卒業論文

デザインパターンを用いた

教育用プロセッサの設計

氏　　名：寺澤　佳高
学籍番号：2260070062-0
指導教員：山崎 勝弘　教授
提 出 日：2011 年 2 月 18 日

立命館大学理工学部　電子情報デザイン学科

内容梗概

本論文では，ハード/ソフト協調学習システムとデザインパターンを用いて，プロセッサを設計することにより，ハードウェアとソフトウェアの両方の観点から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の複合演算命令搭載プロセッサを設計した．設計したプロセッサは MONIT（MONI includes additional functions by Terazawa）と名付けた．また，設計したプロセッサを HDL シミュレータで検証，およびプロセッサデバッガと接続して 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複合演算命令搭載プロセッサを設計することで，どのような実現方法があるかをハードウェアとソフトウェアの両分野から理解し，今後の学習者が目的・原理を理解しやすくすることを目的とする．

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の進歩により，LSI の小型化，軽量化，高速化，低消費電力化が可能となった．携帯電話，デジタルカメラ，複写機，自動販売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ている．そのため、組み込み機器に要求される仕様は大規模かつ複雑になり，小型化や低消費電力化が進んでいる．さらに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．したがって，開発期間短縮に大きく影響するハード/ソフト協調設計がますます重要となっている．近年の半導体開発技術はハードウェアとソフトウェアに密接な関係があり，ハードとソフト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．こうした LSI 開発技術の発展の時代背景において，システム LSI に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．

以上の背景から，大学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．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[2]．MONI を，このシステムの基本になるプロセッサとして設計し，MONI を学習した本研究員が設計したプロセッサが SOAR[3]や，SARIS[4]にあたる．ハード/ソフト協調学習システムは学習者が評価を行うことにより，システムの改善，または今後のシステム拡張を行う。システムの拡張として，今年からデザインパターンを利用している[12]．デザインパターンとは，基本的なプロセッサの構成をデータ化することにより，学習者の理解を深めることが出来るとともに，時間効率の良い設計を目的としたシステムである．デザインパターンで過去の設計を再利用することにより，過去のハード/ソフト協調学習システムを使用したプロセッサの理解を促進し，より高度なプロセッサの設計・学習に時間を充てることが出来る．

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の設計を行う．また，今回乗算器を搭載することにより，複合演算命令セットに対応させる．実際にプロセッサ設計を行うことによりシングル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．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．

本研究の目的は，ハード/ソフト協調学習システムとデザインパターンを利用し，実際にプロセッサ設計を行うことによって，どの程度この両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．また両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．

本論文では，第2章で今回設計したMONITプロセッサについての概要と構成，命令セッ

トアーキテクチャの詳細を説明する．また，複合演算命令セットについても2章で説明する．第3章ではデザインパターンについての詳細を説明する．章第4章ではMONITの評価，第5章でハード/ソフト協調学習システムとデザインパターンの評価について述べる．

2　MONIT プロセッサ

2.1　ハード/ソフト協調学習システム

2.1.1　システム概要

ハード/ソフト協調学習システム[2]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．

2.1.2　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．

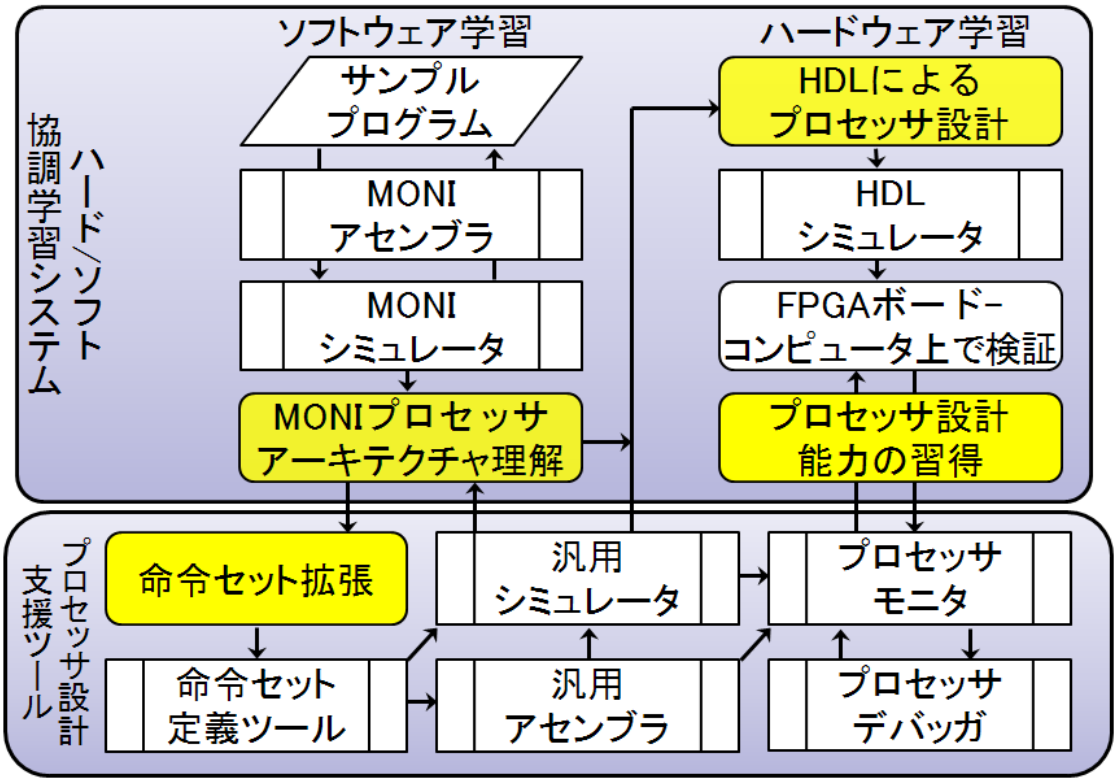

図 1：ハード/ソフト協調学習システムの学習体系

2.2　MONIT の設計思想

本研究では，ハード/ソフト協調学習システムとデザインパターンがプロセッサ設計において有効であるかどうかを評価するために，シングルサイクルマイクロプロセッサを設計した．設計したプロセッサは MONI プロセッサの命令セットを参考にし，新しい機能として乗算器を搭載している，MONIT（MONI includes additional functions by Terazawa）と名付けた．MONI プロセッサとは，本研究室を卒業された池田 修久氏と大八木 睦氏によって設計されたプロセッサである．MONIT プロセッサには以下のような特徴がある．

- 16ビット固定命令長
- 3オペランド命令方式
- 全70命令

4 つの命令形式

2.3 MONIT アーキテクチャ

2.3.1 命令セット

MONIT には，Register 形式（R 形式），Immediate5 形式（I5 形式），Immediate8 形式（I8 形式），Jump 形式（J 形式）の４つの命令形式がある．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．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．表 1 に MONIT の命令形式を示す．

表 1：MONIT の命令形式

format＼bit	5	3	3	3	2
R 形式	op	rs	rd	rt	fn
I5 形式	op	rs	rd	Imm	
I8 形式	op	rs	Imm		
J 形式	op	Imm			

op フィールドを 5bit 使用し，命令を定義している．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，さらに拡張が可能である．表 2 に命令フィールドを示す．また，MONIT の命令セットの詳細は付録に添付する．

表 2：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

2.3.2　　複合演算命令セット

(1)目的

プロセッサの処理と並行して乗算計算を行えるように乗算器を設計した．また行列式の積和演算等を高速に行うことが出来るように，複合演算命令セットを採用した．

(2)実現方法

複合演算命令セットを実現するために乗算器を設計し，複合演算命令セット用レジスタを用意する．複合演算命令セットの動作を下図 2 に示す．動作フローとしては，レジスタから送られた 2 つのデータをブースアルゴリズムにて乗算計算を行う．計算中は NOP 処理を行い，計算終了時に乗算器内のレジスタに解答を保存しておく．終了後 ALU に解答を送り，加算を行う．

表 3：追加した命令セット

命令名	OPECODE	FN	意味
MUL	11101	00	乗算開始
ADDM	11101	01	乗算結果との加算
SUBM	11101	10	乗算結果との減算

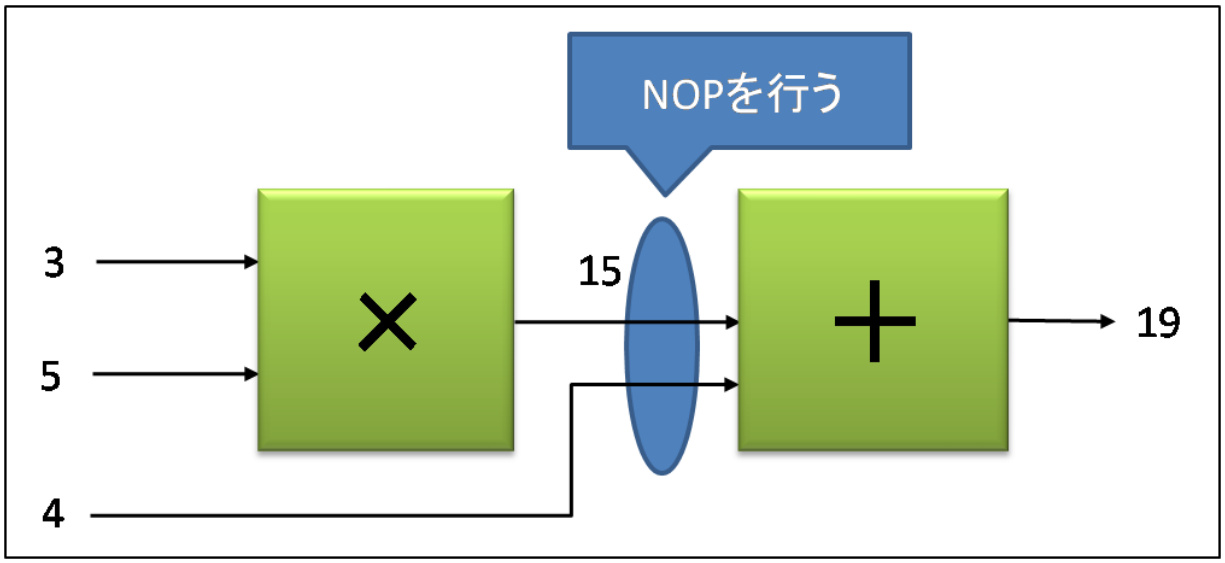

図 2：複合演算命令実現方法

2.3.3　MONIT プロセッサアーキテクチャ

全体のアーキテクチャを図 3 に，各モジュールの動作と意味を表 3 に示す．MONIT プロセッサは赤色で示されたシングルサイクルのプロセッサモジュールとして 13 個，複合演算命令セット実現のために加えた緑色で示された 1 個のモジュール，そして命令メモリ(IM)とデータメモリ(DM)の全部で 16 個からなる．命令メモリとデータメモリはハード/ソフト協調学習システムのプロセッサデバッガから提供される．なお，可読性の低下につながるので細かな制御線を省略している．

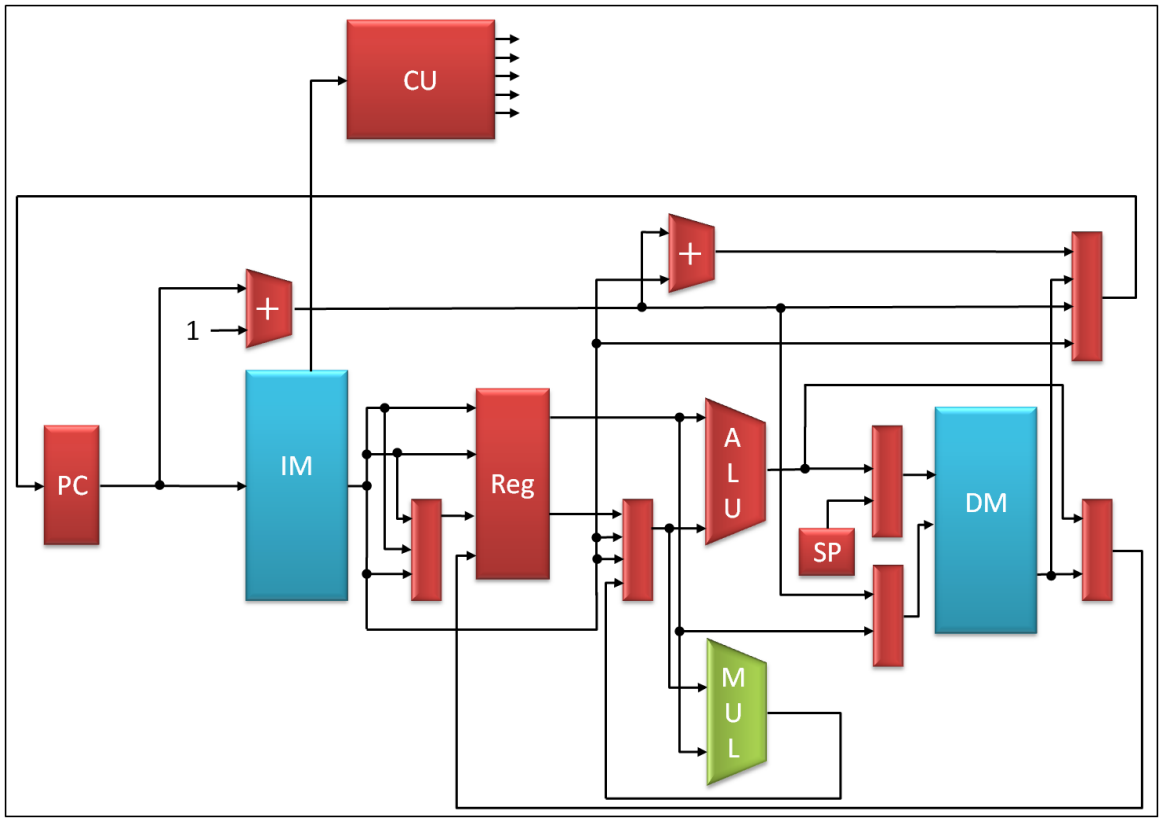

図 3：MONIT プロセッサのアーキテクチャ

表 4：各モジュールの動作と意味

モジュール名	意味	動作
PC	Program Counter	命令メモリへアドレスを渡す.
Reg	Register File	2 つのソースレジスタと，1 つのディスティネーションレジスタを指定出来る.
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.
記名無しモジュール	Multiplexer	複数の入力から, CU からの制御信号により出力を選択する.
CU	Control Unit	IM から Opecode や Fnction を受け取り，ADD 以外の各モジュールへ制御信号を送る.

2.3.4　データパス

実行される命令形式についてのデータパスを述べる．すべての命令は，PC の値の指すアドレスから取り出した命令から，コントロールユニットに OPECODE フィールドと FN フィールドを送る．コントロールユニットはそれぞれのフィールドからデータパスを選択することでデータパスが変化する．

（1）　R 形式

R 形式の命令のデータパスを図 4 に示す．R 形式は，ADD や SUB など，レジスタ値＋レジスタ値の演算を行い，結果をレジスタに代入する命令形式である．まず，命令から rs，rt の 2 つのフィールドのアドレス値からそれぞれの指すレジスタ値を取り出す．次に FN フィールドで ALU の動作を選択し，ALU の演算結果を rd の指すレジスタに代入する．PC はアドレス加算機によって 1 加算され，次のクロックで次の命令が実行される．

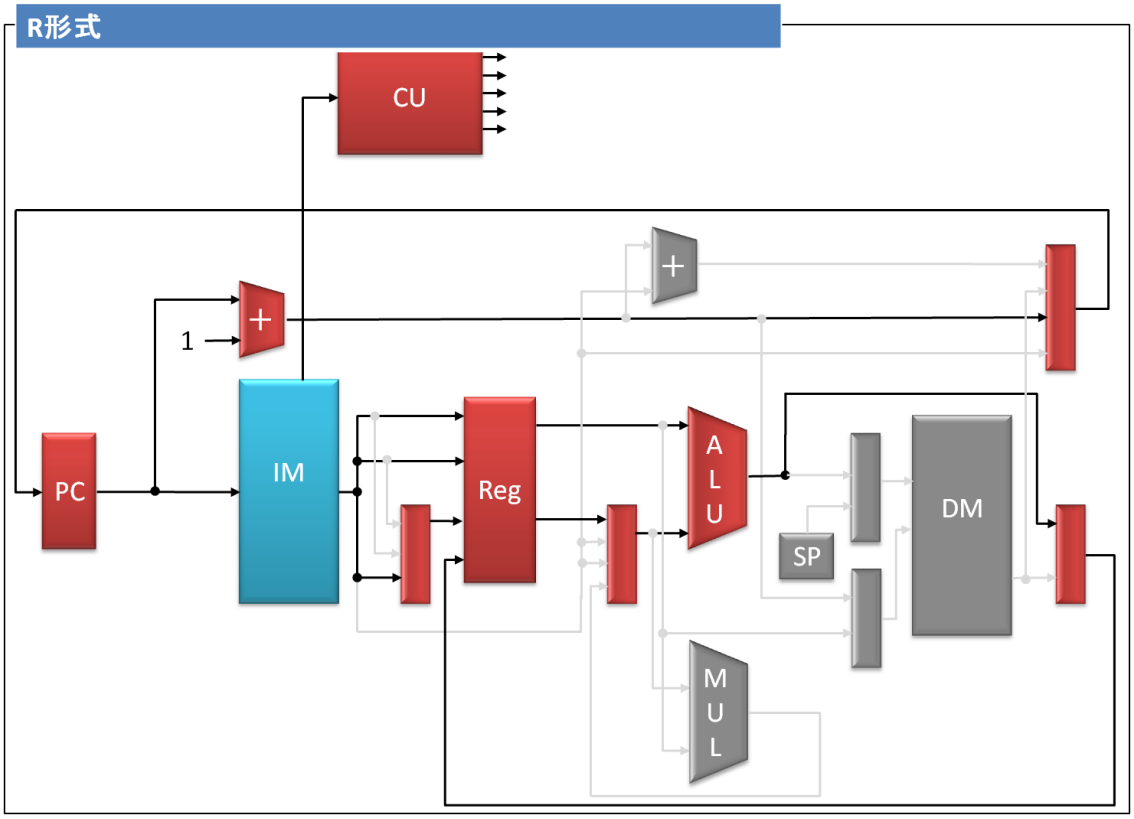

図 4：R 形式のデータパス

(2)　I5 形式

I5 形式の命令が実行される過程を図 4 に示す．I5 形式は，ADDI や SUBI など，レジスタ値と即値 5 ビットの演算を行い，結果をレジスタに代入する命令形式である．まず，rs の指すレジスタ値と Immediate フィールドから即値幅セレクタを使い取り出した下位 5 ビットの値を用いて ALU で演算を行う．ALU は OPECODE フィールドのみで動作を選択する．そして，演算結果を rd の指すレジスタに代入する．PC は R 形式と同様である．

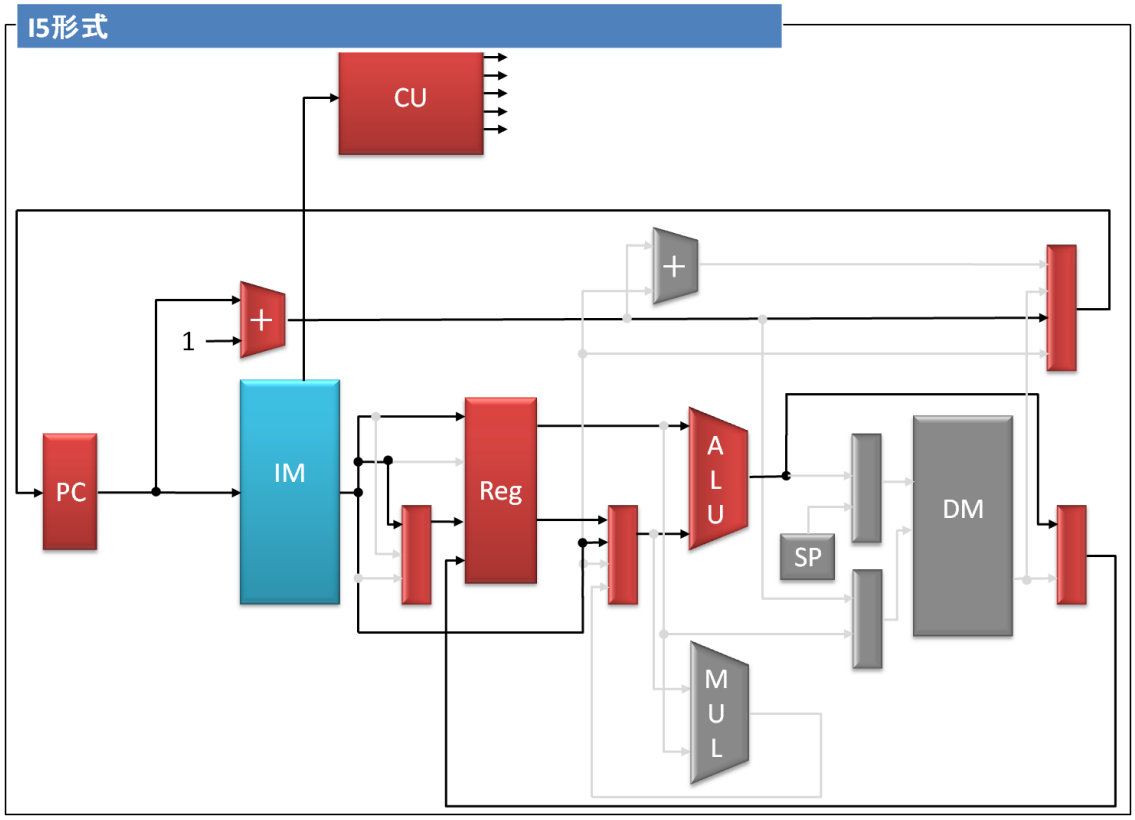

図 5：I5 形式のデータパス

(3)　ロード命令

MONIT の命令セットでは rs がデータの格納先を指し，rt がメモリ番地を指している. データメモリにある rt 番地の値をレジスタファイル rs に格納するために IM からアドレス rt を出力し，ロード命令を行っている．PC の値は R 形式と同様である.

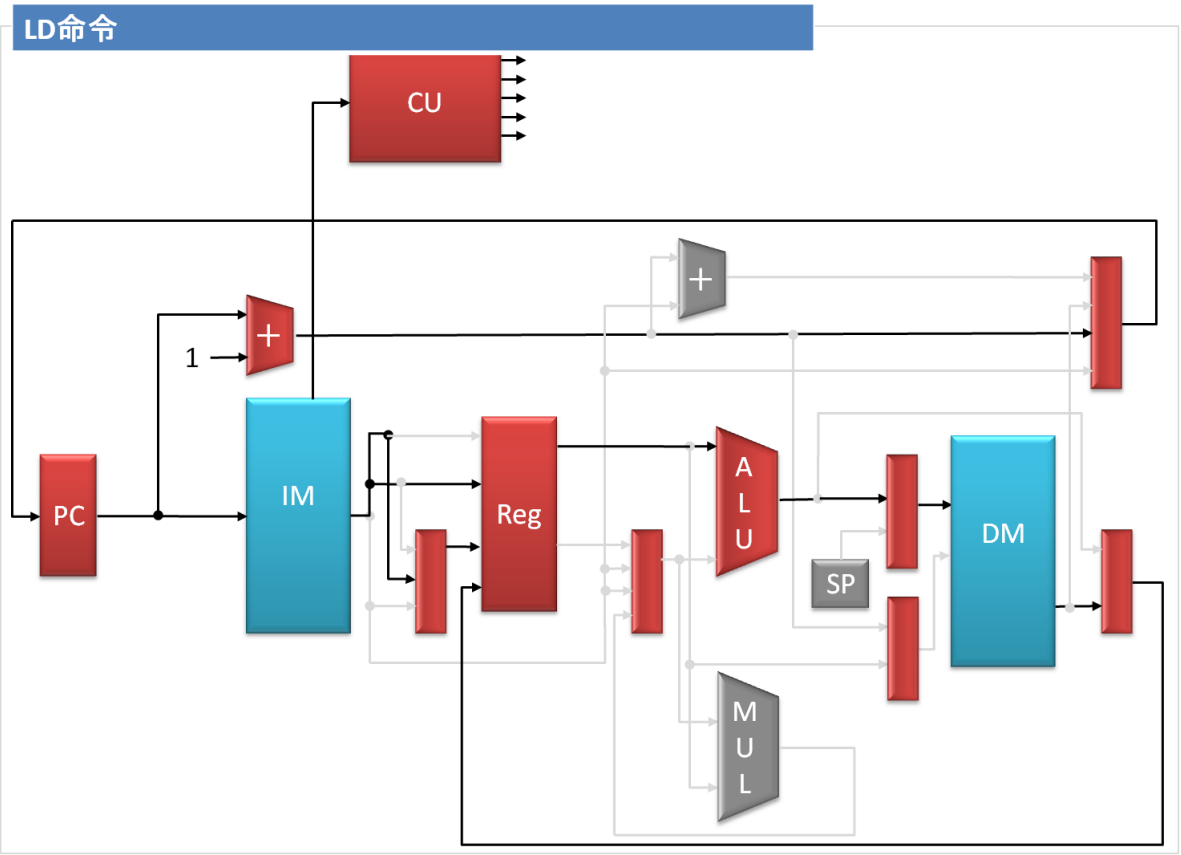

図 6：ロード命令のデータパス

(4)　ストア命令

MONIT の命令セットでは rs がデータのアドレスを指し, rt が値を指している. ロード命令とは反対のレジスタ指定になっている. レジスタを逆に指定する方法としては ALU で ST の場合はレジスタ入力の下側を出力するように指定する方法をとっている. 動作としては, rs のデータをレジスタファイルからデータメモリの rt 番地に格納している. PC の値は R 形式と同様である.

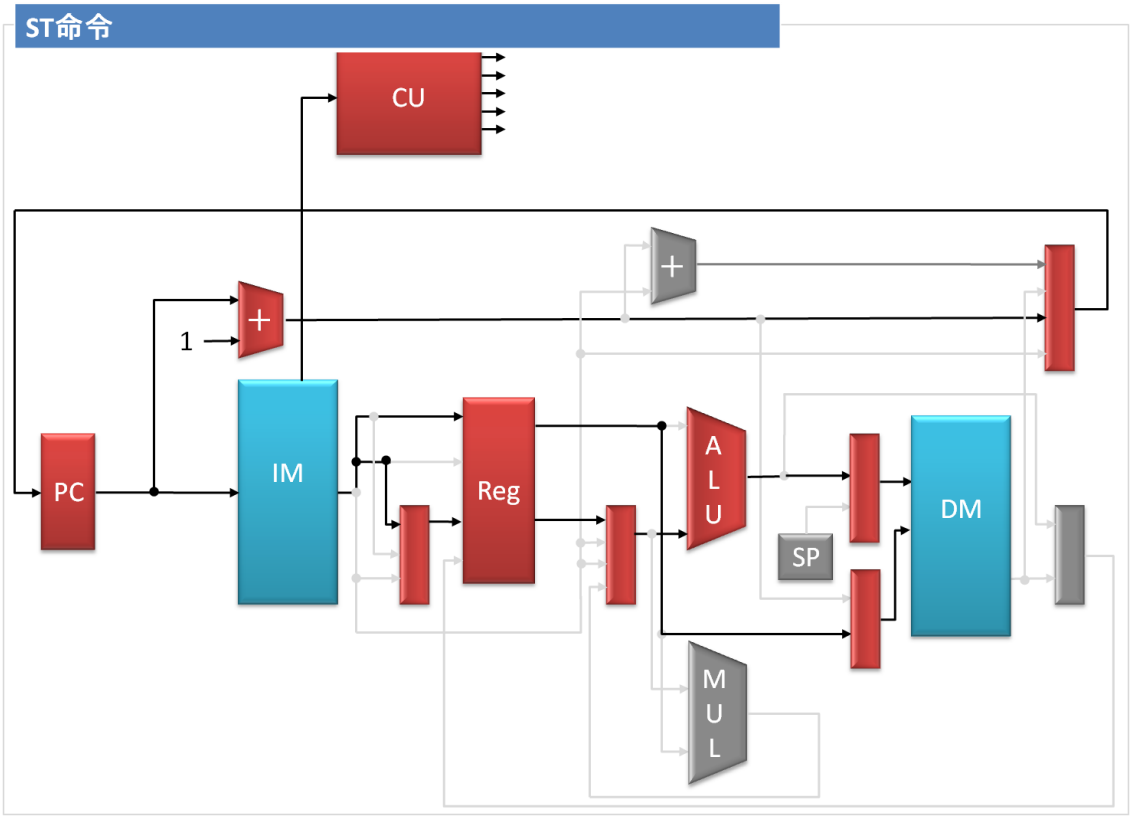

図 7：ストア命令のデータパス

(5)　I8 形式

I8 形式の命令が実行される過程を述べる. I8 形式の命令には BEQZ, BNEZ や, PUSH, POP 等が含まれる.

(i)　分岐命令

レジスタ rs の値により次の PC の値を決定する. 図には配線を載せていないが実際は ALU で判別を行いその結果を PC_NEXT_MUX に送信している.

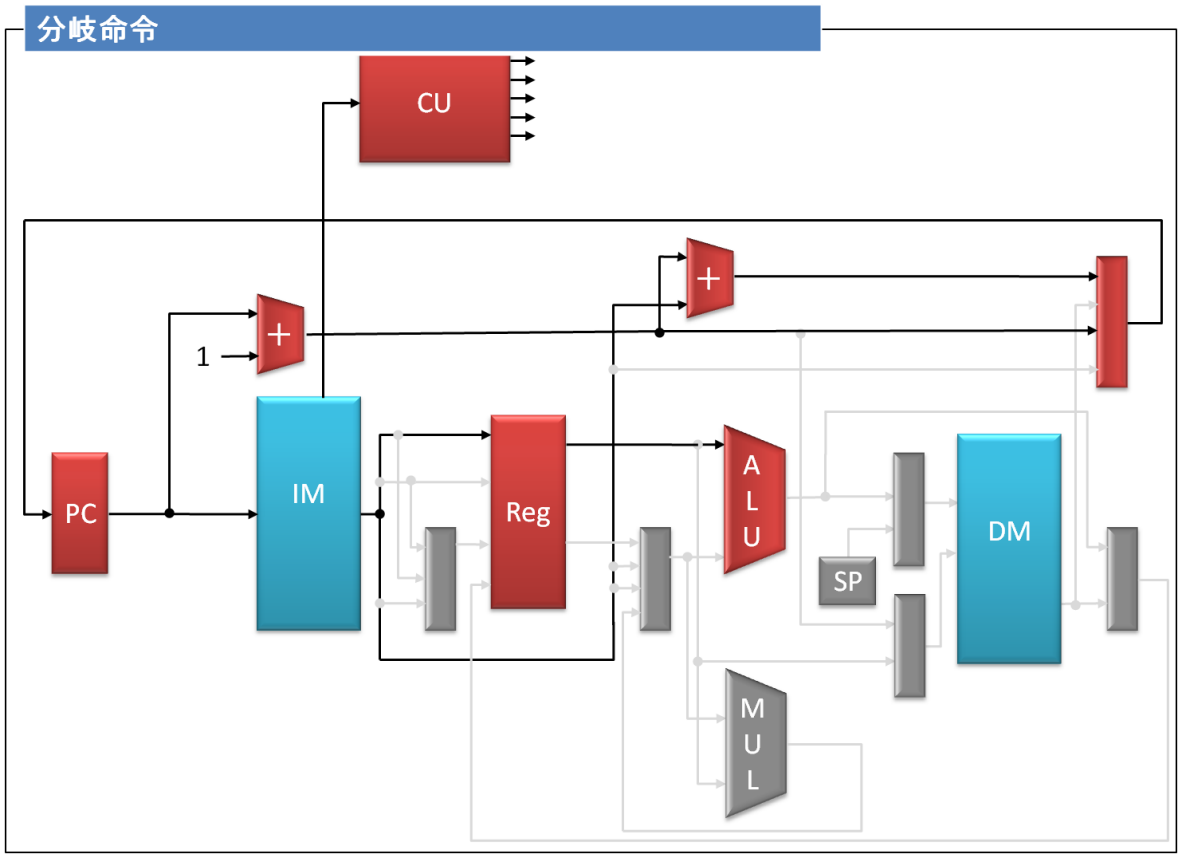

図 8：分岐命令のデータパス

(ii)　スタック命令

スタックポインタを搭載しているため，PUSH・POP 命令を行うことが出来る．PUSH 命令の動作はスタックポインタ自体がデータメモリの一番下から積み上げていく仕組みのため，スタックポインタを 1 上げることにより，PUSH 動作として扱う．下げると同時にそのアドレスをデータメモリに保存しておく．POP 命令の動作は PUSH の逆になり，スタックポインタを 1 下げると同時に，データメモリのアドレスを送信する．

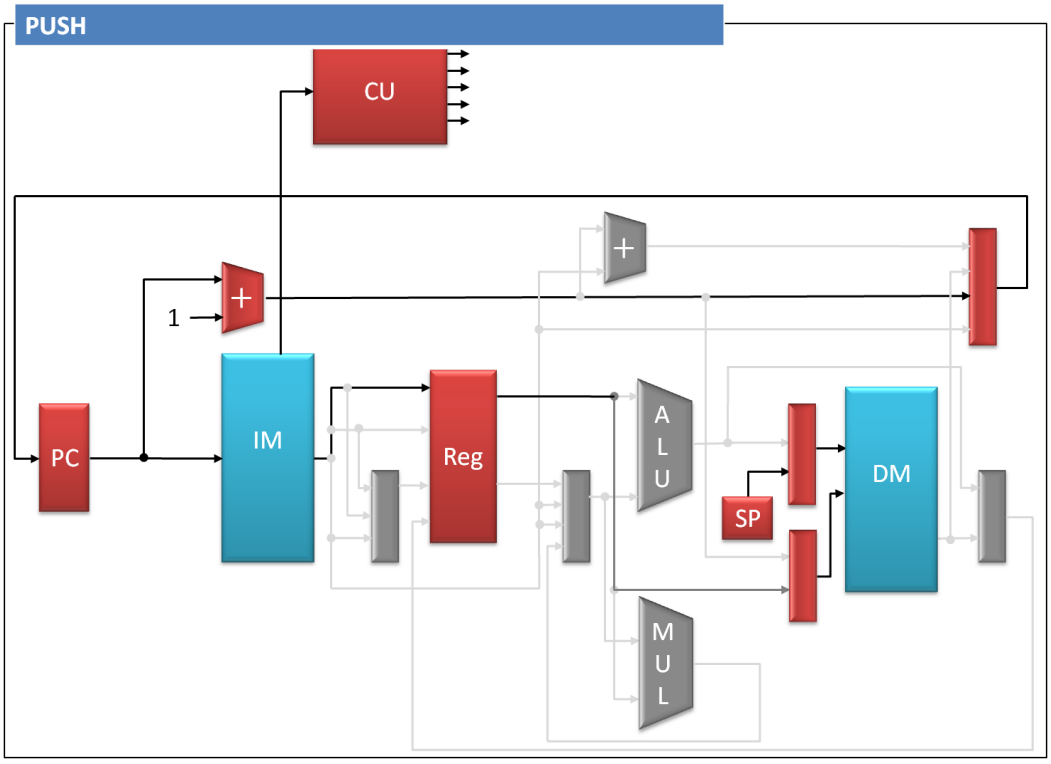

図 9：PUSH 命令のデータパス

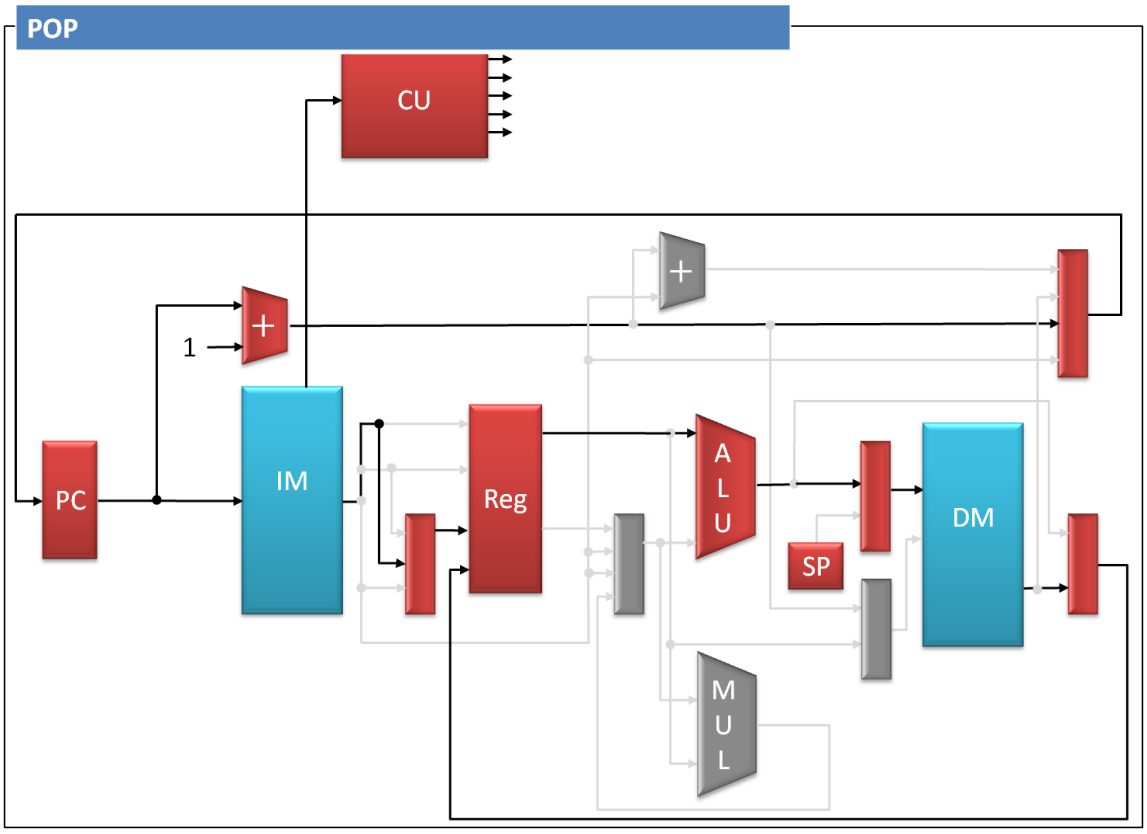

図 10：POP 命令のデータパス

(6)　複合演算命令

複合演算命令を図 11 に示す．複合演算命令は，今回 MONI の命令セットに追加した命令である．動作としては，2.3.2 で述べた通りである．

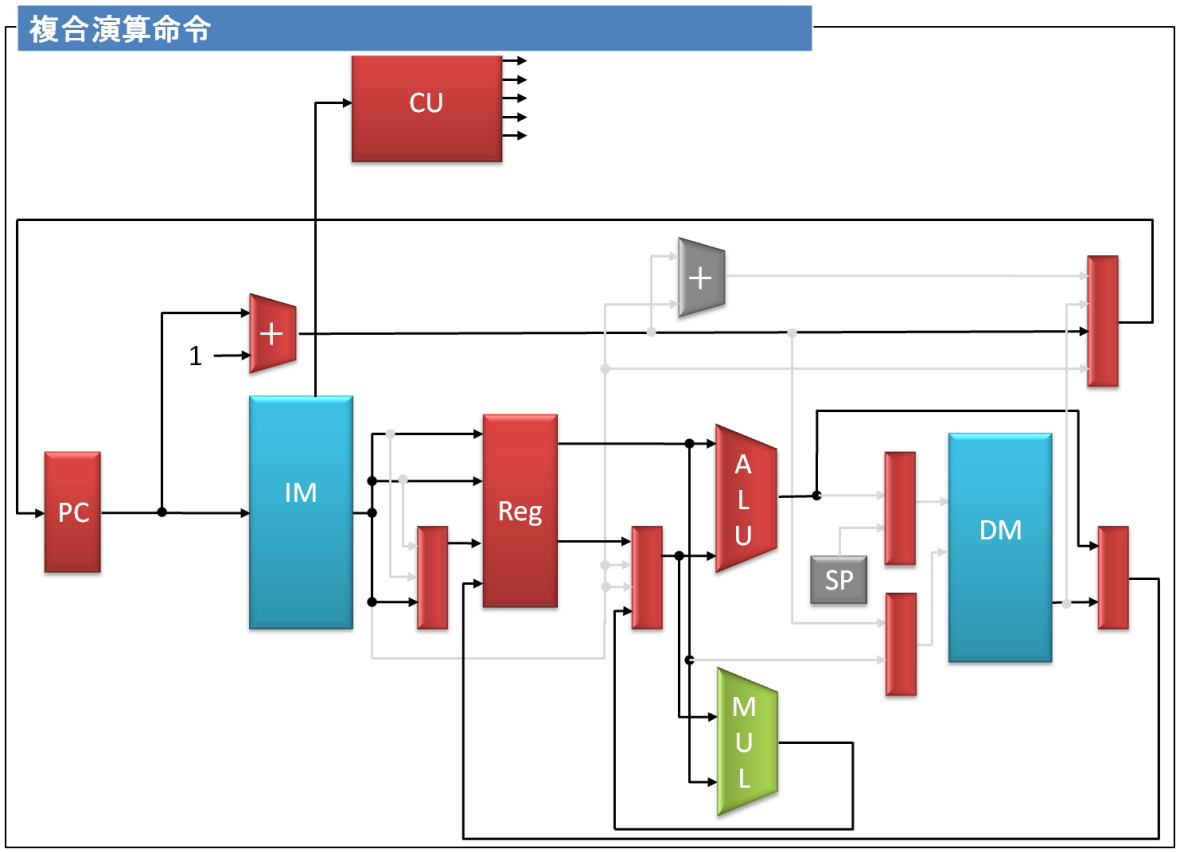

図 11：複合演算命令のデータパス

3.　ザインパターンを用いた MONIT の設計

3.1　デザインパターンの概要

従来のハードウェア，ソフトウェアデザインパターンは仕様モデル，実装モデル，そして設計結果の一部を取り出して再利用してきた．ハードウェアでは IP(Intellectual Property)，ソフトウェアではミドルウェアやライブラリと呼ばれる部分である．つまり，経験豊富な技術者は問題を抱えたときにゼロから解くのではなく，過去に解いた問題があれば答えを再利用しようと考える．開発者が抱える問題は共通なものが多く，解決方法を他者と共有することで, より柔軟に対応できる．

プロセッサ設計に用いるデザインパターンは，過去のプロセッサ設計データを仕様，実装に限らず，分類した構成要素を用いてプロセッサの学習，設計を補助する．本研究で用いるプロセッサ設計の再利用の形を図　に示す．ハード/ソフト協調学習システムを用いた学習では仕様の策定，アプリケーション，プロセッサの設計と幅広い領域を学習することになるため，デザインパターンで過去の設計を再利用することで，学生の負担を軽減でき，より高度なプロセッサの設計，学習に時間を充てることができる．

デザインパターンの構成と，それを用いたプロセッサ設計の手順を図 8 に示す．本研究で提案するデザインパターンは，学習者が MONI やオリジナルプロセッサの新規設計をするときに，ほしい情報のキーワードを入力し，デザインパターンのデータベースから検索する．そして，デザインパターン内にキーワードに類似したパターンが存在すれば学習者へ提供する．

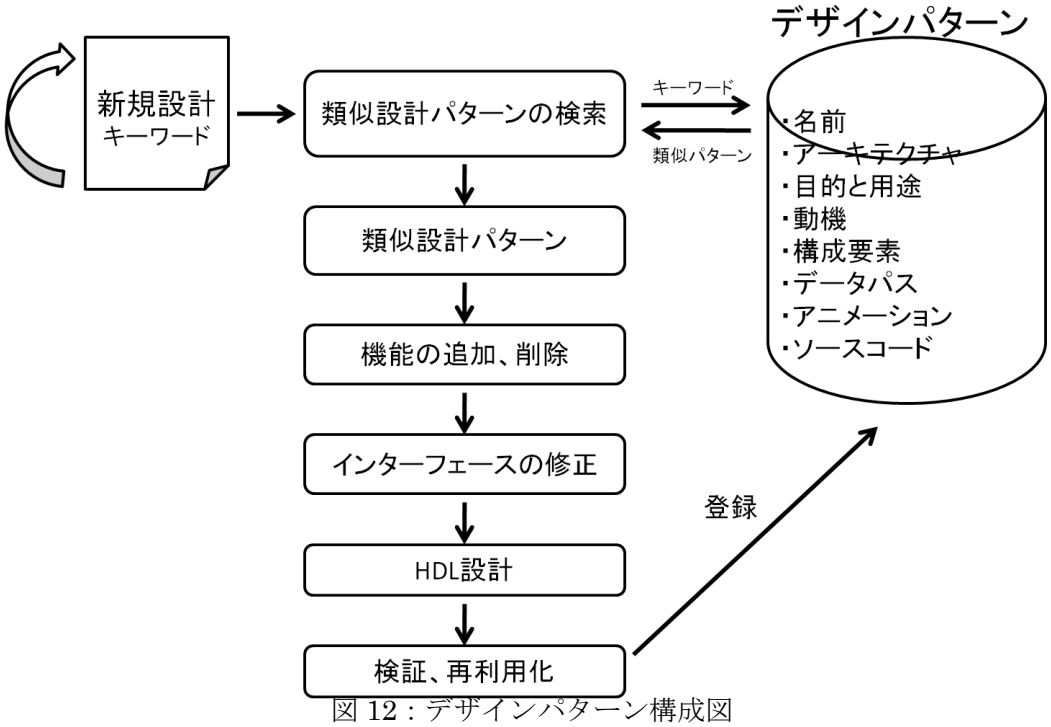

図 12：デザインパターン構成図

本研究では，初めてプロセッサを設計する人，より高度なアーキテクチャを搭載したプロセッサを設計する人それぞれが望む活用しやすい設計資産のデザインパターンへの分割案を提案する．デザインパターンは過去の設計データから仕様と実装を再利用可能な要素ごとに分類することで，設計見本として有効的に使うことができる．設計と学習両面をサポートして，学習者がなるべく独習の形で最適なプロセッサの構成を導きだすための柔軟な設計能力を効率よく学習できるシステムを目指している．

これまで挙げたデザインパターンの設計思想と特徴に基づいて，構成は名前，目的と用途，動機，アーキテクチャ，構成要素，データパス，アニメーションと verilog-HDL ソースコードの 8 要素に分類した．

各設計段階において，学習者は求めるキーワードでデザインパターンに検索をかけてキーワードに類似したパターンがある場合に取り出せる．

新規設計を始めるときは，各種プロセッサアーキテクチャのアニメーションによる細かな回路動作の観察が行えるため，アーキテクチャ学習にかかる時間を短縮できる．また，アーキテクチャ，命令セットやデータパスからプロセッサの長所と短所を把握でき，効率化のポイントや改善点を発見しやすくなるのでプロセッサの仕様イメージをより明確にでき，仕様策定にかかる時間を効率的に短縮できる．

次に設計段階では，デザインパターン上にある単一機能モジュール，機能ブロック，命令形式ごとなどに分類された IP を用いることができる．ハードウェア設計の設計見本として用いるとプログラミング能力の向上と抱えている設計問題の解決策を探すことが可能である．また，設計の再利用では利用したい IP を選択してプロセッサの仕様にあったインターフェースの修正を行い，搭載することで 1 から機能を設計する必要がなくなり，設計期間の短縮ができる．

最後に検証段階では，オリジナルプロセッサと過去のプロセッサの性能比較と検証が行えるため，実際に設計したプロセッサが他のプロセッサと比べたときにどの程度の性能を持つか，また，特徴として挙げたポイントがきちんと設計できているか確認できる．そして，検証の終わったプロセッサをデザインパターンと比較して再利用可能なものがあれば，次の学習をサポートする設計見本として構成要素に分類し，デザインパターンへ登録する．

3.2 Verilog-HDL による設計

MONIT プロセッサの設計には，Verilog-HDL を用いた．表モジュールの一覧を示す．

表 5：Verilog-HDL 記述ファイル一覧

名 前	モジュール名	目 的
(ヘッダファイル)	_header	ビット幅や命令などの定数を定義する
ALU	ALU	命令で指定された演算を行う
ALU入力セレクタ	ALU_MUX	演算に使用する値を選択する
コントロールユニット	CU	全モジュールを制御する
PC	PC	実行する命令アドレスを指定する
PCアドレス加算機	PC_INCR	PCに1加算したものを出力
PCアドレスセレクタ	PC_NEXT	次のPCのアドレスを選択する
REG_IMMセレクタ	PC_RELATIVE_MUX	相対ジャンプを行うアドレスを選択する
レジスタ	REG	レジスタファイル 16bit×16word
レジスタ入力セレクタ	REG_ADDR_MUX	レジスタに入力する値を選択する
書込アドレスセレクタ	PC_NEXT_MUX	書込むアドレスを選択する
リセット	(無し)	リセット割込み用、外部入力
無し	(無し)	常にLOWが出力される
オーバーフロー	(無し)	ALUの演算結果のオーバーフロー
(JINTプロセッサトップモジュール)	MONIT	シミュレーション用トップモジュール
データメモリ	DM	シミュレーション用データメモリ
命令メモリ	IM	シミュレーション用命令メモリ
(デバッガ接続用トップモジュール)	TOP	デバッガに接続するためのトップモジュール

4. 考察

4.1 デザインパターンの評価

本研究で，デザインパターンを用いてシングルサイクルの MONI と MONIT を設計した．デザインパターンから過去に設計された MONI や，JINT 等の R 形式等のソースコードを参考にて設計した．命令セットは MONI を使用したので，MONI の命令セット学習時間は 0，MONIT は今回追加した命令セット時間とする．

表 6：学習時間全体

学習時間＼工程	ソフトウェア学習		ハードウェア学習	
	アセンブリ	命令セット	HDL 設計	HDL シミュレータ
MONI	25	0	80	40
MONIT	13	3	30	20

4.2 MONIT 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 ISE11 を使用した。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 ボード上に実装する際に、設計したプロセッサのデバッガ接続用トップモジュールとプロセッサデバッガを接続した。このとき、命令メモリとデータメモリはプロセッサデバッガから提供されるものを使用する。

表 7：乗算器搭載プロセッサ MONIT 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）
TOP	1643	3056	1195	64.787

MONITプロセッサのFPGA使用率はスライス数が85%、LUT数が79%、フリップフロップ数が31%となった。最高動作周波数は15.435MHzであった。
1 から N までの和や，スタックポインタを用いたプログラム，複合演算命令等で動作を確認した。

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て、シングルサイクルの乗算器搭載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。本研究を通して、複合演算命令処理の目的と原理・仕組みをハードウェアとソフトウェアの両方の観点から学習を行えた。 今後の課題としては、ハードウェア面ではより多くの行列式の計算や同時に複数に対応すること、またマルチサイクルやパイプラインに対応したプロセッサの設計を行うことがあげられ、ソフトウェア面では仮想シミュレータ環境があるとより効果的に割込み原理の理解度を高めることができるため、仮想シミュレータ環境の開発を進めることがあげられる。

謝辞

本研究の機会を与えて下さり、数々の助言を頂きました山崎勝弘教授に心より感謝致します。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安倍 厚志氏とPISHVA JOHN CYRUS P 氏をはじめ，励ましの言葉や貴重な御意見を頂きました本研究室の皆様に心より感謝致します。

参考文献

[1] 志水健太：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

[2] 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

[3] 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

[4] 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情報学科卒業論文、2008

[5] 志水健太：命令セット定義ツール・シミュレータ・アセンブラ設計仕様書・使用方法説明書，2009

[6] 志水健太：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

[7] 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

[8] 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

[9] 大八木睦：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

[10] 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

[11] PISHVA JOHN CYRUS P：ハード/ソフト協調学習システムを用いた割り込みプロセッサの設計，立命館大学理工学部情報学科卒業論文，2008

[12] 安倍厚志：各種プロセッサアーキテクチャの設計に基づいたデザインパターンの検討，立命館大学理工学研究科修士論文，2011

[13] David A.Patterson / John L.Hennessy：コンピュータの構成と設計　第 3 版　下，日経 BP 社，2006.

[14] 小林優：入門 Verilog HDL 記述－ハードウェア記述言語の速習＆実践（Design wave basic）[単行本]，CQ 出版社，2004.

[15] booran：ためになるページ・Verilog，
http://www.booran.com/menu/verilog/

付録

加減算・論理演算

R 型	*I5 型*		*記述*	*動作*
ADD	ADDI		加算	
			ADD $rd $rs $rt	Reg[rd] ←Reg[rs] + Reg[rt]
			ADDI $rt $rs #imm	Reg[rt] ←Reg[rs] + imm(5bit)
SUB	SUBI		減算	
			SUB $rd $rs $rt	Reg[rd] ←Reg[rs] - Reg[rt]
			SUBI $rt $rs #imm	Reg[rt] ←Reg[rs] - imm(5bit)
AND	ANDI		論理積	
			AND $rd $rs $rt	Reg[rd] ←Reg[rs] & Reg[rt]
			ANDI $rt $rs #imm	Reg[rt] ←Reg[rs] & #imm(5bit)
OR	ORI		論理和	
			OR $rd $rs $rt	Reg[rd] ←Reg[rs] | Reg[rt]
			ORI $rt $rs #imm	Reg[rt] ←Reg[rs] | #imm(5bit)
XOR	XORI		排他的論理和	
			XOR $rd $rs $rt	Reg[rd] ←Reg[rs] ^ Reg[rt]
			XORI $rt $rs #imm	Reg[rd] ←Reg[rs] ^ #imm(5bit)
NOT			ビット反転	
			NOT $rd $rs $rt	Reg[rd] ←~Reg[rs] ##rt==unkown

セット命令

R 型	*I5 型*	*I8 型*	*記述*	*動作*
SLT	SLTI		値の比較　小さければ１、そうでなければ０ が代入される	
			SLT $rd $rs $rt	Reg[rd] ← 1 (Reg[rs] < Reg[rt]) else 0
			SLTI $rt $rs #imm	Reg[rt] ← 1 (Reg[rs] < #imm(5bit) else 0
SGT	SGTI		値の比較　大きければ１、そうでなければ０ が代入される	
			SGT $rd $rs $rt	Reg[rd] ← 1 (Reg[rs] > Reg[rt]) else 0
			SGTI $rt $rs #imm	Reg[rt] ← 1 (Reg[rs] > imm(5bit)) else 0
SLE	SLEI		値の比較　以下ならば１、そうでなければ０ が代入される	
			SLE $rd $rs $rt	Reg[rd] ← 1 (Reg[rs] <= Reg[rt]) else 0
			SLEI $rt $rs #imm	Reg[rt] ← 1 (Reg[rs] <= imm(5bit)) else 0

SGE	SGEI		値の比較　以上ならば１、そうでなければ０ が代入される	
			SGE $rd $rs $rt	Reg[rd] ← 1 (Reg[rs] >= Reg[rt]) else 0
			SGEI $rt $rs #imm	Reg[rt] ← 1 (Reg[rs] >= imm(5bit)) else 0
SEQ	SEQI		値の比較　同じならば１、そうでなければ０	
			SEQ $rd $rs $rt	Reg[rd] ← 1 (Reg[rs] == Reg[rt]) else 0
			SEQI $rt $rs #imm	Reg[rt] ← 1 (Reg[rs] == imm(5bit)) else 0
SNE	SNEI		値の比較　異なれば１、そうでなければ０ が代入される	
			SNE $rd $rs $rt	Reg[rd] ← 1 (Reg[rs] != Reg[rt]) else 0
			SNEI $rt $rs #imm	Reg[rt] ← 1 (Reg[rs] != imm(5bit)) else 0
		LDHI	レジスタの上位８ビットに即値を設定	
			LDHI $rs #imm	Reg[rs[15,8]] ← #imm(8bit)
		LDLI	レジスタの下位８ビットに即値を設定	
			LDLI $rs #imm	Reg[rs[7,0]] ← #imm(8bit)

シフト命令

R 型	*I5 型*		*記述*	*動作*
SLL	SLLI		左論理シフト	
			SLL $rd $rs $rt	Reg[rd] ← Reg[rs]{=rs[14,0],0} ## rd=0
			SLLI $rt $rs #imm	Reg[rd] ← Reg[rs] << #imm
SRL	SRLI		右論理シフト	
			SRL $rd $rs $rt	Reg[rd] ← Reg[rs]{=0,rs[15,1]} ## rd=0
			SRLI $rt $rs #imm	Reg[rd] ← Reg[rs] >> #imm
SRA	SRAI		右算術シフト	
			SRA $rd $rs $rt SRAI $rt $rs #imm	Reg[rd] ← Reg[rs]{rs[15],rs[15,1]} ## rd=0 If (!rs[15]) Reg[rd] ← Reg[rs] >> #imm Else Reg[rd] ← (16'hffff << #imm) | (Reg[rs] >> #imm)

メモリアクセス命令

I5 型			*記述*	*動作*
LD			データメモリからレジスタファイルへのロード(レジスタ間接)	
			LD $rt MEM[$rs]	Reg[rt] ←(DataMem[Reg[rs]])

ST			レジスタファイルからデータメモリへのストア(レジスタ間接)	
			ST MEM[$rt] $rs	DataMem[Reg[rt]] ← Reg[rs]

スタック操作命令

I8 型			*記述*	*動作*
PUSH			レジスタの値をスタックにプッシュ	
			PUSH $rs	DataMem[SP-1] ← Reg[$rs] SP ← SP - 1
POP			スタックからレジスタにポップ	
			POP $rs	Reg[$rs] ← DataMem[SP] SP ← SP + 1

条件分岐命令

I8 型			*記述*	*動作*
BEQZ			レジスタの値が0の時、ジャンプ	
			BEQZ $rs FLAG	if (Reg[rs] == 0) PC ← PC + 1 + (imm)8
BNEZ			レジスタの値が0でない時、ジャンプ	
			BNEZ $rs FLAG	if (Reg[rs] != 0) PC ← PC + 1 + (imm)8

無条件分岐命令

J 型			*記述*	*動作*
JUMP			即値による絶対ジャンプ	
			JUMP FLAG	PC ← #imm
CALL			サブルーチンへ分岐(絶対番地) & PC+1 をスタックへ格納	
			CALL FLAG	PC ← #imm DataMem[SP-1] ← PC+1 SP ← SP -1
RETURN			データメモリより PC+1 を呼び出しサブルーチンより復帰	
			RETURN	PC ← DataMem[SP] SP ← SP + 1

その他命令				
I8 型			*記述*	*動作*
NOP			何もしない	
			NOP	
HALT			停止	
			HALT	

擬似命令				
			記述	*動作*
COPY			値のコピー	
			COPY $rt $rs	ADDI $rt $rs #0
CLEAR			値に０を代入	
			CLEAR $rs	SUB $rs $rs $rs