジスタ 1 番地とデータメモリ 10 番地が 等しいと 0 番地に 1、等しくないと 0 を格納
	SNE	等しくないと 1, 等しいと 0	SNE $1 $2 $0	レジスタ 2 番地と 0 番地が等しくないと 1 番地に 1、等しいと 0 を格納
関係 演算子	SLT	成り立つと 1,成り立たないなら 0	SLT $2 $3 5	レジスタ 3 番地が即値 5 より小さいと 2 番地に 1、それ以外は 0 を格納
	SGT	成り立つと 1,成り立たないなら 0	SGT $3 $4 [$7]	レジスタ 3 番地が 7 番地が指すデータメモリの データより大きいと 3 番地に 1、それ以外は 0 を格納
	SLE	成り立つと 1,成り立たないなら 0	SLE $4 $3 [$1+2]	レジスタ 3 番地が 1 番地と即値 2 の加算が指す データメモリのデータと同じか小さいと 4 番地に 1、それ以外は 0 を格納
	SGE	成り立つと 1,成り立たないなら 0	SGE $3 $2 [0]	レジスタ 2 番地がデータメモリ 0 番地と同じか 大きいと 3 番地に 1、それ以外は 0 を格納
シフト 演算子	SLL	左論理シフト	SLL $2 $1 $1	レジスタ 1 番地が 1 番地分左論理シフト結果を 2 番地に格納
	SRL	右論理シフト	SRL $1 $0 2	レジスタ 0 番地が即値 2 分右論理シフト結果を 1 番地に格納
	SRA	右算術シフト	SRA $0 $0 [$3+4]	レジスタ 0 番地が 3 番地と即値 4 の加算が指す データメモリのデータ分右算術シフト結果を 0 番地に格納
データ 代入	LDLI	下位半分に即値を設定	LDLI $0 $1 1	レジスタ 1 番地の下位半分に即値 1 を代入し 0 番地に格納
	LDHI	上位半分に即値を設定	LDHI $2 $2 3	レジスタ 2 番地の上位半分に即値 3 を代入し 2 番地に格納
ロード, ストア	LD	データメモリからレジスタへロード	LD $1 1	即値 1 をレジスタ 1 番地に格納
	ST	レジスタからデータメモリへストア	ST [$3] $1	レジスタ 1 番地を 3 番地が指す データメモリに格納
ポップ	POP	スタックからレジスタへポップ	POP $0	スタックのデータをレジスタ 0 番地にポップ
プッシュ	PUSH	レジスタデータをスタックへプッシュ	PUSH $1	レジスタ 1 番地をスタックへプッシュ
コール,	CALL	サブルーチンへ分岐	CALL LABEL	LABEL へジャンプ

ジャンプ	JUMP	ラベルへ分岐	RETURN	CALL 命令の次の命令へジャンプ
リターン	RETURN	サブルーチンから復帰	JUMP　LABEL	LABEL へジャンプ
条件分岐	BEQZ	レジスタが 0 なら絶対分岐	BEQZ　$0　LABEL	レジスタ 0 番地が 0 なら LABEL へ分岐
	BEQZP	レジスタが 0 なら PC 相対分岐	BEQZP　$1　LABEL	レジスタ 1 番地が 0 なら LABEL へ分岐
	BNEZ	レジスタが 0 でないなら絶対分岐	BNEZ　$2　LABEL	レジスタ 2 番地が 0 なら LABEL へ分岐
	BNEZP	レジスタが 0 でないなら PC 相対分岐	BNEZP　$3　LABEL	レジスタ 3 番地が 0 なら LABEL へ分岐
	BZF	ゼロフラグが 1 なら絶対分岐	BZ　LABEL	ZF が 1 なら LABEL へ分岐
	BZFP	ゼロフラグが 1 なら PC 相対分岐	BZP　LABEL	ZF が 1 なら LABEL へ分岐
	BNZ	ゼロフラグが 0 なら絶対分岐	BNZ　LABEL	ZF が 0 なら LABEL へ分岐
	BNZP	ゼロフラグが 0 なら PC 相対分岐	BNZP　LABEL	ZF が 0 なら LABEL へ分岐
	BP	ゼロフラグが 0 かつ ネガティブフラグが 0 なら絶対分岐	BP　LABEL	ZF が 0 かつ NF が 0 なら LABEL へ分岐
	BPP	ゼロフラグが 0 かつ ネガティブフラグが 0 なら絶対分岐	BPP　LABEL	ZF が 0 かつ NF が 0 なら LABEL へ分岐
	BN	ネガティブフラグが 1 なら絶対分岐	BN　LABEL	NF が 1 なら LABEL へ分岐
	BNP	ネガティブフラグが 1 なら PC 相対分岐	BNP　LABEL	NF が 1 なら LABEL へ分岐
	BZP	ネガティブフラグが 0 なら絶対分岐	BZP　LABEL	NF が 0 なら LABEL へ分岐
	BZPP	ネガティブフラグが 0 なら PC 相対分岐	BZPP　LABEL	NF が 0 なら LABEL へ分岐
	BZN	ゼロフラグが 1 か ネガティブフラグが 1 なら絶対分岐	BZN　LABEL	ZF が 1 か NF が 1 なら LABEL へ分岐
	BZNP	ゼロフラグが 1 か ネガティブフラグが 1 なら PC 相対分岐	BZNP　LABEL	ZF が 1 か NF が 1 なら LABEL へ分岐
	BC	キャリーフラグが 1 なら絶対分岐	BC　LABEL	CF が 1 なら LABEL へ分岐
	BCP	キャリーフラグが 1 なら PC 相対分岐	BCP　LABEL	CF が 1 なら LABEL へ分岐
	BNC	キャリーフラグが 0 なら絶対分岐	BNC　LABEL	CF が 0 なら LABEL へ分岐
	BNCP	キャリーフラグが 0 なら PC 相対分岐	BNCP　LABEL	CF が 0 なら LABEL へ分岐
	BV	オーバーフローフラグが 1 なら絶対分岐	BV　LABEL	VF が 1 なら LABEL へ分岐
	BVP	オーバーフローフラグが 1 なら PC 相対分岐	BVP　LABEL	VF が 1 なら LABEL へ分岐
	BNV	オーバーフローフラグが 0 なら絶対分岐	BNV　LABEL	VF が 0 なら LABEL へ分岐
	BNVP	オーバーフローフラグが 0 なら PC 相対分岐	BNVP　LABEL	VF が 0 なら LABEL へ分岐
	BGE	オーバーフローフラグとネガティブフ	BGE　LABEL	VF と NF がともに 0 か 1 なら LABEL へ分岐

		ラグの排他的論理和が 0 なら絶対分岐		
	BGEP	オーバーフローフラグとネガティブフラグの排他的論理和が 0 なら PC 相対分岐	BGEP　LABEL	VF と NF がともに 0 か 1 なら LABEL へ分岐
	BLT	オーバーフローフラグとネガティブフラグの排他的論理和が 1 なら絶対分岐	BLT　LABEL	VF と NF が違うなら LABEL へ分岐
	BLTP	オーバーフローフラグとネガティブフラグの排他的論理和が 1 なら PC 相対分岐	BLTP　LABEL	VF と NF が違うなら LABEL へ分岐
	BGT	オーバーフローフラグとネガティブフラグの排他的論理和が 0 かつゼロフラグが 0 なら絶対分岐	BGT　LABEL	VF と NF がともに 0 か 1 かつ ZF が 0 なら LABEL へ分岐
	BGTP	オーバーフローフラグとネガティブフラグの排他的論理和が 0 かつゼロフラグが 0 なら PC 相対分岐	BGTP　LABEL	VF と NF がともに 0 か 1 かつ ZF が 0 なら LABEL へ分岐
	BLE	オーバーフローフラグとネガティブフラグの排他的論理和が 1 かつゼロフラグが 1 なら絶対分岐	BLE　LABEL	VF と NF が違いかつ ZF が 1 なら LABEL へ分岐
	BLEP	オーバーフローフラグとネガティブフラグの排他的論理和が 1 かつゼロフラグが 1 なら PC 相対分岐	BLEP　LABEL	VF と NF が違いかつ ZF が 1 なら LABEL へ分岐
キャリー	SCF	キャリーフラグに 1 セット	SCF	CF を 1 にセット
	RCF	キャリーフラグのリセット	RCF	CF を 0 にリセット
その他	NOP	何もしない	NOP	1 命令何もしない
	HALT	停止	HALT	プログラムの終了

「