卒業論文

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

氏名　　：西田　範行
学籍番号：2210020340-9
担当教員：山崎　勝弘
提出日　：２月　22 日

内容梗概

本論文では、ＬＳＩ設計開発技術者の育成を促進するハード/ソフト協調学習システムについて解説した後、ハード/ソフト協調学習システムの構成要素である命令セット定義可能な汎用シミュレータについて述べる。

ハード/ソフト協調学習システムは、以前から同研究室で検討されているものである。ハード/ソフト協調学習の仕組みを知った上で、拡張案を考察しより効率的な学習体系を実現させる必要がある。本論文では、その拡張案および拡張によるメリットについても記述している。以下に記述する汎用シミュレータは、この拡張案の一部となっている。

汎用シミュレータでは、命令のビット幅、フィールドの数と各フィールドのビット幅、レジスタ数、メモリ領域といった命令セットそのものをユーザが定義できることが特徴となっている。また、AND や ADD などあらかじめ用意してある基本的な動作を組み合わせることによって、ユーザの要求する動作を実現することが可能である。シミュレーション結果として得られる情報は、命令の総実行回数、実行後の各レジスタの値、実行後のメモリの使用状況となっている。ブレークポイントの設定や実行回数の指定を行うことで、実行途中の各レジスタの値およびメモリの使用状況を確認することも可能となっている。

汎用シミュレータのみではなく、同研究室の富樫氏によるハード/ソフト協調学習システムの構成要素である汎用アセンブラと命令セット定義ツール、中川氏によるインターフェースを利用してプロセッサの知識を学習することが望ましい。

目次

１．はじめに

１．１　研究背景

科学技術の目覚しい発展により、パソコンなど現代の電子技術を駆使した商品の質の向上を実現させた。それに伴い消費者の商品に対する要求も高度になっている。しかし、現時点での要求の実現に成功しても、より条件の厳しい要求が発生することは明白である。その結果、短期間で以前より高性能な商品開発を実現させることが必須となる。

現時点で消費者の要求には、ある機能に特化させることによる商品の高性能化、小型化が含まれている。その要求を実現するために利用するのがシステム LSI である。システム LSI を用いることで、マイクロプロセッサ、メモリ、ASIC、アナログ回路などをワンチップにまとめることができる。この特徴を利用して必要な機能だけをワンチップにまることで、ユーザの要求をみたすのである。現在、システム LSI はパソコン、自動車、携帯電話など様々な製品に搭載されており、製品の基幹部品として必要不可欠なものとなっている。そして、今後もシステム LSI の需要は上昇すると思われる。

近年の LSI 技術の発展はすばらしいものがあり、ワンチップに載せられるトランジスタ数は数百万を超えている。［9］半導体集積技術の向上によって、ワンチップ上に搭載できる機能を増加している。そのため、消費者の要求である商品の高性能化、小型化を実現するための土台は確実に存在するのである。しかし、その技術を有効利用できる人材とは、ハードウェア・ソフトウェアの知識を持った開発者に限られてしまう。なぜなら、ハードウェアでの設計の場合高性能の計算が可能であるが設計コストが高い、ソフトウェア設計の場合設計が容易だが性能面でハードウェアに劣るという特徴は理解していても、ハード・ソフトの両方を理解していないと、ハードとソフトをどこで切り分けシステムを設計すれば効率的かわからないのである。残念ながら、ハード・ソフトの知識を持ち、ＬＳＩ技術を有効利用できる技術者は現時点では少数しか存在しないのが事実である。よって、LSI を用いた開発に必要不可欠なハードウェア/ソフトウェア両方の知識を兼ね備えた人材育成が急務となっている。

私の所属する研究室では、その人材育成を補助するためハード/ソフト・コラーニングシステムが考案された。このハード/ソフト・コラーニングシステムとは、ハードウェアの分野とソフトウェアの分野を相互に関連づけながら両方学ぶという学習システムである。学習体系は、ソフトウェアでプロセッサアーキテクチャを理解した後ハードウェアによるプロセッサの設計となっている。現段階で、ソフトウェア学習では単一サイクル、マルチサイクル、パイプラインに対応したＭＯＮＩプロセッサ［2］が作成され、その動作を確認できるＭＯＮＩシミュレータの作成が完了しており、プロセッサおよびアーキテクチャの理解を促すことに成功している。またハードウェア学習の面では、プロセッサデバッガ［6］が作製され、ＨＤＬにおけるプロセッサ設計およびＦＰＪＡ上でのプロセッサの動作確認

ができる環境がある。しかし、現在私の研究室で用意されているプロセッサは MONI プロセッサ、SOAR プロセッサ［3］など命令セットが限られているため、自由度の高いプロセッサの設計を行える環境に達していない。より効果的な学習をするには、ハードウェアによるプロセッサ作製を行う前に、ソフトウェア学習の段階で独自の命令セットを考案し、その動作を確認することが必要である。

１．２　研究目的

本研究では、ＬＳＩ設計開発に必要不可欠であるハードウェアとソフトウェアの知識を学習するシステムの提案および学習システムの構成要素である汎用シミュレータを設計することを目的とする。

上記の研究背景で示唆したとおり、半導体集積技術の向上によってハードウェア/ソフトウェアの両方を理解したうえでＬＳＩ設計を進めることが望まれている。しかし、現在はハードウェア/ソフトウェアを両方理解した人材を育成するシステムは確立されておらず、技術の向上に対する人材の供給スピード間に合っていない状態である。

私の研究室で考案されているハード/ソフト・コラーニングシステムでは、研究室で開発したＭＯＮＩプロセッサおよびＭＯＮＩシミュレータによって、ＭＯＮＩに対応したプロセッサの構造および動作を理解することは可能であるが、プロセッサの知識を十分身につけるに至らない。しかし、従来の学習体系では、ＭＯＮＩの知識を獲得した段階で、次にプロセッサの設計という手順となっている。つまり、この学習体系では、十分なプロセッサ知識を得る前にプロセッサの設計段階に突入してしまうのである。様々なプロセッサを設計し、プロセッサの知識を得るためには、ＭＯＮＩのみならず、様々な命令セットのプロセッサの構造および動作を知ることで、自分で命令セットを考案できる状態でなければならない。また、FPGA ボード上でのデバッグが困難、LED でしか内部レジスタの確認ができないなどの不具合が生じている［3］そこで、プロセッサの理解を促すために可変命令セットに対応した汎用アセンブラ、汎用シミュレータがハード/ソフト・コラーニングシステムに追加された。

上記の通りこのシステムはまだ発展途上であり、改善の余地は十分にあると考えられており、ハード/ソフト・コラーニングシステムを拡張し、より効率的なハードウェア・ソフトウェアの協調学習を促進させることが必要となる。

私は拡張ハード/ソフト・コラーニングシステムを構築するために、その構成要素のひとつである汎用シミュレータの設計を担当した。よって、私の研究はハード/ソフト協調学習システム全体の構築ではなく、汎用シミュレータの設計がメインとなる。

１．３　関連研究

各大学で開発した教育用プロセッサをもちいた学習システムを採り上げた。

・ PICO（慶應義塾大学）［１１］
教育用のマイクロプロセッサであり、命令セットのフレームワークは複数用意されている。教育体系としては、ユーザがフォワーディングやストールなどの機能を追加し独自のパイプラインプロセッサを設計して学習するものとなっている。

・ KITE（熊本大学）［１２］
KITE マイクロプロセッサとは、教育用１６ビット・マイクロプロセッサである。基本的な命令セットのみが用意されていて、アキュムレータ型のマイクロプロセッサである KITE-１、割り込み処理やメモリ管理などの機能を追加した KITE-2 が存在する。ともに可観測性を含んでいる。

・ City-1（広島市立大学）［１３］［１４］
ユーザが独自の命令セットを考案でき、それを元に独自のプロセッサを設計させる。「自分で作る」を重視しているため、ボードの製作まで行っている点がユニークである。

１．４　研究内容

本研究では、ハード/ソフト・コラーニングシステムを構築させる上で必要な汎用シミュレータの設計を行った。

この汎用シミュレータでは、自分で考えた命令セットを用いた場合のプロセッサの内部動作（レジスタの変化など）を確認できるものとなっている。つまり、可変な命令長、フィールド数およびフィールド幅、レジスタ数およびビット幅、メモリの領域に対応したシミュレータとなっている。動作記述は基本的な２１命令の動作しか用意していないが、それらを組み合わせることで複雑な命令を考案することが可能となっている。また、同研究室で開発した MONI や SOAR のプロセッサのテンプレートは用意されているので、各プロセッサ専用のシミュレータを用いた場合との比較を行うことで、このシミュレータの精度を確認する。なお、この研究には、マルチサイクルでの実行、パイプラインでの実行など複数の実行モードを追加する余地はあるが、今回は動作の確認を目標としているため、実行モードは単一サイクルのみとなっている。

2．ハード/ソフト・コラーニングシステム

2．1　システムの概要

ハード・ソフトの両方を理解できる人材を育てるシステムであり、プロセッサアーキテクチャを意識したプログラミング能力を向上させることを目的としている。

ソフトウェア面では、アーキテクチャが可変な命令セットシミュレータを用いて、アーキテクチャの仕組みの理解を促す。また、アセンブリ言語や C 言語で書かれたプログラムの評価を行う。ハードウェア面では、ソフトウェア上での命令シミュレーションにより得たプロセッサアーキテクチャの知識を基に HDL によるプロセッサ設計を行い、そのプロセッサを FPGA ボードに取り込み、その評価を行うことによって、プロセッサアーキテクチャの理解を促す。

2．2　学習体系

学習体系の大きな流れとして、まず C 言語やアセンブリ言語を用いてソフトウェア開発することによって、ソフトウェア学習を行いプロセッサアーキテクチャの理解を促進する。次に実際に HDL でハードウェアの設計を試み、シミュレータだけでは理解できない遅延などハードウェア特有の問題を解決できる能力を向上させる。図１に既存の学習体系を示す。

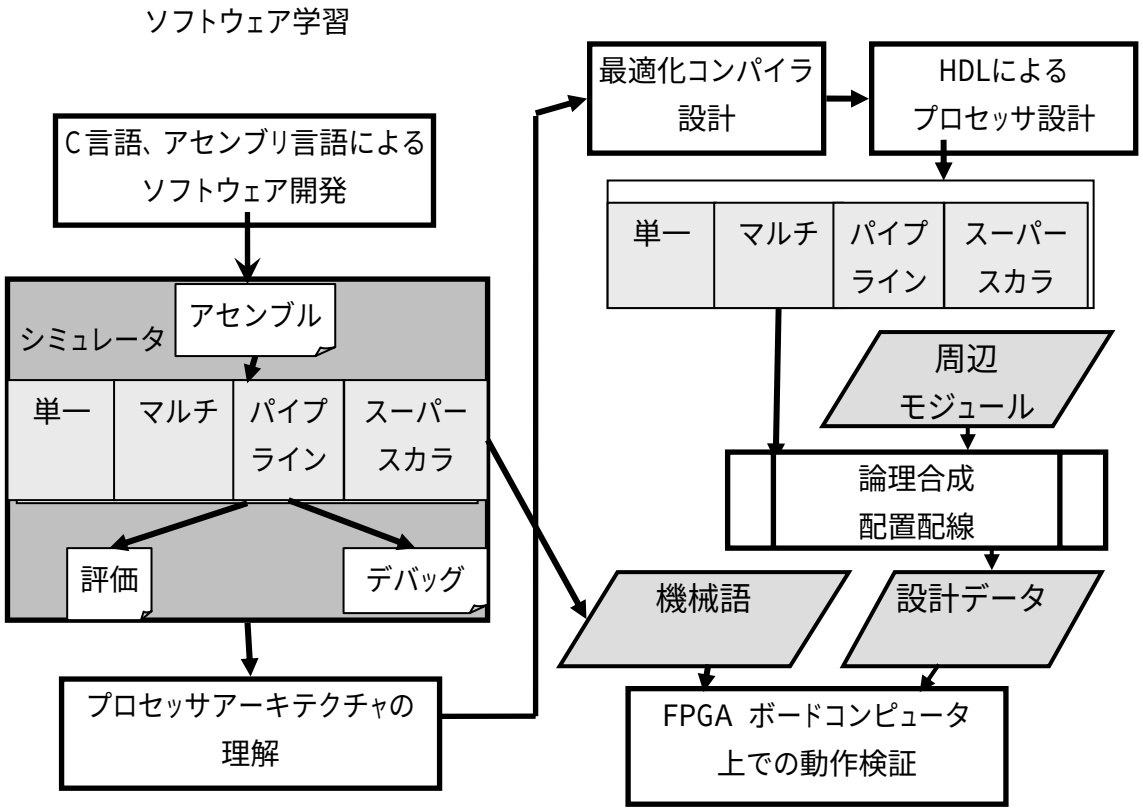

図 1：既存の学習体系

既存の学習体系［3］は、各種アーキテクチャの理解の後に、すぐハードウェアの設計に取り掛かる手順となっている。しかし、この学習体系では、限られた命令セットしか用意されておらず、より自由度の高いプロセッサアーキテクチャの作製を期待することは困難である。よって、様々な命令セットを各個人が考察し、その動作を確認できる環境を整える必要がある。その環境を付け加えたシステムが図２に示す拡張コラーニングシステムの学習体系である。

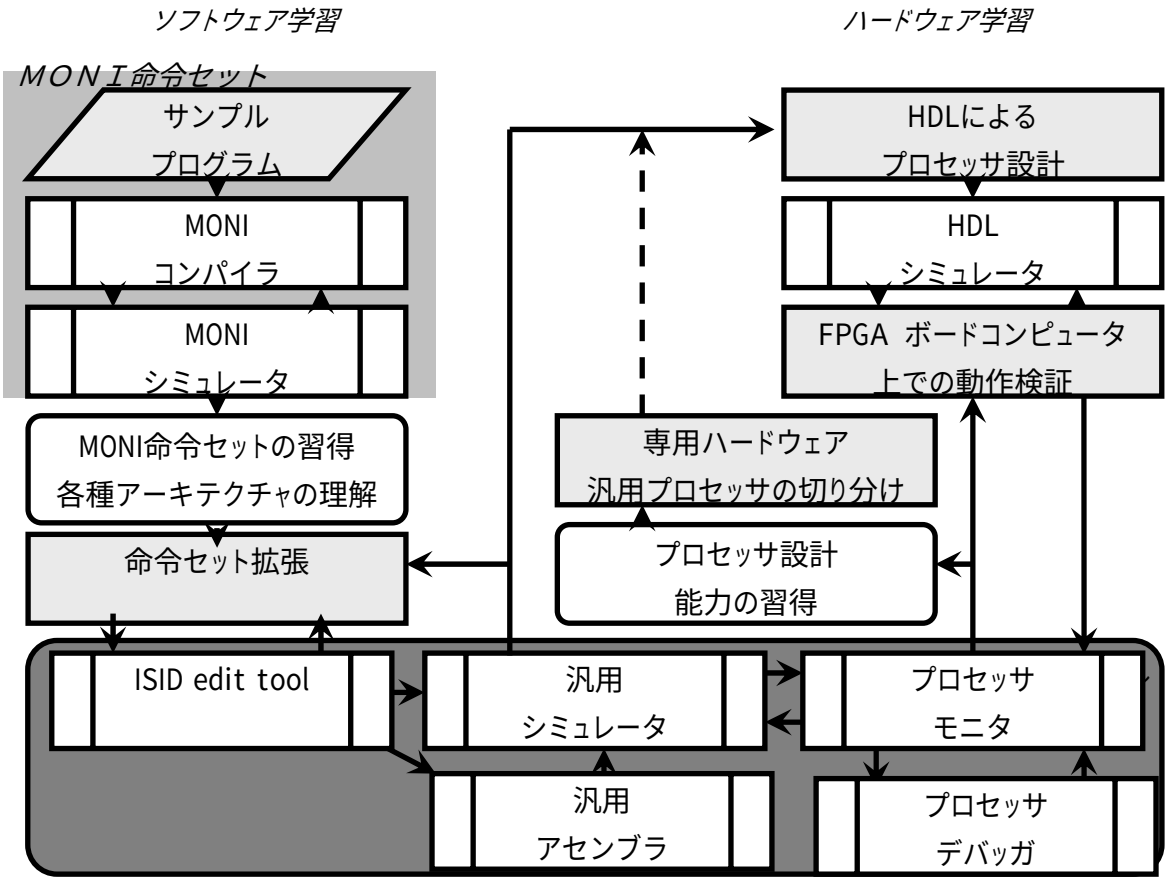

図 2：拡張した学習体系

前述のとおり既存の学習体系にユーザが独自に命令セットを考案し、そのアーキテクチャを実際に動かし、その動作を確認する手順を追加したものが拡張案である。また、汎用アセンブラ、汎用シミュレータはユーザが定義した命令セットに対応できる仕様となっている。拡張案では、ユーザが独自に命令セットを考案してアーキテクチャのイメージをつかむことで、更なるアーキテクチャの理解の促進を手助けする効果が期待できる。

２．３　システムの構成要素

・ ＭＯＮＩコンパイラ［2］

同研究室で設計した、サンプルプログラムを機械語に変換するツールである。現在アセンブリ言語で書かれたプログラムを機械語に変換可能である。

←（難波さんと話し合った結果、現段階ではアセンブリ言語しか対応していないが、今後 C 言語やほかの言語に対応させる予定なので、学習体系としては「MONI アセンブラ」ではなく「MONI コンパイラ」としました。）

・ ＭＯＮＩシミュレータ［2］

ＭＯＮＩプロセッサをシミュレートさせるツールである。複数のアーキテクチャに対応しており、プロセッサの構造や動作の知識を得るために作成されたツールである。

・ ＨＤＬシミュレータ［1］

ＨＤＬで書かれた言語がＦＰＧＡ上でどのように動作するかを確認するツール。ハードウェア学習では、ＦＰＧＡの実機上に載せる前にＨＤＬで動作の確認を行う。

・ ＩＳＩＤ edit tool（命令セット定義ツール）［5］

命令セットの定義を援助するツールである。ISID edit tool の指示に従ってビット長やレジスタ数などの命令セットを定義することで、ユーザ独自の命令セット情報を定義することができる。

・ 汎用シミュレータ

アセンブリソースプログラムを読み取り、ユーザが設定した命令セットに照らし合わせて、シミュレートするツールである。アセンブリソースプログラムには、命令セットで定義された命令が使われている。

・汎用アセンブラ［4］

アセンブリソースプログラムをユーザが設定した命令セットに照らし合わせて、機械語を生成するツールである。命令セット情報を変更することで、可変長のプロセッサに対応している。

・ プロセッサモニタ［6］

ホスト PC 上で、FPGA ボードに対するデータの送受信、MPU を実行させるなどのインターフェース上のコマンドの生成を行うツールである。データやコマンドを FPGA 上に送信することでプロセッサのデバッガをサポートする。

・ プロセッサデバッガ［6］

FPGA 上では困難なプロセッサのデバッグを、FPGA ボードと接続したホスト PC 上で行うためのツールである。ホスト PC 上では、プログラムプロセッサの実行を途中で停止させて、レジスタやメモリの値を確認・変更可能である。さらに、ボード上で行っていたスイッチの切り替えなどをホスト PC 上でも操作することができる。

３．汎用シミュレータ

３．１　要求仕様

この研究で設計した汎用シミュレータは、可変長の命令セットに対応した命令セットシミュレータである。

単一サイクルのアーキテクチャに対応しており、全命令実行、ブレークポイント指定実行、命令実行回数指定実行の３つの実行モードが用意されている。実行後には、実行後のレジスタの値、実行後のメモリの値、実行回数、分岐回数が表示可能である。レジスタの値・メモリの値の初期値はユーザが指定可能である。

ブレークポイント実行や命令回数実行を利用することで、プロセッサの内部動作の途中経過を確認することができ、プログラムを評価できる。

３．２　汎用シミュレータの機能

汎用シミュレータの機能は大きく３つに分けることが可能である。その内容は、シミュレーション設定、実行モードの設定、実行結果の表示である。表１～表３に機能別にユーザが使用可能なコマンドを示す。

表 1：シミュレーション設定

コマンド名	機能
SetISI()	命令セット情報の指定
SetACode()	シミュレートさせたいソース（アセンブリソース）の指定
SetR()	レジスタの初期値設定
SetM()	メモリの初期値設定

表 2：実行モードの設定

コマンド名	機能
RunALL()	全命令実行
RnuBreak()	ブレークポイント実行
RnuInst()	実行命令回数指定実行

表 3：実行結果の表示

コマンド名	機能
PrintReg()	レジスタの値を表示
PrintMem()	メモリの値を表示
PrintInstNum()	命令実行回数の表示
PrintBranchNum()	分岐回数の表示

シミュレート前の設定では、命令セットの指定、アセンブリソースの指定は必須であるが、レジスタおよびメモリの初期値の設定は任意である。レジスタおよびメモリの初期設定を行わない場合、すべてのレジスタの値・メモリの値が０に設定されている。
命令セット、アセンブリソース、レジスタファイル、メモリファイルの記述例は論文後方の付録に記述した。

実行モードの設定では、３つの実行モードのうち１つを指定する。ブレークポイントの指定や実行命令回数の指定で、ソースを最初から途中まで実行可能だが、ソースを途中から最後まで実行させることはできない。

シミュレート結果表示は、シミュレーション実行前にはできない。なぜなら命令実行回数や分岐回数はシミュレーション後に得られるものであり、シミュレーション後にそのデータが保持されるからである。

３．３　汎用性の実現方法

汎用シミュレータでは、可変長の命令セットに対応した反面、実行できる命令が１９種類に限定される。表４に２１種類の命令の詳細を示す。

表 4：汎用シミュレータで使用可能な命令

命令名	動作説明	記述例	記述例の説明
AND	レジスタの論理積	AND $2 $0 $1;	r0 と r1 の論理積を r2 に書き込む
OR	レジスタの論理和	OR $2 $0 $1;	r0 と r1 の論理和を r2 に書き込む
XOR	レジスタの排他的論理和	XOR $2 $0 $1;	r0 と r1 の排他的論理和を r2 に書き込む
NOT	レジスタのビット反転	NOT $1 $0;	r0 のビット反転を r1 に書き込む
ADD	レジスタの加算	ADD $2 $0 $1;	r0 と r1 の加算を r2 に書き込む
SUB	レジスタの減算	SUB $2 $0 $1;	r0 と r1 の減算を r2 に書き込む
MUL	レジスタの乗算	MUL $2 $0 $1;	r0 と r1 の乗算を r2 に書き込む
DIV	レジスタの除算	DIV $2 $0 $1;	r0 と r1 の除算を r2 に書き込む
SLL	左論理シフト	SLL $1 $0 i3;	R0 を 3 ビット左論理シフトして r1 に書き込む
SRL	右論理シフト	SRL $1 $0 i2;	r0 を 2 ビット右論理シフトして r1 に書き込む
SRA	右算術シフト	SRA $1 $0 i3;	r0 を 3 ビット右算術シフトして r1 に書き込む
LD	メモリからレジスタへ書き込む	LD　$1 $0;	メモリ番地 0 の値を r1 に書き込む
ST	レジスタからメモリへ書き込む	ST　$1 $0;	r1 の値をメモリ番地 0 に書き込む
STR	即値をレジスタへ書き込む	STR　$0 i5;	r0 に即値5を書き込む
NOP	動作なし	NOP;	何もしない
SLT	レジスタの比較 (大小関係)	SLT $2 $0 $1;	(r0 < r1)なら r2 に 1 を、違うなら 0 を代入
SEQ	レジスタの比較 (ゼロ判定)	SEQ $r3 $r1 $r2;	r1 と r2 の値が等しい場合 r3 に1を代入
BNZ	分岐 (ゼロ判定)	BNZ $0 $1;	r0 の値が 0 でないとき r1 の値へ分岐
BEZ	分岐 (ゼロ判定)	BEZ $0 $1;	r0 の値が 0 のとき r1 の値へ分岐
JUMP	レジスタの値へジャンプ	JUMP $r0;	r0 の値へジャンプ
HALT	停止	HALT;	プログラムの停止

基本的な命令は用意されているので、複数の命令を組み合わせることで、複雑な命令にも対応可能である。

（1）レジスタの値と定数を加算する命令（Add Immediate）の場合

STR 命令で即値をテンポラリーレジスタに代入した後、ADD 命令を用いてレジスタ同士の加算を行うことで実現できる。

```
STR     temp    im;
ADD     rd      rs      temp;     という手順で命令を行う。
```

（2）ゼロ判定で比較を行う命令（Set Not Equal）の場合

SLT 命令でレジスタの rs が rt より小さい場合 rd に１を設定、rs が rt より大きい場合 temp に１を設定する。その後 XOR 命令を用いて、どちらか片方にのみ当てはまる場合 rd に１を設定することで実現できる。

```
SLT     rd      rs      rt;
SLT     temp    rt      rs;
XOR     rd      rd      temp;     という手順で命令を行う。
```

（3）レジスタの比較を行う命令（Set Greater Equal）の場合

SLT 命令でレジスタの rs が rt より小さい場合 rd に１を設定、rs が rt と等しい場合 temp に１を設定する。その後 OR 命令を用いて、どちらかに当てはまる場合 rd に１を設定することで実現できる。

```
SLT rd rs rt;
SEQ temp rt rs;
OR rd rd temp;        という手順で命令を行う。
```

（4）下位８ビットの値を書き換える命令（RDLI）の場合

SRL 命令でレジスタの値を８ビット右シフトした後に、SLL 命令でレジスタの値を右に上位８ビットシフトさせる、最後に ADD 命令でレジスタの値に即値を加えることで実現できる。

```
SEQ temp temp temp;
ADD temp temp temp;
ADD temp temp temp;
ADD temp temp temp;
SRL rd rd temp;
SLL rd rd temp;
STR temp im;
ADD rd rd temp;       という手順で命令を行う。
```

（５）スタックされたデータを取り出し PC に代入する命令（RETURN）の場合

LD 命令で SP が示すデータをプログラムカウンタ（PC）に書き込む、SEQ 命令を用い temp に１を代入する。最後に PC、SP の値に１を加えることで実現できる。

```
LD PC SP;
SEQ temp temp temp;
ADD PC PC temp;
ADD SP SP temp;
```

という手順で命令を行う。

STR、ADD,SLL などの命令は、汎用シミュレータ対応の命令である。また、テンポラリーレジスタである temp、プログラミングカウンタを示す PC,スタックポインタを示す SP は、あらかじめ汎用シミュレータ内に用意されている。よって両方とも固定である。しかし、レジスタの値である rd、rs　と即値である im といった名前はユーザが決めるものであり、固定ではない。

４．C++による汎用シミュレータの試作

４．１　開発環境

開発ツールとして Microsoft 社が提供する Visual C++6.0 を利用した。今後、私のシミュレータが様々な人に評価され、改善されることを期待し、オブジェクト指向を考慮したプログラミングを用いて設計した。

４．２　モジュール構成

モジュールは、シミュレーション命令実行情報変換モジュール（SITT）とシミュレーション実行モジュール（Intepreter）の２つに大別できる。

（１） SITT

命令セット情報、アセンブリコードを読み取り、アセンブリコードを Interpreter で実行可能なアセンブリソースに変換するモジュールである。以下に SITT で使用する主な関数を記述する。

a) SetRMInfo() ： 命令セットからレジスタとメモリの情報を得る
b) SetField() ： 命令セットからフィールド情報を得る
c) SetFormat() ： 命令セットからフォーマット情報を得る
d) SetInst() ： 命令セットから命令情報を得る
e) SetBehabvior() ： 命令セットから命令の動作情報を得る
f) SetSimCode() ： 命令セットを参照してアセンブリソースプログラムを汎用シミュレータで実行可能な命令シミュレーションコードに書き換える

（２） Intepreter

レジスタとメモリの初期設定を行った後に、SITT によって書き換えられたソースをシミュレーションさせるモジュールである。シミュレーション結果はモジュール内に保持される。以下に Interpreter で使用する主な関数を記述する。
実行モード別に作成した関数郡である。以下にそれぞれの関数を示す。

a) SetR() ： レジスタの設定１
b) SetRwithD() ： レジスタの設定２（初期値を指定する場合）
c) SetM() ： メモリの設定１
d) SetMwithD() ： メモリの設定２（初期値を設定する場合）
e) SetEnum() ： 実行回数の指定
f) SetBP() ： ブレークポイントの指定
e) Interprrete() ： シミュレーション実行
f) PrintReg() ： レジスタの値表示

g) PrintMem() ： メモリの値表示

h) PrintExeNumber() ： 実行回数の表示

i) PrintBranchNumber()： 分岐回数の表示

図３にモジュール構成のイメージを記述する。

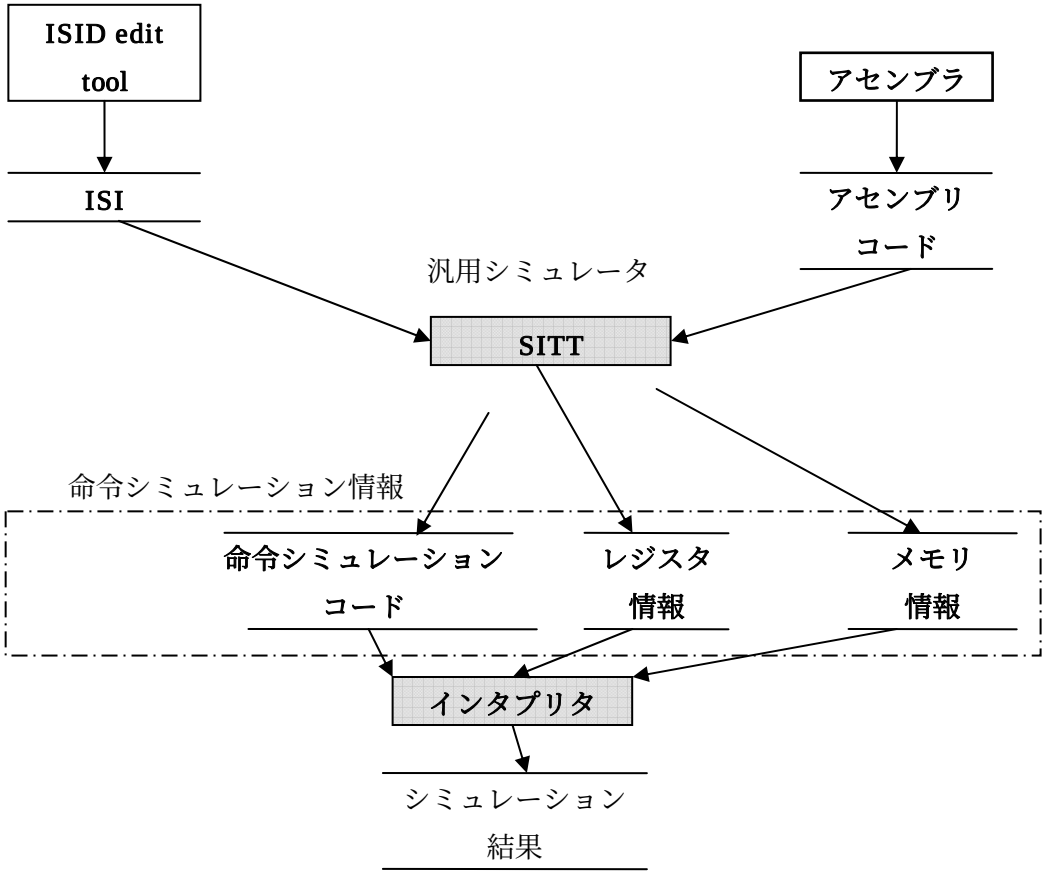

図 3：モジュールの構成のイメージ

汎用シミュレータの大まかな流れは以下のとおりである

１． SITT を利用して、
「アセンブリコード」、「命令セット情報（ISI)」を取得する。

２． SITT を利用して、
「レジスタ情報」、「メモリ情報」、「命令シミュレーションコード」を生成する。

３．インタプリタを利用して
「命令シミュレーション情報」を用いて、シミュレーション実行させ、シミュレーション結果を確認する。

４．３　各モジュールの実現方法

各モジュールは、それぞれが持つ関数によって実現されている。以下に各モジュールの実現方法である処理手順を記述する。

（１）SITT

SITT では、SITT が持つ関数はすべて使用され、処理手順も固定されている。図４に SITT の処理手順を記述する。

図 4：SITT の実行手順

（2）Interpreter

Interpreter では、実行内容やユーザが要求するシミュレーション結果によって使用する関数が違う。また、使用する順序も違う。図５に Interpreter の実行手順を記述する。

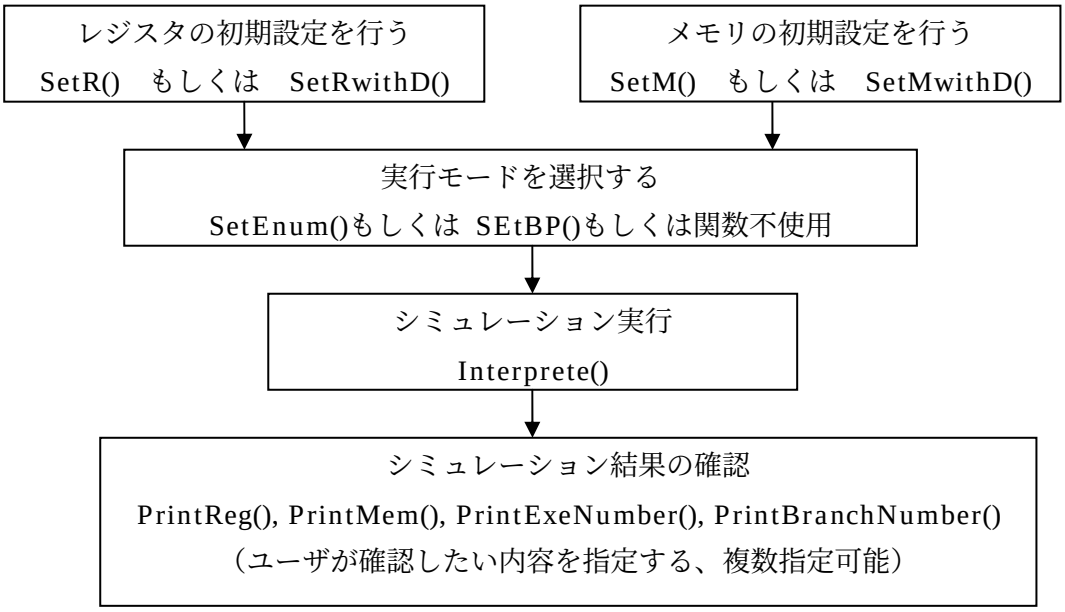

図 5：Interpreter の実行手順

５．汎用シミュレータの実行と考察

５．１　実行結果

シミュレータのテストプログラムとして「１から N までの和」「最大公約数」を作成しシミュレートさせた。

プログラム	使用した値	実行回数指定	命令実行回数	分岐回数	計算結果
1 から N までの和	N = 10	指定なし	57	9	55
	N = 10	10回	10	1	1
	N = 10	1000 回	57	9	55
	N = 100	N = 100	507	99	5050
	N = 100	56 回	56	9	55
	N = 100	57 回	57	9	66
	N = 100	1000 回	507	99	5050
最大公約数	100, 75	指定なし	35	5	25
	124, 36	指定なし	58	11	4
	1983, 622	指定なし	622	116	27

図 6：シミュレート結果

図６にシミュレート結果を載せる。N に１００に設定して実行回数を指定せずにシミュレートさせる場合、１から１００までの和である 5050 という結果が得られた。実行回数を指定する場合、５６回では 55 が得られたが、57 回では 66 という結果となった。N に１０に設定した場合は５７回目の命令がメモリに書き込むという ST 命令であるが、N に１００を設定した場合は５７回目の命令が ADD 命令であるので、このような結果となる。「１から N までの和」「最大公約数」ともに期待通りの結果が得ることができた。　シミュレート結果はユーザが作成する命令セットやアセンブリソースによって違う。今回は汎用シミュレータの精度を確認するため、MONI に対応した命令セットを作成し、MONI シミュレータの実行結果との比較を行った。比較の結果、期待通り命令実行回数、分岐回数ともに等しい結果が得ることができた。シミュレートに使用したプログラムは付録に記述した。

５．２　汎用シミュレータの改善案

今回レジスタ・メモリの値の表示、命令実行回数・分岐回数の表示、ブレークポイントの指定、実行回数の指定という最低限必要な機能を作成することには成功したが、複数のアーキテクチャに対応させるなどの改善の余地は十分にある。以下に現在検討される改善案を列挙する。

（汎用シミュレータ内部）
・マルチサイクルやパイプラインなど複数のアーキテクチャへの対応
・　総クロック数の確認
・　フォワーディング回数の確認
・　ストール回数の確認
・　１クロック実行の実現
・　指定箇所からの実行
・　シミュレーション途中でのレジスタ・メモリの書き換え

（外部とのつながり）
・　操作性のよいインターフェースの作成
・　汎用アセンブラとの結合
・プロセッサモニタとの結合

汎用シミュレータは単一サイクルにしか対応しておらず、クロックを扱うには大幅な変更が必要になる。命令途中からの実行やレジスタ・メモリの書き換えは単一サイクルのみに対応でもシミュレータでも比較的容易に実現できるとおもわれる。現在汎用アセンブラとの結合を行っている段階である。将来的にはプロセッサモニタとも結合予定である。

５．３　考察

プログラムの途中でレジスタやメモリの値を確認することで、プログラムや命令セットの定義の善し悪しが把握できる。現在の汎用シミュレータでは、単一サイクルにのみ対応しているので、クロック数や CPI を確認することはできないが、命令実行回数や分岐回数を確認することで、分岐命令によるストール回数の多少を確認することできる。命令別の使用回数（ADD の使用回数など）を表示できれば、ストールの回数の詳細が確認できると思われる。今後 SOAR や独自で考えた命令セットをシミュレートさせて汎用シミュレータの精度を高める必要がる。

６．おわりに

ハード/ソフト・コラーニングシステムで、ソフトウェア上でプロセッサの評価をおこなう汎用シミュレータを設計した。現在 MONI に対応した命令セットには完成させた。シミュレーションの結果、実効命令回数、分岐回数、レジスタ・メモリの値の確認ができる。様々な命令セットを用いてシミュレートさせることで、汎用シミュレータの精度の向上、汎用シミュレータに足りない機能・不具合の発見が期待できる。今後の課題として、クロック実行や総クロック数の確認など、MONI プロセッサが持つ機能を汎用シミュレータにも搭載させることがあげられる。また、デバッガとして有用な、１クロック戻る機能、汎用シミュレータの操作性を向上させるための、ユーザインターフェースなどを追加することで、汎用シミュレータのパフォーマンスを更に向上させることできる。

現在の汎用シミュレータでは、アセンブリソースを読み取り命令セットを参照してシミュレートさせる仕組みになっている。汎用アセンブラと結合し、アセンブラから得た機械語をシミュレートできれば、プロセッサの仕組みや動作を理解できる。命令セットを読み取る箇所を改良し、すれば機械語に対応させることも可能であると思われる。

汎用シミュレータに単体としての改良の余地は十分にある。しかし、レジスタ値の確認など最低限の機能を搭載することができたので、今後ユーザインターフェースの作成や汎用アセンブラとの結合など外部との結合を行いハード/ソフト・コラーニングシステムの構築を進める必要がある。今後もハード/ソフト協調学習システムの研究が進み、実際に大学でこのハード/ソフト協調学習システムが採用されれば幸いに思います。

謝辞

本研究の機械を与えてくださり、指導していただきました山崎教授に深く感謝いたします。また、共同研究者である中村氏、難波氏、富樫氏、中川氏、貴重な意見を下さった皆様に心より感謝いたします。

参考文献

［1］ 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.

［2］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.

［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卒業論文、2004.

［4］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

［5］中川裕樹：命令セット定義可能な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

［6］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

［7］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.

［8］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.

［9］John L.Hennessy, David A. Patteerson 著、成田光彰訳：コンピュータの構成と設計（上）(下)、日経 BP 社、1999

［10］アンク著、C++の絵本、ムックハウス Jr、2005.

［11］西村、額田、天野：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　電子情報通信学会技術研究報告、Vol99,No532(CPSY99 106-116), pp61-68, 2000

［12］高橋、児島、上土井、吉田：マイクロコンピュータ設計教育環境 City-1FPGA コンピュータの自由な設計と製作、情報処理学会研究報告、Vol97, No.17(DA-83),pp41-48, 1997

［13］田中、久我、末吉、小羽田：教育用マイクロプロセッサ KITE とその開発支援環境、情報処理学会研究報告、Vol93, No.49(ARC-100), pp.59-66, 1993

［14］末吉、小羽田、野崎、田中、久我：FPGA を利用した教育用マイクロプロセッサ KITE-2 システムソフトウェア教育への対応情報処理学会研究報告、Vol.95, No.25(ICD9511-22), pp.71-78, 1995

付録１

MONI に対応した命令セット

```
architecture{
        widthofword     16;
        numberofregister        8;
        numberofmemory          256;
};

iset 16 MONI{
        field constant typ[2];
        field constant ope[5];
        field constant fun[2];
        field variable reg[3];
        field variable i05[5];
        field variable i08[8];
        field variable i11[11];
        format R{
                        ope op;
                        fun fn;
                        reg rs;
                        reg rt;
                        reg rd;
        };
        format I5{
                        ope op;
                        i05 im;
                        reg rt;
                        reg rd;
        };
        format I8{
                        ope op;
                        i08 im;
                        reg rd;
        };
        format J{
                        ope op;
```

```
                    i11 im;
};
inst R ADD{
                    op = 0;
                    fn = 0;
                    rs = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    ADD rd rt rs;
                    };
};
inst R SUB{
                    op = 0;
                    fn = 1;
                    rs = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    SUB rd rt rs;
                    };
};
inst R AND{
                    op = 0;
                    fn = 2;
                    rs = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    AND rd rs rt;
                    };
};
inst R OR{
                    op = 0;
                    fn = 3;
                    rs = @3;
```

```
                rt = @2;
                rd = @1;
                behavior{
                OR rd rs rt;
                };
};
inst R XOR{
                op = 1;
                fn = 0;
                rs = @3;
                rt = @2;
                rd = @1;
                behavior{
                XOR rd rs rt;
                };
};
inst R NOT{
                op = 2;
                fn = 3;
                rs = @3;
                rt = @2;
                rd = @1;
                behavior{
                NOT rd rs rt;
                };
};
inst R SLT{
                op = 1;
                fn = 1;
                rs = @3;
                rt = @2;
                rd = @1;
                behavior{
                SLT rd rt rs;
                };
};
```

```
inst R SGT{
                    op = 1;
                    fn = 2;
                    rs = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    SLT rd rs rt;
                    };
};
inst R SLE{
                    op = 1;
                    fn = 3;
                    rs = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    SLT rd rt rs;
                    SEQ temp rt rs;
                    OR rd rd temp;
                    };
};
inst R SGE{
                    op = 2;
                    fn = 0;
                    rs = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    SLT rd rs rt;
                    SEQ temp rt rs;
                    OR rd rd temp;
                    };
};
inst R SEQ{
                    op = 2;
```

```
                fn = 1;
                rs = @3;
                rt = @2;
                rd = @1;
                behavior{
                SEQ rd rt rs;
                };
};
inst R SNE{
                op = 2;
                fn = 2;
                rs = @3;
                rt = @2;
                rd = @1;
                behavior{
                SLT rd rs rt;
                SLT temp rt rs;
                XOR rd rd temp;
                };
};
inst R SLL{
                op = 3;
                fn = 0;
                rs = @3;
                rt = @2;
                rd = @1;
                behavior{
                SLL rd rt rs;
                };
};
inst R SRL{
                op = 3;
                fn = 2;
                rs = @3;
                rt = @2;
                rd = @1;
```

```
                behavior{
                SRL rd rt rs;
                };
};
inst R SRA{
                op = 3;
                fn = 3;
                rs = @3;
                rt = @2;
                rd = @1;
                behavior{
                SRA rd rt rs;
                };
};
inst I5 ADDI{
                op = 4;
                im = @3;
                rt = @2;
                rd = @1;
                behavior{
                STR temp im;
                ADD rd rt temp;
                };
};
inst I5 SUBI{
                op = 5;
                im = @3;
                rt = @2;
                rd = @1;
                behavior{
                STR temp im;
                SUB rd rt temp;
                };
};
inst I5 ANDI{
                op = 6;
```

```
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    AND rd rt temp;
                    };
};
inst I5 ORI{
                    op = 7;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    OR rd rt temp;
                    };
};
inst I5 XORI{
                    op = 8;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    XOR rd rt temp;
                    };
};
inst I5 SLLI{
                    op = 9;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    SLL rd rt temp;
```

```
                    };
};
inst I5 SRLI{
                    op = 10;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    SRL rd rt temp;
                    };
};
inst I5 SRAI{
                    op = 11;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    SRA rd rt temp;
                    };
};
inst I5 SLTI{
                    op = 12;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    SLT rd rt temp;
                    };
};
inst I5 SGTI{
                    op = 13;
                    im = @3;
                    rt = @2;
```

```
                    rd = @1;
                    behavior{
                    STR temp im;
                    SLT rd temp rt;
                    };
};
inst I5 SLEI{
                    op = 14;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    SLT rd rt temp;
                    SEQ temp rt temp;
                    OR rd rd temp;
                    };
};
inst I5 SGEI{
                    op = 15;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    SLT rd temp rt;
                    SEQ temp rt temp;
                    OR rd rd temp;
                    };
};
inst I5 SEQI{
                    op = 16;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
```

```
                    STR temp im;
                    SEQ rd rt temp;
                    };
};
inst I5 SNEI{
                    op = 17;
                    im = @3;
                    rt = @2;
                    rd = @1;
                    behavior{
                    STR temp im;
                    SLT rd rt temp;
                    SLT temp temp rt;
                    XOR rd rd temp;
                    };
};
inst I5 LD{
                    op = 20;
                    im = @0;
                    rt = @2;
                    rd = @1;
                    behavior{
                    LD rd rt;
                    };
};
inst I5 ST{
                    op = 21;
                    im = @0;
                    rt = @2;
                    rd = @1;
                    behavior{
                    ST rt rd;
                    };
};
inst I8 LDHI{
                    op = 22;
```

```
                    im = @2;
                    rd = @1;
                    behavior{
                    SEQ temp temp temp;
                    SUB SP SP temp;
                    ADD temp temp temp;
                    ADD temp temp temp;
                    ADD temp temp temp;
                    ADD temp temp temp;
                    SLL rd rd temp;
                    SRL rd rd temp;
                    STR temp im;
                    ST rd SP;
                    STR rd im;
                    ADD temp temp temp;
                    ADD temp temp temp;
                    ADD temp temp temp;
                    ADD temp temp temp;
                    SLL rd rd temp;
                    LD temp SP;
                    ADD rd rd temp;
                    SEQ temp temp temp;
                    ADD SP SP temp;
                    };
};
inst I8 LDLI{
                    op = 23;
                    im = @2;
                    rd = @1;
                    behavior{
                    SEQ temp temp temp;
                    ADD temp temp temp;
                    ADD temp temp temp;
                    ADD temp temp temp;
                    SRL rd rd temp;
                    SLL rd rd temp;
```

```
                STR temp im;
                ADD rd rd temp;
                };
};
inst I8 BEQZ{
                op = 18;
                im = @2;
                rd = @1;
                behavior{
                STR temp im;
                BEZ rd temp;
                };
};
inst I8 BNEZ{
                op = 19;
                im = @2;
                rd = @1;
                behavior{
                STR temp im;
                BNZ rd temp;
                };
};
inst I8 PUSH{
                op = 24;
                im = @2;
                rd = @1;
                behavior{
                SEQ temp temp temp;
                SUB SP SP temp;
                ST rd SP;
                };
};
inst I8 POP{
                op = 25;
                im = @2;
                rd = @1;
```

```
                behavior{
                LD rd SP;
                SEQ temp temp temp;
                ADD SP SP temp;
                };
};
inst J JUMP{
                op = 28;
                im = @1;
                behavior{
                STR temp im;
                JUMP temp;
                };
};
inst J CALL{
                op = 26;
                im = @1;
                behavior{
                SEQ temp temp temp;
                SUB SP SP temp;
                ST PC SP;
                STR PC im;
                };
};
inst J RETURN{
                op = 27;
                im = @1;
                behavior{
                LD PC SP;
                SEQ temp temp temp;
                ADD PC PC temp;
                ADD SP SP temp;
                };
};
inst J NOP{
                op = 30;
```

```
                im = @0;
                behavior{
                NOP;
                };
        };
        inst J HALT{
                op = 31;
                im = @0;
                behavior{
                HALT;
                };
        };
        dummy ADDI COPY{
                im = @0;
                rt = @1;
                rd = @2;
            substitution{
                ADDI rt rd 0;
            };
        };
        dummy SUB CLEAR{
                rs = @1;
                rt=@0;
                rd=@0;
                substitution{
                        SUB rs rs rs;
                };
        };
};
```

付録２

N から１までの和のプログラム

```
        SUB     $3 $3 $3;
        LD      $1      $3;
        SUB $4 $4 $4;
        SUB $2 $2 $2;
LOOP:   ADDI    $4      $4      1;
        SUBI    $1      $1      1;
        ADD     $2      $2      $4;
        SGTI    $5      $1      0;
        BNEZ    $5      LOOP;
        LDLI    $3      1;
        ST      $3 $2;
        HALT;
```

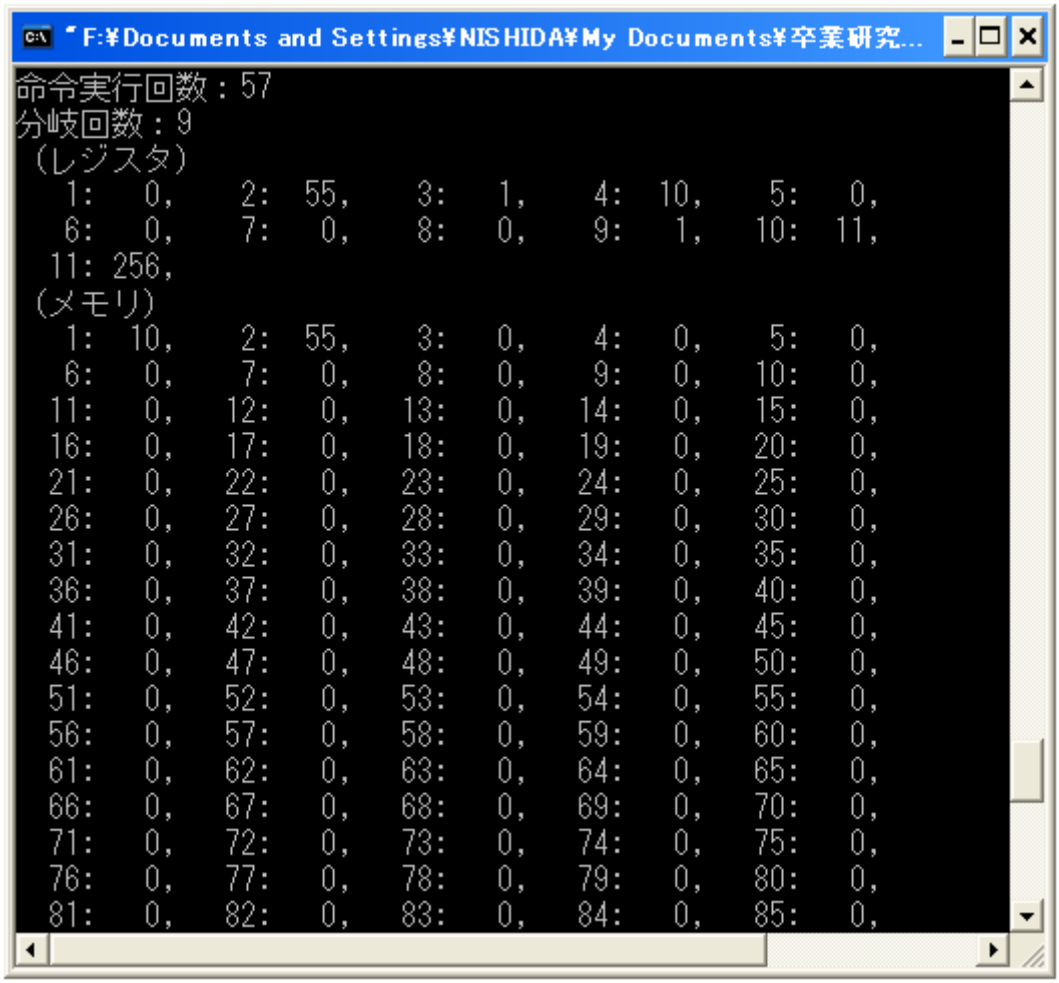

図 7：１から１０までの実行結果

付録３

最小公約数のプログラム

```
SUB     $3 $3 $3;
        LD      $1      $3;
        LDLI    $3      1;
        LD      $2      $3;
FLAG1: SUB      $1      $1      $2;
        SGEI    $5      $1      0;
        BNEZ    $5      FLAG1;
        ADD     $1      $1      $2;
        ADDI    $4      $1 0;
        ADDI    $1      $2 0;
        ADDI    $2      $4 0;
        BNEZ    $2      FLAG1;
        LDLI    $3      2;
        ST $3 $1;
        HALT;
```

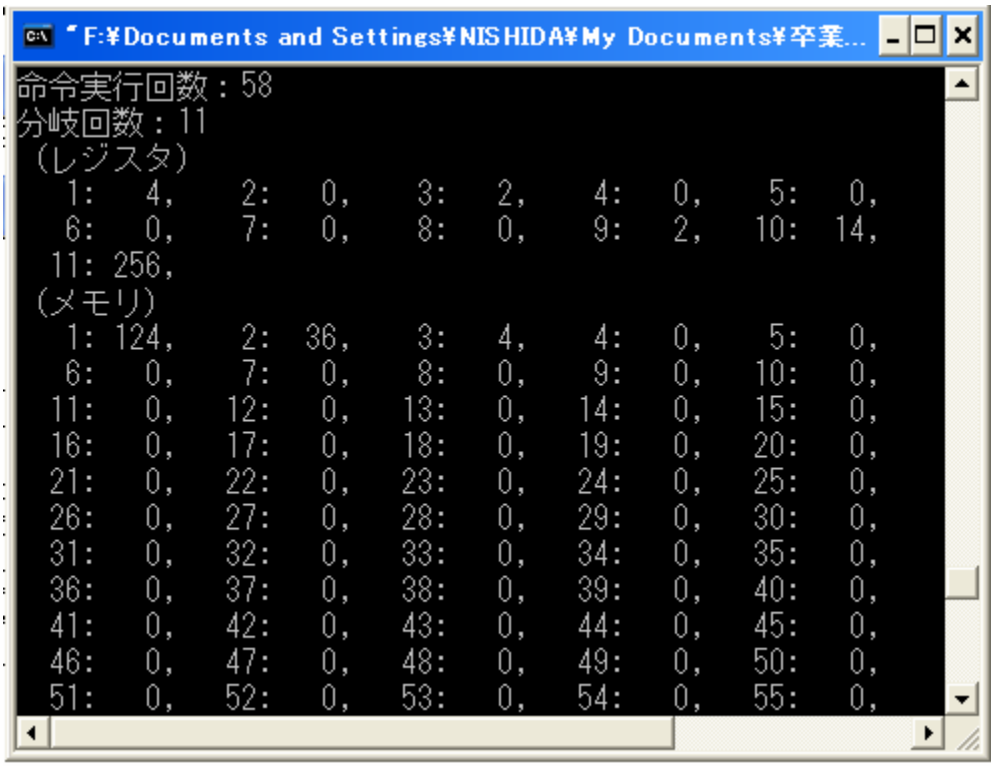

図 8：１２４と３６の最大公約数の実行結果