卒業論文

命令セット定義可能な汎用アセンブラの設計と実装

氏名　　：富樫　望
学籍番号：2210020303-4
指導教員：山崎勝弘教授
提出日　：2006 年 2 月 20 日

立命館大学理工学部情報学科

内容梗概

本研究では、プロセッサにおけるマイクロアーキテクチャ、命令セットアーキテクチャの理解と設計を目標としたハード/ソフト・コラーニングシステムへの拡張を行い、その構成要素である命令セット定義ツールと汎用アセンブラの開発を行った。

従来のハード/ソフト・コラーニングシステムは、オリジナルプロセッサ MONI に対してハードウェアとソフトウェアを協調的に学習するものであった。さらにハードウェアとソフトウェアの協調的な理解を促すシステムへと移行するために、独自の命令セットを定義できる機構を検討し、拡張を行った。拡張したコラーニングシステムでは、命令セット情報の定義を助けるツール、独自の命令セットに対しての汎用アセンブラ、汎用シミュレータやホスト PC から制御、観測ができるプロセッサデバッガなどがある。

命令セット定義ツールは、命令セット情報の定義を支援し、汎用アセンブラと汎用シミュレータを定義した命令セット用のアセンブラとシミュレータとして動作させるために用いるツールである。汎用アセンブラは、命令セット定義ツールで定義した情報に従って、アセンブリソースを対応する機械語に変換、汎用シミュレータでのシミュレーションコードの出力を行う。

本論文では、ハード/ソフト・コラーニングシステムの詳細について述べ、命令セット定義ツールと汎用アセンブラの設計、実装、評価について述べる。

目次

図目次

表目次

1　はじめに

1.1　研究背景

近年、FPGA の集積度、性能が著しく向上し、マイクロプロセッサを中心としたソフトウェアとハードウェアを搭載したシステムLSIが半導体業界の主流となっている。これは、プロセッサなどのシステム全体を 1 チップ上に載せる SoC（System on a Chip）が一般化していることを表している。また、さらに広い概念のシステムをパッケージ化する SiP（System in Package）へと進みつつある。

FPGA の集積度、性能の向上により、LSI による可能性が拡大する反面、システムの大規模化によるコストの上昇、設計の複雑化による設計期間の長期化が問題となってきている。大規模化、複雑化した SoC、SiP をこれまでより短期間かつ低コストで信頼性高く設計・製造することが要求されている[1][2]。これらの問題を解決するため、ハード/ソフト・コデザイン[3][4]やコンフィギャラブル・プロセッサ[5]などの技術が注目されている。

ハード/ソフト・コデザイン（Hardware-Software Co-design）とは、対象となる組み込みシステムに対して、ハードウェア部とソフトウェア部をシステム全体が最適(性能･コスト･消費電力など)になるように協調設計する手法である。ハード/ソフト・コデザインは、システムレベル記述言語(SpecC、SystemC など)を用いてシステム全体を記述するため、設計の最上流で多くのバグを潰すことができ、開発期間短縮への効果が大きい。しかし、ハードウェアとソフトウェアとの分割の決定は人間に委ねられる。そのため、システム設計者にはハードウェア、ソフトウェア、さらにはシステム設計に関する体系的な知識が必須である。

コンフィギャラブル・プロセッサとは、用途に応じて命令セットなどをカスタマイズできるプロセッサである。コンフィギャラブル・プロセッサを用いることにより、ソフトウェアによる実装を行いつつ、対象とするアプリケーションに最適なカスタム命令を組み込むことで、高い性能を実現することができる。また、対象となるアプリケーションを効率よく動作させる命令やプロセッサ構成を探し出す設計空間探索（design space exploration）を効率よく行うことができ、開発期間短縮へと繋がる。しかし、生成されたプロセッサの動作だけではなく、コンパイラなどのソフトウェア開発環境の動作についても検証、評価を行う必要がある。そのため、ハードウェアとソフトウェアについてそれぞれの知識が問われる。

システムレベルでの最適化が求められる現在の組み込みシステムにおいて、システムに占めるソフトウェア量は機能の向上に応じて急速に増加している。そして限られたメモリ容量の中では、アセンブリレベルでの最適化も必要となる。

このようにシステム LSI の開発において、ハードウェアとソフトウェアは切り離せない関係にあり、開発に携わる技術者はハードウェアとソフトウェアの知識習得が必要となる。

1.2　研究目的

システム LSI の高集積化に伴い、ハード・ソフト協調設計の必要性が高まってきている。ハード/ソフト・コデザイン、コンフィギャラブル・プロセッサでの設計は、必要な命令セットを載せたプロセッサアーキテクチャの HDL による構築、かつアーキテクチャに適したソフトウェアの開発を行う能力を必要とする。これらの要素を満たすために、開発者の早い段階での教育が必要不可欠となる。

我々は、大学でのハード・ソフト協調学習を目的としてオリジナルプロセッサ「MONI[10]」を用いたハード/ソフト・コラーニングシステム[8][9]の開発を進めてきた。このシステムは、可観測性を重視したプロセッサシミュレータとハードウェア設計演習とを融合させ、プロセッサアーキテクチャをテーマにハードウェアとソフトウェアをバランスよく学習できるものである。そのシステムの有効性を検証するために、我々はこの学習システムを用いて SOAR[6]プロセッサの設計を行い、FPGA 上に実装した。そこで、以下のような問題点が明らかとなった。

○ FPGA 内のプロセッサ制御・観測が困難
○ ユーザ独自の命令セットを設計できない点
○ 独自の命令セットを実装するための環境が整っていない点

この学習システムの意向を受け継ぎ、より充実した学習システムを目指して拡張した。以下にそれぞれの項目について解決案を示す。

FPGA 内のプロセッサ制御・観測は 7 セグメント LED、スイッチでしか行うことができない。この点を解消するために、FPGA ボード上にプロセッサデバッガを作成してホスト PC から制御、観測を行えるようにする。

ユーザ独自の命令セットを設計することは、MONI プロセッサを対象としているシステムであるため困難となる。よりプロセッサ理解を促すシステムを構築するためには、ユーザ独自の命令セットを設計できるシステムへと移行する必要がある。命令セットの定義を手助けする命令セット定義ツール(Instruction Set Information Define: 以下 ISID と略す)を作成する。命令セットの作成が円滑に行え、命令セットの理解を深めることができる。

独自の命令セットを実装する際の環境が整っておらず、ハードウェア上でのデバッグや定義した命令セットに対するアセンブルが困難である。この点を解消するために、プロセッサデバッガや定義した命令セットに対するアセンブラを作成する。プロセッサデバッガにより、ハードウェア上の動作をホスト PC 上で観測・制御することができ、デバッグを円滑に行える。汎用アセンブラにより、独自の命令セットに対するアセンブラを作成する必要がなく、ハードウェアとソフトウェアの協調的な理解に時間を多く使うことができる。

以上の点より本システムでは、命令セット設計を可能とし、マイクロプロセッサの設計の自由度をあげると共に学習環境の整備を行う。そして、マイクロアーキテクチャ、命令セットアーキテクチャの理解と設計を通してハードウェアとソフトウェアとを協調的に学習してもらうことが本研究の目的である。

1.3　関連研究

関連研究として、教育用プロセッサを用いたシステム、プロセッサシミュレータ、ハード/ソフト協調学習システムがある。他大学におけるこれらの学習システムを以下に記す。

（1）教育用プロセッサを用いたシステム

- PICO^2 [23]

慶応義塾大学が開発した教育用パイプラインプロセッサ PICO^2 を用いたシステム。4 ビットから 16 ビット命令セットのフレームワークが用意されており、学習者は簡単な命令を持つパイプラインプロセッサを改良し、命令・フォワーディング・ストールの機能を追加しながら、パイプラインプロセッサの完成を目指す。

- City-1[24]

広島市立大学が開発した命令セットアーキテクチャや、アドレス空間を自由に設計させるシステム。合成可能だが不完全な記述を学生に提供し、プロセッサを設計させる。

- KITE [25][26]

九州工業大学が開発した教育用プロセッサ KITE を用いた学習システム。16 ビット長で教育用として最低限の命令を持ち、アキュムレータ方式で動作する KITE1 と、割り込み処理などを追加した KITE2 の 2 種類がある。KITE システムの特徴は、観測性のあるボードコンピュータとホームページを利用した e-Learning の充実である。

（2）プロセッサシミュレータ

- Mikage[27]

広島市立大学が開発した教育用スーパスカラプロセッサ・シミュレータ。最適なスーパスカラプロセッサを設計するためのハード/ソフト評価ツール。パイプラインの使用状況を可視化し、スーパスカラの並列度が可変である。

- DLX-View[28]

観測性と対話性を兼ね備えたパイプラインプロセッサシミュレータ。プロセッサの動作を正しく理解するのが目的で、命令セットの理解・デバッグ・プロセッサの性能評価にも使用できる。

- VisuSim[29]

香川大学が開発した計算機の内部構造や動作原理を視覚的に理解させることが目的のシミュレータ。Web 上からダウンロードして使用することができる。

（3）ハード/ソフト協調学習システム

- 港[30]

拓殖大学が開発したシステムソフトウェア教育支援環境。プロセッサ・OS・コンパイラの相互関係を意識しながら改良が行えるという実装的観点からのアプローチを取ったシステムプログラミング学習環境である。つまり、プロセッサ・OS・コンパイラの実装知識をバランスよく身に付けることができるシステムである。

- SEP4[31]

静岡大学が開発した 16 ビット 3 段パイプラインプロセッサ SEP4 を用いたハード/ソフト協調学習システムである。ハードウェア設計では、ASIP Meister を用いてパイプラインプロセッサ設計を行う。ソフトウェア演習ではハザードを回避するための命令スケジューリングなどを行う。

1.4 研究概要

本研究では、高性能計算研究室のハード/ソフト・コラーニングシステムを学習の習熟度という観点から拡張した。プロセッサアーキテクチャを体系的に学習でき、更なる探求が行える教育システムの設計を行った。

ハード/ソフト・コラーニングシステムでは、プロセッサアーキテクチャの理解を目標としたソフトウェア学習と、プロセッサ設計を行うことでコンピュータアーキテクチャの理解を深めるハードウェア学習によって構成される。ハードウェアとソフトウェアの協調的な理解を目的としている。

ソフトウェア学習では、命令セットとプロセッサアーキテクチャの関係理解を目的としている。以下の 3 点が主な目的達成のための要素である。

○ オリジナルプロセッサ MONI[10]を対象としたプロセッサシミュレータを用いて、プロセッサの内部動作の理解を図る
○ 与えられた課題に対して、解を導き出せる命令セットを命令セット定義支援ツールで形にし、解に対する最適な命令セットの探求を図る
○ 汎用アセンブラ、シミュレータを用いて、プロセッサの動作の理解を図る

ハードウェア学習では、ソフトウェア学習によるアーキテクチャ理解の確認と実践を目標としている。以下の 3 点が主な目的達成のための要素である

○ プロセッサ設計による、プロセッサアーキテクチャのさらなる理解を図る。
○ HDL を用いたハードウェア設計による設計手順の理解を図る
○ プロセッサデバッガ、プロセッサモニタ、インタフェースモジュールを用いて、FPGA ボードコンピュータでのデバッグにより、コンピュータアーキテクチャの理解を図る

FPGA ボードコンピュータとは、ハード/ソフト・コラーニングシステムでの実機検証に用いるプロセッサ検証システムである。設計では、メモリ以外は IP を使用せず Verilog-HDL による RTL 設計を行う。Celoxica 社の RC200 ボードをターゲットとしている。

以上の点を踏まえて、アーキテクチャを自由に変更しながら、アーキテクチャに適したプログラミングや、プログラムにあった命令セットの探索が可能になる。

本論文では、第 2 章で拡張ハード/ソフト・コラーニングシステムについて詳しく説明する。第 3 章で ISID の設計と実装について述べ、第 4 章では汎用アセンブラの設計と実装について述べる。第 5 章で ISID と汎用アセンブラのテストと評価を行う。そして最後に総評を行う。

2　ハード/ソフト・コラーニングシステムの拡張

2.1　従来のハード/ソフト・コラーニングシステム

我々は、コンピュータアーキテクチャにおけるマイクロプロセッサ(ハードウェア)とプログラム(ソフトウェア)の関係を学習するハード/ソフト･コラーニングシステムの構築を目的として研究を進めてきた。図 1 に MONI オリジナルプロセッサを基盤としたハード/ソフト・コラーニングシステムの学習フローを示す。

このシステムの特徴は、プロセッサアーキテクチャが選択可能な教育用シミュレータを用いて、ソフトウェアプログラミングとプロセッサアーキテクチャを体系的に学習できることである。またプロセッサを設計し FPGA を用いて実装することでコンピュータアーキテクチャに対する理解も深める。

コラーニングシステムにおける学習は、ソフトウェア学習とハードウェア学習から構成される。ソフトウェア学習では命令セットの評価とプロセッサアーキテクチャの理解を目標として、教育用シミュレータを使用してアセンブリ言語によるプログラミングやデバッグ、プロセッサアーキテクチャの変更によるプログラムの比較や評価、及び最適化コンパイラ設計などを学習する。ハードウェア学習では実際に MPU コア部を設計し、FPGA に実装することで、命令セットとプロセッサアーキテクチャに対する理解をより確かなものとする。

オリジナルプロセッサ MONI[10]は、教育システムにおけるアセンブリプログラミングのしやすさを考慮して定義した MIPS のサブセットで、命令長 16 ビット、全 43 命令からなる。また MONI は 4 つの命令形式(R、I5、I8、J)を持ち、加減算・論理演算・シフト・セット命令に即値アドレシングとレジスタアドレシングの両方を備え、メモリアクセスはレジスタ間接アドレシングを採用する。レジスタ間演算は 3 オペランド方式を採る。MONI では、①単一サイクル(1 命令を 1 クロックで実行)、②マルチサイクル(1 命令を 3～5 クロックで実行)、③パイプライン(5 ステージ)、④スーパースカラ(In-order の 3 命令同時発行)の４種類の異なる方式で、FPGA ボードコンピュータ上に実装されている。

これらの機能を用いて、ハードウェアとソフトウェアを相互理解できる学習システムとなっている。

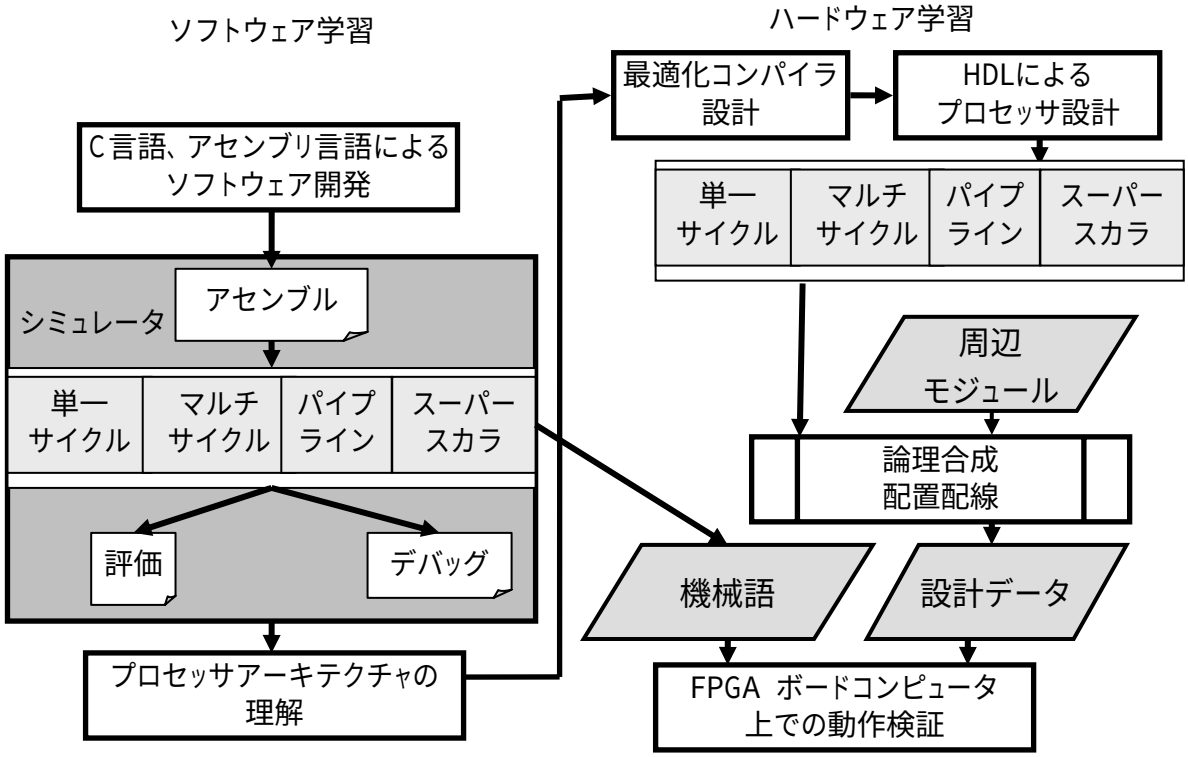

図 1：ハード/ソフト・コラーニングシステムの学習フロー

2.2　拡張点とシステム概要

前節で述べたハード/ソフト・コラーニングシステムでは以下の問題点が挙げられる。

○　FPGA 内のプロセッサ制御・観測が困難

　－FPGA 内のプロセッサ制御・観測は LED、スイッチでしか行えない

○　ユーザ独自の命令セットを設計できない

　－MONI プロセッサを対象としているため、新たな命令セットの設計は困難

○　プロセッサ作成のための環境が整っていない

　－ハードウェア上でのデバッグや定義した命令セットに対するアセンブルが困難

以上の問題点を解消するために、以下の点で拡張した。

○　マイクロアーキテクチャ、命令セットアーキテクチャの理解

　⇒命令セットを独自に定義できる

○　学習環境の充実

　⇒命令セットの定義を助けるソフトウェアがある

　⇒定義した命令セットに対してアセンブル、シミュレーションできる

　⇒ホスト PC から制御、観測できる

ソフトウェア学習では ISID、汎用アセンブラ、汎用シミュレータを使用する。これらは

命令セットアーキテクチャの設計を手助けするツールである。

ハードウェア学習では、プロセッサモニタ、インタフェースモジュール、プロセッサデバッガを提供する。これらは、FPGA への実装を手助けするツールである。これらの各モジュールの詳細については、次節で述べる。

次に、システム概要について言及する。本システムは、命令セットを独自に定義することができるハード・ソフト協調学習システムである。命令セット定義可能な学習システムにより、マイクロアーキテクチャ、命令セットアーキテクチャの理解を促す。

ソフトウェア面では、独自で定義した命令セットのアセンブリ言語に対して、プロセッサを意識したプログラミングを行う。また、シミュレータを用いてプロセッサアーキテクチャの動作について理解する。これらを通して、より最適な命令セットの探求、プロセッサを意識したプログラミングが行えるようにすることがソフトウェア面での目的である。

ハードウェア面では、独自で定義した命令セットが正常動作するようなプロセッサをHDLにより設計する。そして、設計したプロセッサをFPGAボードコンピュータに実装し、周辺回路、及び開発したソフトウェアと共に実機検証を行う。これらを通して、ハードウェア設計フローの理解、プロセッサの体系的な理解がハードウェア面での目的である。

システム全体では、ハードウェアとソフトウェアの協調的な理解、より最適な命令セットの探求を目的としている。図２に拡張コラーニングシステムの学習フローを示す。

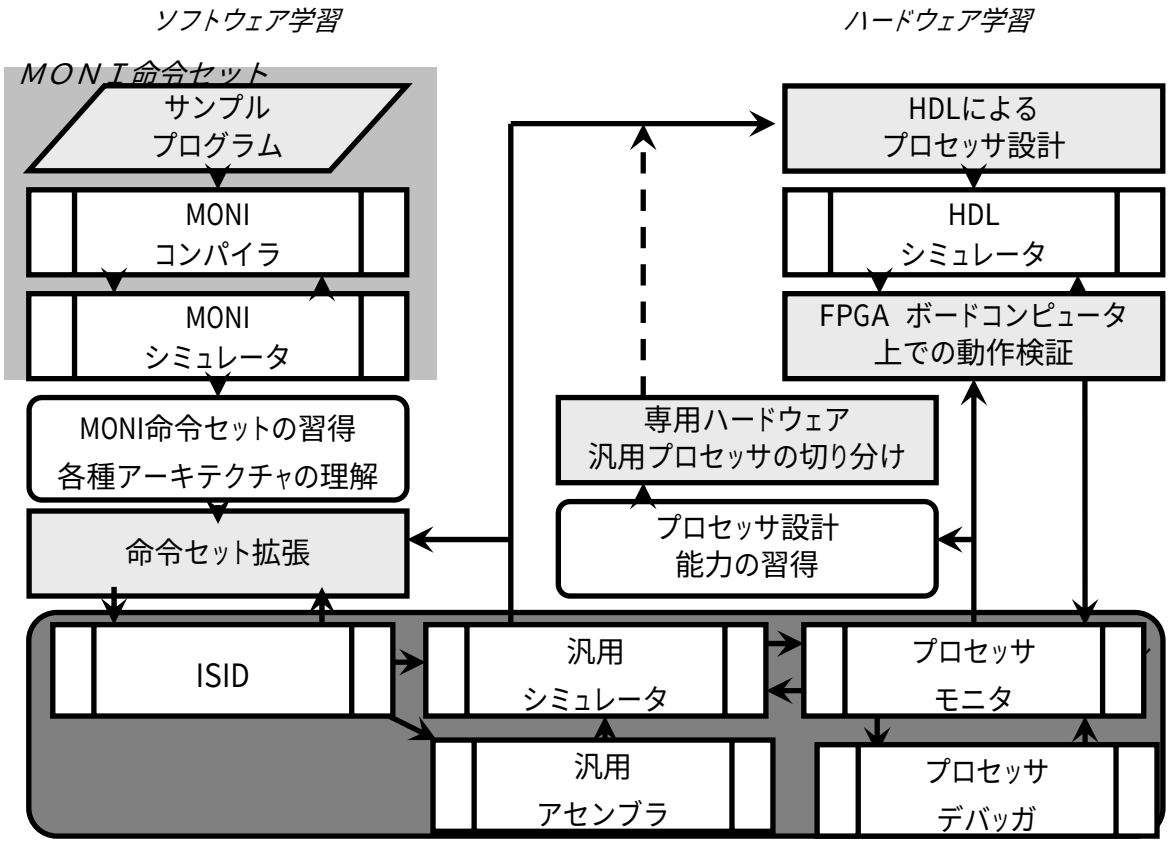

図 2：拡張ハード/ソフト・コラーニングシステムの学習フロー

拡張ハード/ソフト・コラーニングシステムでは、ソフトウェア学習とハードウェア学習に分けられる。まずソフトウェア学習を行う。

① MONI 命令セットで記述されたサンプルアセンブリコードを用いて、MONI コンパイラとシミュレータを動作させる。これにより、MONI 命令セットの理解、シミュレータによるアーキテクチャとプロセッサの動作の理解を促す。

② 新たな問題を提示し、その問題を解くための命令セットを考えてもらう。

③ ISID により、命令セットとアーキテクチャに関する情報の入力を行う。ISID とは、汎用アセンブラ、汎用シミュレータを動作させるために命令セットの定義を補助するツールである。

④ 命令セットを定義し終えたら、汎用アセンブラで独自の命令セットのアセンブリコードをアセンブルする。汎用アセンブラとは、学習者が定義した独自の命令セットに対して、アセンブル、機械語生成、シミュレーションコード生成を行うツールである。

⑤ 正しくアセンブルが行うことができれば、汎用シミュレータを用いてシミュレーションを行う。汎用シミュレータとは、独自の命令セットのアセンブリコードに対して結果を出力するソフトウェアである。もし、与えられた問題を解くことができなければ①に戻り、繰り返し学習を行う。

ソフトウェア学習でプロセッサの動作が理解した上で、ハードウェア学習を行う。

⑥ 与えられた問題を解くことができる命令セットであれば、実際にプロセッサを HDL で作成する。

⑦ HDL で作成したコードを HDL シミュレータ(Meutor Grphics 社の Modelsim など)でシミュレーションし、正常に動作しているかを確認する。

⑧ プロセッサモニタとプロセッサデバッガを用いて、FPGA 上でハードウェアをデバッグする。プロセッサモニタとは、ホスト PC から FPGA 上の動作観測や変更を行うツールである。プロセッサデバッガとは、CPU コア内のレジスタやメモリの値の変更やブレイクポイントの設定などの機能を用いてデバッグを補助するツールである。汎用シミュレータの結果と比べ、もし正しく FPGA 上で動作していなければ、プロセッサデバッガを用いてデバッグを行い、⑥に戻り、繰り返し実装を行う。

ソフトウェア学習では、MONI プロセッサのシミュレータを動作させることにより、プロセッサの内部動作を理解することができる。また、独自の命令セットの作成を援助する。そして、汎用アセンブラや汎用シミュレータによって、より最適な命令セットの探求を行う。これにより、プロセッサの動作の視覚的な理解を促すことができると考える。

ハードウェア学習では、HDL を用いて MPU の設計を行い、FPGA へ設計データをダウンロードして実機検証することにより、プロセッサアーキテクチャの体系的な理解を促す。また、コンフィギャラブル・プロセッサ合成技術に必要な設計空間探索能力の習得を促す。

以上のプロセスにより、ハードウェアとソフトウェアを協調的に学習する。

2.3　システム構成要素

（1）MONI プロセッサ

MIPS[14]のサブセットとして定義した 16 ビット教育用マイクロプロセッサである。教育用を意識し、アセンブリプログラミングのし易さを考慮した命令セットを持つ。単一サイクル、マルチサイクル、パイプライン、スーパスカラの 4 種類のアーキテクチャがある。歴史的な理解を深めようという意図から、初期段階のマイクロプロセッサの単一サイクル方式から、マルチサイクル、パイプライン方式を通して、現在のマイクロプロセッサの主流であるスーパースカラへと段階的に学習する。学習者は各プロセッサのコアとなる部分を HDL により設計し、ハードウェア設計の学習を行う[8]。

（2）MONI 命令セットシミュレータ

MONI プロセッサを対象とした命令セットシミュレータの特徴を以下に示す。

○　4 種類のプロセッサアーキテクチャを選択可能
○　プロセッサで命令が実行されている様子をデータパスの強調表示で可視化
○　レジスタやメモリの内容を表示
○　複数の実行モード
○　ハザード通知
○　プロセッサアーキテクチャやプログラムの評価

上記の特徴により、HDL によるプロセッサ設計を行う前にプロセッサアーキテクチャを理解するためだけではなく、ソフトウェア開発とデバッグにも利用できる[9]。

（3）ISID（命令セット情報定義ツール）

アーキテクチャ情報と命令セット情報の定義を GUI により援助するツールである。ISID による定義事項について以下に示す。

○　アーキテクチャ情報
　・　汎用レジスタ数
　・　汎用レジスタのビット幅
　・　メモリのワード数
○　命令セット情報
　・　命令セット名
　・　命令長
　・　フィールド情報
　・　フォーマット情報
　・　命令情報

- 命令の動作
- 擬似命令情報
- 機械語

上記の点を順に定義することで、命令セットを理解することができる。また、汎用アセンブラとシミュレータを動作させるための情報を作成できる。 ISID では、GUI ベースの編集により、入力ミスの軽減や流れに沿った定義が行える[11]。また、以前に定義した命令セットを再編集するために ISID プログラムとしてファイル出力を行う。プロセッサ設計に必要な情報の定義により、汎用アセンブラ、汎用シミュレータを動作させることによって、より最適な命令セットの探求を補助するツールとなっている。ISID の詳細については第 3 章で述べる。

（4）汎用アセンブラ

ISID で定義した命令セットと各命令に対応したアセンブリコードに対してアセンブル、機械語出力を行うツールである。以下にアセンブリソースの基本構文について記す。

○ 命令語長は 8・16・32・64bit から選択（命令セット内で固定）
○ 命令形式は自由に設定可能（命令セット内で固定）
○ op ラベル（命令）は、アルファベットの大文字のみ
○ reg ラベルは‘ $ ’＋数字
○ mem ラベルは‘ ＊’＋数字（メモリのアドレス）
○ ラベル（Jump 先のアドレス）は、アルファベットの小文字のみ＋‘ ： ’
○ 定数は 10 進定数のみ
○ コメントは、“ // ” ＋コメント文

上記のように統一することにより、定義した命令セットに対してアセンブルを行うことができる。汎用アセンブラの詳細については第 4 章で述べる。

（5）汎用シミュレータ

ISID で定義した命令セットとそれに対応したアセンブリコードに対して、単一サイクルを対象としてシミュレーションを行うツールである。シミュレーションにより、レジスタやメモリの変化や各命令によって使用されたレジスタやメモリを確認することができる。専用レジスタでは、プログラム・カウンタ、スタック・ポインタが用意されている。また、シミュレーションを行うために、21 種類の命令が用意されており、組み合わせることで様々な命令を動作させることができる。表 1 に 21 種類の命令を示す[12]。

表 1：汎用シミュレータに用意されている 21 種類の命令

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 $r2 $r0 $r1；	r0 と r1 の論理積を r2 に書き込む
OR	レジスタの論理和	OR $r2 $r0 $r1；	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 $r2 $r0 $r1；	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 $r1 $r0；	r0 のビット反転を r1 に書き込む
ADD	レジスタの加算	ADD $r2 $r0 $r1；	r0 と r1 の加算を r2 に書き込む
SUB	レジスタの減算	SUB $r2 $r0 $r1；	r0 と r1 の減算を r2 に書き込む
MUL	レジスタの乗算	MUL $r2 $r0 $r1；	r0 と r1 の乗算を r2 に書き込む
DIV	レジスタの除算	DIV $r2 $r0 $r1；	r0 と r1 の除算を r2 に書き込む
SLL	左論理シフト	SLL $r1 $r0 i 3；	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 i 2;	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 i 3；	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　$r0 $r1；	メモリ番地 r1 の値を r0 に書き込む
ＳＴ	レジスタからメモリへ書き込む	ST　$r0 $r1；	r0 の値をメモリ番地 r1 に書き込む
STR	即値をレジスタへ書き込む	STR　$r0 i 5；	r0 に５（即値）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 を代入、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

（6）**プロセッサモニタ**

ホスト PC からユーザコマンドを用いてプロセッサデバッガを制御して FPGA 上の動作確認や変更を行えるツールである。FPGA 上と同じ容量の仮想的なメモリ（バッファ）をホスト PC 上に確保し、操作するデータメモリやレジスタファイルの内容を保持している。表 2 にユーザコマンドを示す。

表 2：ユーザコマンド一覧

コマンド	動作内容
run	最後まで実行
run next	次のブレイクポイントまで実行
run clk	次のクロックまで実行
run inst	次の命令まで実行
set bp [アドレス]	ブレイクポイントの設定
del bp	ブレイクポイントの削除
read	全レジスタ値をバッファに読み込み
read dm [アドレス]	dm バッファにデータメモリの読み込み
read im [アドレス]	im バッファに命令メモリの読み込み
read pc	pc バッファに PC の読み込み
send	全バッファを書き込み
send dm	dm バッファからデータメモリに書き込み
send im	im バッファから命令メモリに書き込み
send pc	pc バッファから PC に書き込み
load dm [アドレス] [データ]	dm バッファに読み込み
load dm [ファイル]	dm バッファに読み込み
load im [ファイル]	im バッファに読み込み
init	バッファの初期化
halt	プロセッサの停止
exit	プロセッサモニタの終了

（7）**プロセッサデバッガ**

プロセッサモニタにより、学習者から入力されたユーザコマンドに従って、プロセッサ内部のレジスタ値の観測や変更、メモリ内のデータや命令の変更、またプログラムのステップ実行、ブレイクポイント実行などを実現するハードウェアモジュールである。クロックジェネレータを介してプロセッサに供給されるクロックを制御することで、1 クロック/1 命令/通常実行/ブレーク実行など様々な実行モードでプロセッサの検証ができる[13]。

（8）**インタフェースモジュール**

プロセッサモニタと FPGA をパラレルポートにより接続するためのドライバである。また、RC200 ボードに搭載されている Smart Media の制御も行う[13]。

2.4　担当範囲と開発環境

本研究では、ISID と汎用アセンブラの処理部分の設計と実装を行った。ユーザインターフェースに関しては、同研究室の中川氏が担当した。図 3 に担当範囲を示す。

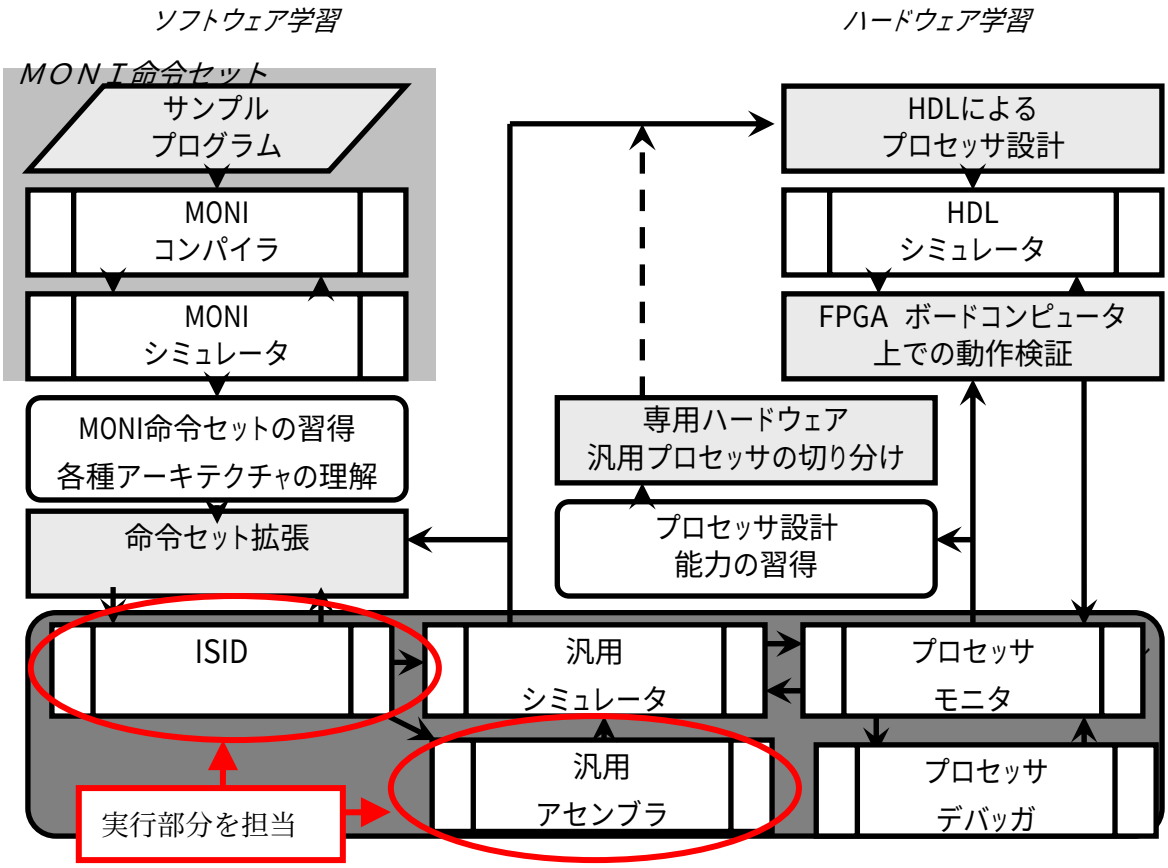

図 3 ：ハード/ソフト・コラーニングシステムにおいての担当範囲

開発環境は Visual Studio 6.0 である。また開発言語は Visual C++を用いた。これらを選択した理由は、ユーザインターフェースの高い成熟度の実現に重点を置いたからである。

3　命令セット定義ツール（ISID）の設計と実装

ISID とは、ユーザオリジナルの命令セットの定義をサポートするツールである。ISID を用いて命令セットを定義することにより、汎用アセンブラと汎用シミュレータを動作させることができ、ハード/ソフト・コラーニングシステムのソフトウェア学習の充実の要素である。

GUI ベースで編集を行うため、命令セットの下位要素から流れに沿って命令セットの定義が行えて入力ミスなどを軽減することができる。また、学習者が命令セットの定義の仕方を考える必要がなく、より最適な命令セットの探求に専念することができる。

定義した命令セットを ISID プログラムとして、ファイル出力することにより定義した内容を保存することができる。また、ISID プログラムの解析を行うことで、以前に定義した命令セットを再編集することが可能である。MONI や SOAR などの命令セットのテンプレートを用意することにより、命令セットの拡張を容易に行える。

汎用アセンブラと汎用シミュレータの動作に必要なアーキテクチャ情報と命令セット情報の定義をサポートする ISID の設計と実装について述べる。

3.1　外部仕様

ISID の外部仕様は、大きくわけて 3 つある。以下にその 3 点を示す。

○　命令セット情報の定義
○　アーキテクチャ情報の定義
○　ISID プログラム

この 3 点について仕様を固めることにより、命令セットの定義をサポートする。これらを順に説明していく。

（1） 命令セット情報で定義する項目

命令セット情報で定義する項目については、命令セットを構成する要素である。以下に定義する項目を述べる。

①　基本情報
②　フィールド情報
③　フォーマット情報
④　命令情報
⑤　擬似命令情報

上記の項目を順番に定義することにより、命令セットの設計を行う。これらの項目について詳しく説明していく。

① 命令セット基本情報

命令セット基本情報は、命令セット名や命令長で構成される命令セットの基本的な情報である。表 3 で命令セット基本情報の定義する事項について示す。

表 3：命令セット基本情報

命令セット基本情報	説明
命令セット名	命令セットに対する名前
命令長	1 命令語長.8，16，32，64bit を対象.

命令長は、2 の乗数で実現可能な範囲を対象としており、命令セット内で固定である。

② フィールド情報

フィールド情報は、フィールド名・種類・ビット幅で構成されるオペレータやオペランドに関する情報である。表 4 でフィールド情報の定義する事項について示す。

表 4：フィールド情報

フィールド情報	説明
フィールド名	フィールドに対する名前.
種類	定数と変数のどちらか 定数(constant)：命令や機能毎に固定される数値 変数(variable)：レジスタやメモリを参照するための数値
ビット幅	フィールドに対するビット幅。命令長を超えない.

種類により、オペレータ（定数）とオペランド（変数）を区別する。また、フィールドのビット幅で機械語に翻訳するビット幅を決定する。

③ フォーマット情報

フォーマット情報は、フォーマット名・フィールド群・フィールド要素群で構成される命令形式を決定する情報である。表 5 でフォーマット情報の定義する事項について示す。

表 5：フォーマット情報

フォーマット情報	説明
フォーマット名	フォーマット（命令形式）に対する名前
フィールド群	命令形式を形成するフィールドの集合
フィールド要素群	各フィールドを判別するための識別子の集合

複数個のフィールドにより、命令形式を形成する。

④　命令情報

命令情報は、命令名・フォーマット・フィールド値・動作記述で構成される命令を決定する情報である。表 6 で命令情報の定義する事項について示す。

表 6：命令情報

命令情報	説明
命令名	命令に対する名前
フォーマット	命令の構成（命令形式）
フィールド値	各フィールドに対する値 定数：命令識別用の定数 変数：'@'+定数、で表される引数
動作記述	汎用シミュレータ上での動作内容を記述

フィールド値の定数により、機械語においてそれぞれの命令を識別する。また、動作記述により汎用シミュレータの動作内容を決定する。実現可能な命令は、表 1 に示されている 21 種類の命令の組み合わせとなる。

⑤　擬似命令情報

擬似命令は、擬似命令名、命令、置換命令記述で構成される命令セットで定義され、1 命令を擬似的に新たな動作を実現するための情報である。表 7 で擬似命令情報の定義する事項について示す。

表 7：擬似命令情報

擬似命令情報	**説明**
擬似命令名	擬似命令に対する名前
命令	命令を用いて新たな命令を作成
置換命令記述	新たな命令を作成する際に引数にどのような値を代入するかを決定するための記述

置換命令記述は、命令セットで定義された 1 命令を用い、オペランドにフィールド要素を記述して、新たな命令を実現するための記述である。

（2） アーキテクチャ情報で定義する項目

アーキテクチャ情報の定義する項目については、汎用シミュレータを動作させるために必要な情報を抽出する。表 8 でアーキテクチャ情報の定義する事項について示す。

表 8：アーキテクチャ情報の定義する項目

アーキテクチャ情報	説明
ワード幅	汎用レジスタとメモリのワード幅
汎用レジスタ数	汎用レジスタの個数
メモリ容量	メモリのワード数

多くのプロセッサでは、ワード幅と命令長が等しく、またメモリ容量と 2 の命令長の乗数とが等しい。しかし、これらが異なるプロセッサも存在するため、汎用性の範囲を広げるために設定できるようにする。

（3） 再編集用の ISID プログラムの構成

ISID では、作成中の命令セットに対して、再定義できるように ISID プログラムをファイル出力する。図 4 に ISID プログラムの例について示す。

```
architecture{
        widthofword         16;
        numberofregister     8;
        numberofmemory      256;
};
iset 16 SOAR{
        field constant typ[2];
                …
        format R{
            ope op;
                …
        };
                …
        inst R ADD{
            op = 0;
                …
            behavior{
                ADD rd rt rs;
            };
        };
            …
        dummy ADDI COPY{
            im = @0;
            …
            substitution{
                ADDI rt rd 0;
            };
        };
            …
};
```

図 4：ISID プログラムの例

図 4 を基にして、ISID プログラムの仕様について説明していく。

ISID プログラムは、命令セット情報（表 3～表 7）とアーキテクチャ情報（表 8）を解析が行えるように作成されたコードである。

architecture{……}; で、アーキテクチャ情報に関する情報の定義を行う。ワード幅、汎用レジスタ数、メモリ容量を定義する。

widhtofword 10 進定数; により、汎用レジスタとメモリのワード幅の定義を行う。

numberofregister 10 進定数; により、汎用レジスタ数の定義を行う。

numberofmemory 10 進定数; により、メモリのワード数の定義を行う。

iset 16 SOAR{……}; で、命令セット情報の基本情報、フィールド情報、フォーマット情報、命令情報、擬似命令情報を定義する。

field 種類 フィールド名[10 進定数]; により、フィールドに関する情報の定義を行う。

format フォーマット名{……}; により、フォーマットに関する情報の定義を行う。

inst フォーマット 命令名{……}; により、命令に関する情報を定義する。

behavior{……}; により、動作記述の定義を行う。

dummy 命令 擬似命令名{……} により、擬似命令に関する情報の定義を行う。

substitution{……}; により、置換命令の定義を行う。

このプログラムを解析することにより、命令セットの再定義を行うことができる。
これらの外部仕様を実装するためのプロセスを次節で言及する。

3.2 内部仕様

ISID の内部構成は、UI をトップとして、大きく分けて 2 つになる。命令セット情報の保持と受け渡しを行うクラス群と ISID プログラムの解析を行うクラス群に分けられる。

命令セット情報の保持と受け渡しを行うクラス群では、命令セットに関する情報（命令セット名・命令長・フィールド情報・フォーマット情報・命令情報・擬似命令情報）を格納して、これらの情報を受け取ることができる。このクラス群は、命令セット情報クラス・擬似命令クラス・命令クラス・フォーマットクラス・フィールドクラスで構成され、命令セットの定義を行う際に情報を保持する。汎用アセンブラ、汎用シミュレータの動作時に命令セットの情報を渡し、動作を助ける。また、ISID プログラムを作成する。

ISID プログラムの解析を行うクラス群では、字句解析、構文解析、意味解析を順に行う。このクラス群は、字句解析クラス、構文解析クラス、意味解析クラス、エラー処理クラスで構成されている。ファイルに保存された ISID プログラムを解析して、再編集を行えるようにする。

図 5 に ISID の UML によるクラス図を示す。

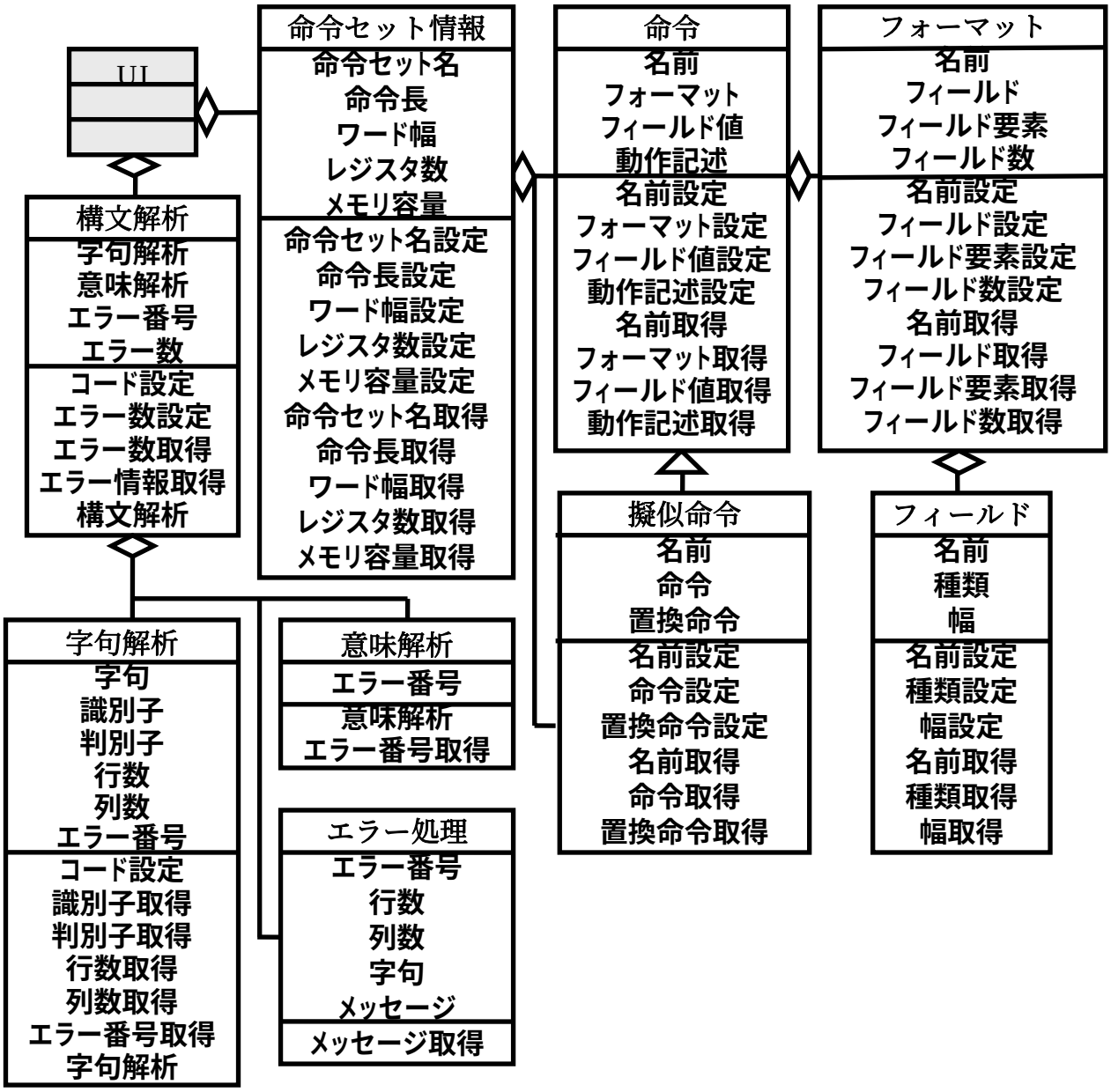

図 5：ISID のクラス図

ISID は、命令セット情報の保持と受け渡しを行うクラス群と ISID プログラムの解析を行うクラス群に分けられる。以下よりこの２つのクラス群について説明していく。

3.3　命令セット情報の保持と受け渡しを行うクラス群

命令セット情報の保持と受け渡しを行うクラス群は、命令セット情報クラス、擬似命令クラス・命令クラス・フォーマットクラス・フィールドクラスで構成されている。このクラス群では、オブジェクト間の繋がりが重要となるため UML のオブジェクト図の作成を行った。図 6 に命令セット情報の保持と受け渡しを行うクラス群のオブジェクト図を示す。

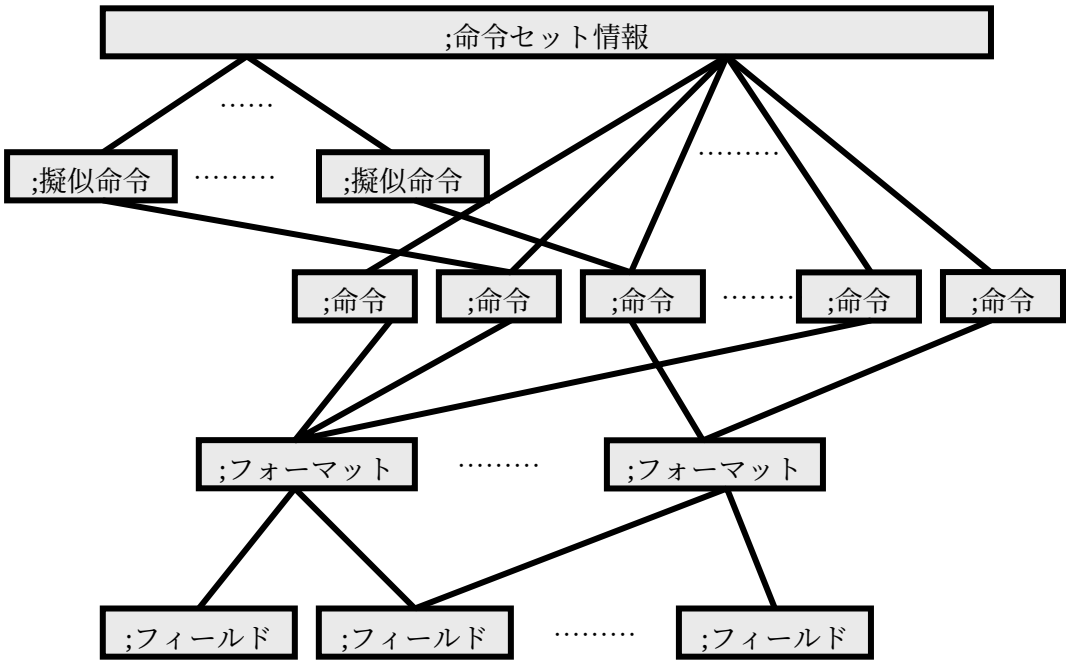

図 6：命令セット情報の保持と受け渡しを行うクラス群のオブジェクト図

図 6 について以下よりクラス毎に説明していく。

（1）命令セット情報クラス

命令セット情報クラスは、命令セットに関する情報を統括するクラスである。また、命令セット情報クラスのインスタンスは、擬似命令クラスのインスタンスと命令クラスのインスタンスを複数持っている。これにより命令セットに関する情報を統括させることでき、有用な情報の抽出が可能となる。そして、以下のような働きをする。

- GUI ベースでの編集時に命令セット定義の補助
- 定義した命令セットの ISID プログラム作成
- 解析の際にエラー発見のために必要な情報の提供

上記の動作を実現するクラスの内部構成について言及する。

命令セット情報クラスのプロパティについては、表 3 と表 8 に定義した事項を実装した。表 9 に命令セット情報クラスのメソッドについて示す。

表 9：命令セット情報クラスのメソッド

メソッド	メソッド名	機能
命令セット名設定	setISIname(char []);	命令セットの名前を設定
命令長設定	setISIlength(int);	命令長を設定
ワード幅設定	setwidth(int);	ワード幅を設定
レジスタ数設定	setregister(int);	汎用レジスタの数を設定
メモリ容量設定	setmemory(int);	メモリのワード数の設定
命令セット名取得	getISIname();	命令セット名を取得
命令長取得	getISIlength();	命令長を取得
ワード幅取得	getwidth();	ワード幅を取得
レジスタ数取得	getregister();	汎用レジスタ数を取得
メモリ容量取得	getmemory();	メモリのワード数を取得

（2）擬似命令クラス

擬似命令クラスは、擬似命令に関する情報を統括するクラスである。また、命令クラスを継承しており、命令クラスと命令と同様の処理を行うことができる。擬似命令クラスのインスタンスは、命令クラスのインスタンスを単一で持っている。これにより、プロセッサ上で仮想的に動作する命令を表現することができ、同じように扱うことができる。

擬似命令クラスのプロパティについては、表 7 に定義した事項を実装した。表 10 に擬似命令クラスのメソッドについて示す。

表 10：擬似命令クラスのメソッド

メソッド	メソッド名	機能
名前設定	setname(char[]);	擬似命令名を設定
命令設定	setDuinst(Instruction);	命令インスタンスを設定
置換命令設定	setsubst(int,int);	置換命令を設定
名前取得	getname();	名前を取得
命令取得	getDuinst();	命令インスタンスを取得
置換命令取得	getsubst();	置換命令を取得

（3）命令クラス

命令クラスは、命令に関する情報を統括するクラスである。命令クラスのインスタンスは、フォーマットクラスのインスタンスを単一で持ち、命令の構成（命令形式）を決定している。また、フィールドの値を保持している。

命令クラスのプロパティについては、表 6 に定義した事項を実装した。表 11 に命令クラスのメソッドについて示す。

表 11：命令クラスのメソッド

メソッド	メソッド名	機能
名前設定	setname(char[]);	命令名を設定
フォーマット設定	setInformat(Format);	フォーマットインスタンスを設定
フィールド値設定	setInvalue(int);	フィールドの値を設定
動作記述設定	setbehavior(int,char[]);	動作記述の設定
名前取得	getname();	命令名の取得
フォーマット取得	getInformat();	フォーマットインスタンスの取得
フィールド値取得	getInvalue(int);	フィールドの値の取得
動作記述取得	getbehavior(int);	動作記述の取得

（4）フォーマットクラス

フォーマットクラスは、フォーマット（命令形式）に関する情報を統括するクラスである。フォーマットクラスのインスタンスは、フィールドクラスのインスタンスを複数持つことにより命令形式を決定している。

フォーマットクラスのプロパティについては、表 5 に定義した事項を実装した。表 12 にフォーマットクラスのメソッドについて示す。

表 12：フォーマットクラスのメソッド

メソッド	メソッド名	機能
名前設定	setname(char[]);	フォーマット名の設定
フィールド設定	setFofield(Field);	フィールドインスタンスを設定
フィールド要素設定	setFofiname(char[]);	フィールド要素を設定
フィールド数設定	setFofinum(int);	フィールドの数を設定
名前取得	getname();	フォーマット名の取得
フィールド取得	getFofield ();	フィールドインスタンスの取得
フィールド要素取得	getFofiname (int);	フィールド要素の取得
フィールド数取得	getFofinum ();	フィールドの数を取得

（5）フィールドクラス

フィールドクラスは、フィールドに関する情報を統括するクラスである。

フィールドクラスのプロパティについては、表 4 に定義した事項を実装した。表 13 にフィールドクラスのメソッドについて示す。

表 13：フォーマットクラスのメソッド

メソッド	メソッド名	機能
名前設定	setname(char[]);	フィールド名の設定
種類設定	setFitype(int);	フィールドの種類を設定
幅設定	setFiwidth(int);	フィールド幅を設定
名前取得	getname();	フィールド名の取得
種類取得	getFitype ();	フィールド種類の取得
幅取得	getFiwidth();	フィールド幅の取得

3.4　ISID プログラムの解析を行うクラス群

ISID プログラムの解析を行うクラス群は、字句解析クラス、構文解析クラス、意味解析クラス、エラー処理クラスで構成されている。ISID プログラムの解析を行うクラス群では、処理の流れが重要になるため、UML のシーケンス図の作成を行った。図 7 に ISID プログラムの解析を行うクラス群についてのシーケンス図を示す。

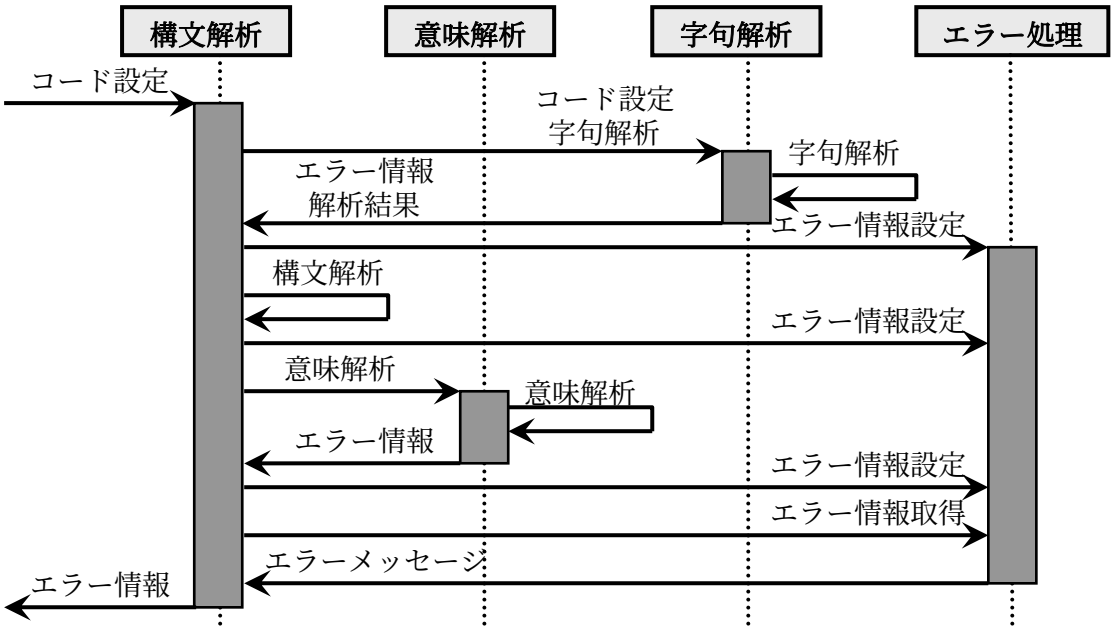

図 7：ISID プログラムの解析を行うクラス群の UML シーケンス図

これらのクラス毎の詳細について説明する。

（1） 字句解析クラス

字句解析クラスは、ISID プログラムを字句毎に区切るクラスである。字句解析を行い、その結果を保持して、構文解析クラスの要求により１つずつ字句を渡していく。そして、字句解析でのエラー情報を構文解析クラスに渡す。

字句の切り分け方について説明していく。図 8 に ISID プログラムの字句の文法規則を示し、表 14 に図 8 の字句の構成を基にした字句の種類と値の対応を示す。

```
文字列      ＝  予約語｜識別子
予約語      ＝  iset | field | constant | variable | format | inst | dummy｜substitution｜
                behavior｜architecture｜widthofword｜numberofregister｜numberofmemory
識別子      ＝  {a～z｜A～Z}+  {１０進定数}
数字列      ＝  １０進定数｜@１０進定数
１０進定数＝  非 0 数字  {数字}
数字        ＝  ０～９の任意の数字
記号        ＝  { | } | [ | ] | ; | =
空白        ＝  /s｜/t
コメント    ＝  //…
```

図 8：字句解析の文法規則

表 14：字句の種類と値の対応表

種類	0	1	2	3	4	5
値	予約語	識別子	数字列	@数字列	記号	その他
0	iset	―			{	―
1	field	―			}	―
2	constant	―			[	―
3	variable	―			]	―
4	format	―	値	値	;	―
5	inst	―	を	を	=	―
6	dummy	―	代	代		―
7	substitution	―	入	入		―
8	behavior	―				―
9	architecture	―				―
１０	widthofword	―				―
１１	numberofregister	―				―
１２	numberofmemory	―				―

※ その他の場合は、エラーとして構文解析クラスにエラー情報を渡す。

表 15 に字句解析クラスのプロパティについて、表 16 に字句解析クラスのメソッドについて示す。

表 15：字句解析クラスのプロパティ

プロパティ名	プロパティ	説明
字句	char token[];	字句を配列で保存
識別子	int type[];	字句の種類
判別子	int value[];	字句の値
行数	int line[];	字句の行数
列数	int position[];	字句の列数
エラー番号	int error[];	エラー情報の番号

表 16：字句解析クラスのメソッド

メソッド	メソッド名	機能
コード設定	setcode(char);	アセンブリコードの設定
識別子取得	gettype(int);	識別子の取得
判別子取得	getvalue(int);	判別子の取得
行数取得	getline(int);	行数の取得
列数取得	getposition(int);	列数の取得
エラー番号取得	geterror();	エラー情報の取得
字句解析	distributetoken();	字句解析

字句に関する情報を字句毎の配列で格納することにより、構文解析クラスが順番に１字句ずつ取り出して解析を行うことができる。

（2） 構文解析クラス

構文解析クラスは、ISID プログラムの解析を行う際の最上位クラスである。字句を次々に受け取り、解析を行う。図 9 に解析を行う ISID プログラムの構文を示す。

```
ISID プログラム　＝　アーキテクチャ情報　　命令セット情報
アーキテクチャ情報　＝　architecture{　アーキテクチャ内容　};
アーキテクチャ内容　＝　widthofword　ワード幅;
                        numberofregister　汎用レジスタ数;
                        numberofmemory　メモリ容量;
ワード幅　＝　１０進定数
汎用レジスタ数　＝　１０進定数
メモリ容量　＝　１０進定数

命令セット情報　＝　iset　命令長　ISID 名　{　内容　}　;
命令長 = １０進定数
ISID 名 = 文字列
内容=　{フィールド}+　　{フォーマット}+　　{命令}+　{擬似命令}+

フィールド =　field　種類　フィールド名　[　幅　]　;
種類 = constant | variable
フィールド名 = 文字列
幅 = １０進定数

フォーマット =　format　フォーマット名　{　{フィールド名定義}+　};
フォーマット名 = 文字列
フィールド名定義 = フィールド名　フィールド要素 ;
フィールド要素 = 文字列

命令 = inst フォーマット名 命令名 {　{フィールド値定義}+　動作定義　};
命令名 = 文字列
フィールド値定義 =　フィールド要素　=　値 ;
動作定義　＝　behavior{　{動作記述}+　};
動作記述　＝　シミュレーション命令　{オペランド}+
シミュレーション命令 = AND | OR | XOR | NOT | SUB | MUL | DIV | SLL | SRA | LD
                      | ST | STR | NOP | SLT | SEQ | BNZ | BEZ | JUMP | HALT
オペランド = フィールド要素
値 = １０進定数　|　@１０進定数

擬似命令 = dummy 命令名 擬似命令名 {　{擬似命令定義}+　置換命令　};
擬似命令名 = 文字列
擬似命令定義 =　フィールド要素　=　@１０進定数 ;
置換命令 = 命令名　{オペランド}+
```

図 9：ISID プログラムの構文

上記の文法に沿って解析を行う。

表 17 に構文解析クラスのプロパティについて、表 18 に構文解析クラスのメソッドについて示す。

表 17：構文解析クラスのプロパティ

プロパティ名	プロパティ	説明
字句解析	LaxicalAnalysis LA;	字句解析のインスタンス
意味解析	MeaningAnalysis MA;	意味解析のインスタンス
エラー処理	ErorrTable ET;	エラー処理のインスタンス
エラー番号	int error;	エラー情報
エラー数	int error_count;	エラーの数

表 18：構文解析クラスのメソッド

メソッド	メソッド名	機能
コード設定	setcode(char);	アセンブリコードの設定
エラー設定	seterror(int);	エラー情報の設定
エラー数取得	geterror_count();	エラー数の取得
エラー情報取得	geterror(int);	エラー情報取得
構文解析	analysis_top();	構文解析を行う

（3） 意味解析クラス

意味解析クラスは、ビット幅に関するエラーや宣言の重複など、意味的な観点から解析を行う。意味解析でのエラーは構文解析に渡して、構文解析からエラークラスに格納する。

表 19 に意味解析クラスのプロパティについて、表 20 に構文解析クラスのメソッドについて示す。

表 19：意味解析クラスのプロパティ

プロパティ名	プロパティ	説明
エラー番号	int error;	エラー情報

表 20：意味解析クラスのメソッド

メソッド	メソッド名	機能
エラー情報取得	geterror(int);	エラー情報の取得
意味解析	analysis_top();	意味解析を行う

（4） エラー処理クラス

エラー処理クラスはエラーに関する情報を蓄える。そして、解析が終了した際にエラーがある場合に、エラーメッセージを返す。1 つのエラーに対して 1 つのインスタンスを生成し、エラーが発生するごとにインスタンスが生成される。また、エラーが多く発生した場合に処理が止まってしまうので、エラーは 100 個までを保持し、それ以後のエラーは保持されない。以下に各解析プロセスのエラーを示す。

＜字句解析のエラー＞

○ 設定外の記号の検出

＜構文解析エラー＞

○ 予約語宣言の入力誤り
○ 定数宣言が来るべき箇所の誤り
○ 文字列宣言が来るべき箇所の誤り
○ 記号などが来るべき箇所の誤り
○ 予期せぬ文字列の検出

＜意味解析エラー＞

○ フィールドの未定義
○ フォーマットの未定義
○ 命令の未定義
○ 型の異なる定数をフィールドに代入（定数　or　@＋定数）
○ フィールド宣言でビット幅が命令長をオーバー
○ フォーマット宣言で総フィールド幅が命令長をオーバー
○ 擬似命令宣言で constant フィールドを宣言（variable にしか宣言できない）
○ 擬似命令・命令宣言で同じフィールドに値を割付
○ 命令宣言での constant フィールドに代入する値のビット幅がフィールド幅オーバー

表 21 にエラー処理クラスのプロパティについて、表 22 にエラー処理クラスのメソッドについて示す。

表 21：エラー処理クラスのプロパティ

プロパティ名	プロパティ	説明
エラー番号	int error;	エラー情報
行数	int line;	エラーの行数
列数	int position;	エラーの列数
字句	char token[];	エラーの字句
メッセージ	char message[];	エラーメッセージ

表 22：エラー処理クラスのメソッド

メソッド	メソッド名	機能
メッセージ取得	getmessage();	エラーメッセージの取得

これらのクラスを実装することにより、ISID の動作部分が完成する。共同で研究を進めている中川氏を中心として、実装したクラスと UI との結合を行った[11]。

4　汎用アセンブラの設計と実装

汎用アセンブラとは、ユーザが定義した命令セットに対してアセンブルを行うツールである。解析を行い、アセンブリコードの正当性を確認する。そして、アセンブリコードを命令セット情報に従って機械語に翻訳する。また、アセンブリコードを汎用シミュレータが動作できるように、擬似命令を命令に変換した汎用シミュレータコードを生成する。

4.1　外部仕様

汎用アセンブラの外部仕様は、大きく分けて４つある。以下にその４点を示す。

（１） アセンブリコードの基本構文

（２） アセンブリコードの解析

（３） アセンブリコードの機械語変換

（４） アセンブリコードの汎用シミュレータ用コード生成

この４点について仕様を固めることにより、汎用アセンブラを実装する。これらを順に説明していく。

（１） アセンブリコードの基本構文

汎用アセンブラを独自の命令セットに対してアセンブルできるようにするために、アセンブリコードの基本構文を統一する。表 23 にアセンブリコードの基本構文を示す。

表 23：アセンブリコードの基本構文

定義事項	決定事項
命令（op ラベル）	アルファベット大文字のみ
reg ラベル	‘$’+１０進定数
mem ラベル	‘＊’+１０進定数
ラベル（JUMP 先のアドレス）	アルファベット+：
定数	１０進定数
コメント	//　＋　コメント文

（２） アセンブリコードの解析

独自の命令セットに対するアセンブリコードを解析して、正当性の確認を行う。解析の手順として、字句解析、構文解析、意味解析を順に行う。

①字句解析では、アセンブリコードの字句を切り出す。また、コメントの切捨てを行う

②構文解析では、命令セット情報を基にして、命令に対してフィールドの構成が正しいか判別する

③意味解析では、数字などの意味的な誤りがないかを選別する

これらにより、命令セットに対応したアセンブリコードの正当性の確認を行う。

（3） アセンブリコードの機械語変換

命令セット情報を基にして、作成したプロセッサを FPGA ボード上で実行できるようにアセンブリコードを機械語に変換する。変換する際の仕様を以下に示す。

○ FPGA 上で実行できるよう、1 文字（1byte）に対して 8bit の 2 進数で表記
○ フォーマットに対応したフィールドの並びで出力
○ 擬似命令は参照する命令のフォーマットに対応

上記の仕様に従い、機械語変換の機能を設計し実装する。

（4） アセンブリコードの汎用シミュレータ用コード生成

独自の命令セットのアセンブリコードに対して、汎用シミュレータで動作させるために汎用シミュレータ用コードの生成を行う。汎用シミュレータコードの概要を以下に示す。

○ コード形式は、アセンブリコードの形式（表 23）に従う
○ 擬似命令を参照している命令に置き換える
○ コメントの切捨て

上記の仕様を実現することで、汎用シミュレータ上で動作できるコードが生成できる。

4.2 内部仕様

汎用アセンブラの内部構成は、UI をトップとして、大きく分けて 2 つになる。アセンブリコードを解析するクラス群と、命令セット情報の保持と受け渡しを行うクラス群である。

汎用アセンブラのクラス図と ISID のクラス図の構成は同じようなものとなっている。

解析に関するクラス群では、字句解析、構文解析、意味解析を順に行い、機械語とシミュレーションコードを生成して出力する。ISID の解析と似た構成をしているが、解析の対象がアセンブリコードとなっており、解析の方法が異なる。クラス毎の違いの詳細については次節で説明する。

命令セット情報に関するクラス群は、ISID で作成したクラスと同じものを使用している。命令セットに関する情報を保持して、その上で命令セット情報を取得する。図 10 に汎用アセンブラのクラス図を示す。

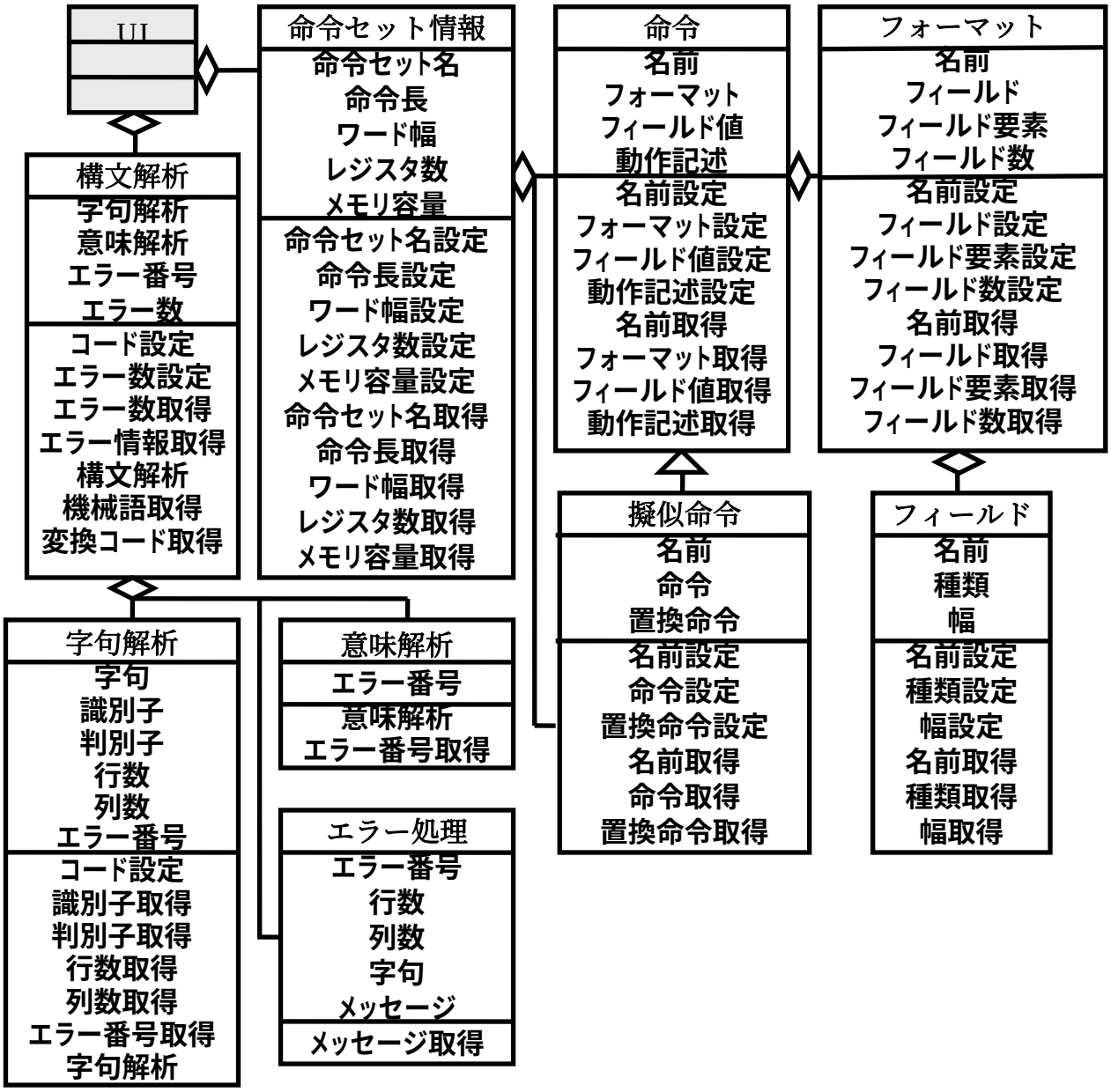

図 10：汎用アセンブラのクラス図

命令セット情報に関するクラス群については前章で説明したので、以下では、アセンブリコードを解析するクラス群について言及する。詳細については、次節で説明する。

4.3 アセンブリコードを解析するクラス群

アセンブリコードを解析するクラス群は、字句解析クラス・構文解析クラス・意味解析クラス・エラー処理クラスで構成されている。図 11 にアセンブリコードを解析するクラス群について、UML のシーケンス図を示す。

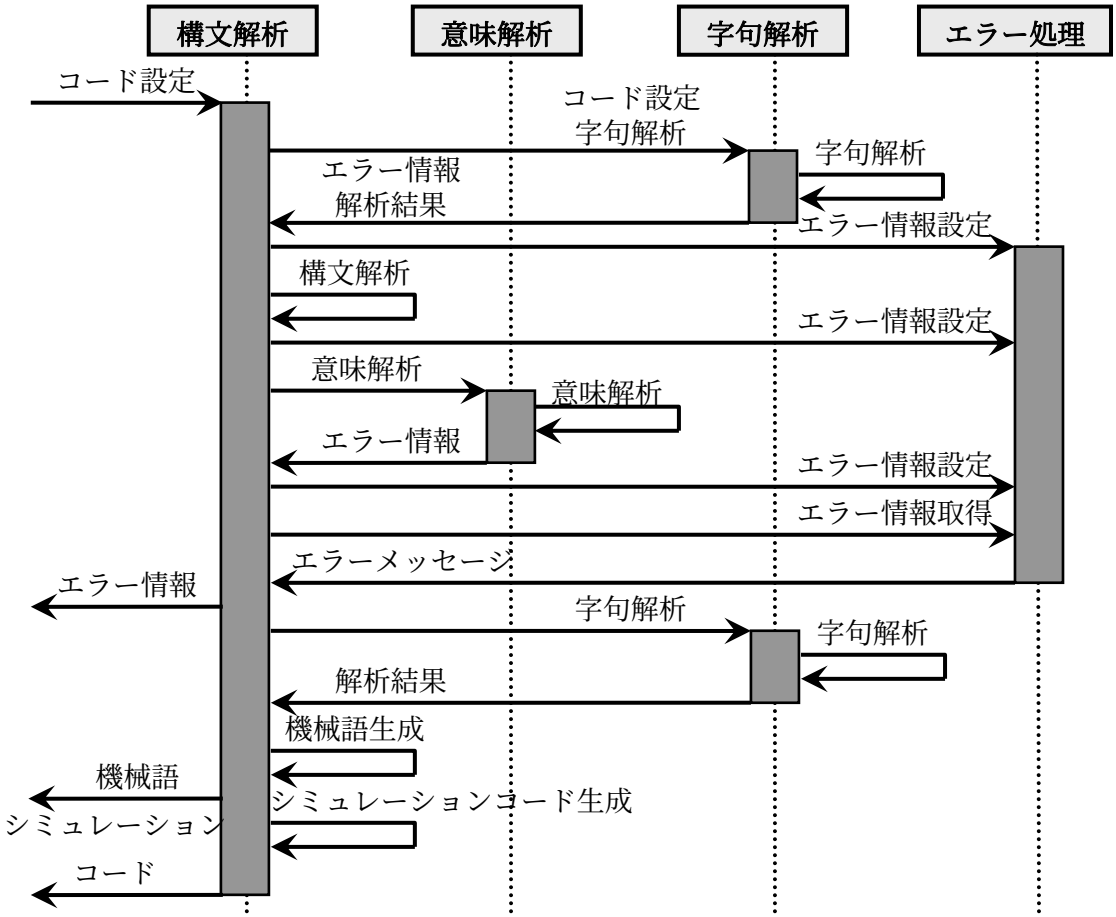

図 11：アセンブリコードを解析するクラス群のシーケンス図

これらのクラス毎の詳細について説明していく。

（1） 字句解析クラス

字句解析クラスは、ユーザが定義した命令セットに対するアセンブリコードを字句ごとに区切るクラスである。字句解析を行い、構文解析クラスの要求により１つずつ字句を渡していく。そして、字句解析でのエラー情報を構文解析クラスに渡す。

字句解析クラスの字句の切り分け方について説明していく。

まず、汎用アセンブラの字句の構成について示す。以下に汎用アセンブラの字句の文法規則を示す。

○ 命令　={A~Z}+

○ ラベル＝{a~z | A~Z}+ {10 進定数}　：

○ 定数　＝非 0 数字{数字}

○ ＄数字＝＄{数字}+

○ ＊数字　＝＊{数字}+

○ その他＝上記以外の記号

これらの字句の構成を基にして字句解析を行う。表 24 に字句の種類と値の対応を示す。

表 24：字句の種類と値の対応表

種類	字句	値
0	命令	命令表のアドレスを代入
1	ラベル	ラベル表のアドレスを代入
2	定数	値を代入
3	$数字	値を代入
4	＊数字	値を代入
5	その他	使用せず

※その他の場合はエラーとして構文解析クラスにエラー情報を渡す

表 25 に字句解析クラスのプロパティについて、表 26 に字句解析クラスのメソッドについて示す。

表 25：字句解析クラスのプロパティ

プロパティ名	プロパティ	説明
字句	char token[];	字句を配列で保存
識別子	int type[];	字句の種類
判別子	int value[];	字句の値
行数	int line[];	字句の行数
列数	int position[];	字句の列数
エラー番号	int error[];	エラー情報の番号

表 26：字句解析クラスのメソッド

メソッド	メソッド名	機能
コード設定	setcode(char);	アセンブリコードの設定
識別子取得	gettype(int);	識別子の取得
判別子取得	getvalue(int);	判別子の取得
行数取得	getline(int);	行数の取得
列数取得	getposition(int);	列数の取得
エラー番号取得	geterror();	エラー情報の取得
字句解析	distributetoken();	字句解析

（2） 構文解析クラス

構文解析クラスは、アセンブリコードの解析を行う際の最上位のクラスである。字句を順番に受け取り、アセンブリコードの文法について解析を行う。構文解析クラスの動作は大きく分けて 3 つある。以下にその 3 点について示す。

① ユーザ独自のプロセッサに対するアセンブリコードの解析
② FPGA ボード上で実行可能となる機械語への翻訳
③ 汎用シミュレータの動作用のシミュレーションコードの生成

① ユーザ独自のプロセッサに対するアセンブリコードの解析

図 12 にアセンブリコードの文法を示す。

アセンブリコード ＝ {命令}+
命令 ＝ {ラベル} オペレータ {オペランド}+
ラベル ＝ アルファベット ：
オペレータ ＝ ISID で定義された命令
オペランド ＝ 10 進定数 ｜ $10 進定数 ｜ ＊10 進定数 ｜ラベル

図 12：アセンブリコードの文法

この文法に従い、構文解析を行う。

② FPGA ボード上で実行可能となる機械語への翻訳

命令セット情報を基にして、作成したプロセッサを FPGA ボード上で実行できるようにアセンブリコードを機械語に翻訳する。以下に機械語生成のための条件を示す。

○ FPGA 上で実行できるよう、1 文字（1byte）に対して 8bit の２進数で表記
○ フォーマットに対応したフィールドの並びで出力
○ 擬似命令は参照する命令のフォーマットに対応

③ 汎用シミュレータの動作用のシミュレーションコードの生成

独自の命令セットのアセンブリコードに対して、汎用シミュレータで動作させるために汎用シミュレータ用コードの生成を行う。以下にシミュレータコード生成のための条件を示す。

○ コード形式は、アセンブリコードの形式に従う
○ 擬似命令を命令に置き換える
○ コメントの切捨て

表 27 に構文解析クラスのプロパティについて、表 28 に構文解析クラスのメソッドについて示す。

表 27：構文解析クラスのプロパティ

プロパティ名	プロパティ	説明
字句解析	LaxicalAnalysis LA;	字句解析のインスタンス
意味解析	MeaningAnalysis MA;	意味解析のインスタンス
エラー番号	int error;	エラー情報
エラー数	int error_count;	エラーの数

表 28：構文解析クラスのメソッド

メソッド	メソッド名	機能
コード設定	setcode(char);	アセンブリコードの設定
エラー数設定	seterror_count(int);	エラー数の設定
エラー数取得	geterror_count();	エラー数の取得
エラー情報取得	geterror(int);	エラー情報取得
構文解析	analysis_top();	構文解析を行う
機械語取得	getmachinelungage();	機械語の取得
変換コード取得	getcode();	シミュレーションコードの返却

（3） 意味解析クラス

意味解析クラスでは、オペランドの値がフィールド幅をオーバーしていないかを判別する。意味解析のエラーは構文解析に渡して、構文解析からエラー処理クラスに格納する。

表 29 に意味解析クラスのプロパティについて示す。表 30 に構文解析クラスのメソッドについて示す。

表 29：意味解析クラスのプロパティ

プロパティ名	プロパティ	説明
エラー番号	int error;	エラー情報

表 30：意味解析クラスのメソッド

メソッド	メソッド名	機能
エラー情報取得	geterror(int);	エラー情報の取得
意味解析	analysis_top();	意味解析を行う

（4） エラー処理クラス

エラー処理クラスはエラーに関する情報を蓄えて、エラーメッセージを作成する。１つのエラーに対して１つのインスタンスを生成する。すなわち、エラーが発生する毎にエラー処理クラスのインスタンスを生成する。以下に各エラープロセスのエラーを示す。

＜字句解析のエラー＞

○ 設定外の記号、識別子の検出

＜構文解析のエラー＞

○ ラベル名に命令名を使用

○ ラベルではない識別子の検出

○ 命令に対するオペランドの数の不一致

＜意味解析のエラー＞

○ オペランドの値のフィールド幅オーバー

表 31 にエラー処理クラスのプロパティについて示す。表 32 にエラー処理クラスのメソッドについて示す。

表 31：エラー処理クラスのプロパティ

プロパティ名	プロパティ	説明
エラー番号	int error;	エラー情報
行数	int line;	エラーの行数
列数	int position;	エラーの列数
字句	char token[];	エラーの字句
メッセージ	char message[];	エラーメッセージ

表 32：エラー処理クラスのメソッド

メソッド	メソッド名	機能
メッセージ取得	getmessage();	エラーメッセージの取得

これらのクラスを実装することにより、アセンブリコードを解析することができる。

5　命令セット定義ツール（ISID）と汎用アセンブラのテストと評価

ISID と汎用アセンブラの動作検証と汎用性の検証を複数の命令セットを用いて行った。検証に用いた命令セットは SOAR[6]と MONI[10]である。また、これらを同研究室の修士課程の方々に使用してもらい評価を行った。ISID と汎用アセンブラの検証と評価について以下で説明する。

5.1　命令セット定義ツール（ISID）の各機能の正当性の検証

ISID の各機能の正当性を検証するために複数の命令セットを用いて、ISID の動作検証を行った。動作検証を行うために、ISID の機能について再考した。以下に ISID の機能を示す。

○ ユーザオリジナルの命令セットの定義をサポート
○ 流れに沿った命令セットの定義
○ ISID プログラムのファイル出力
○ ISID プログラムの解析と命令セット情報の格納
○ 汎用アセンブラと汎用シミュレータの動作のための情報出力

上記の項目において、ISID の動作検証を行う。

ISID の正当性の検証を行うために、テスト項目の選定を行った。以下に、テスト項目について示す。

- 命令セット情報を正しく格納できているか
- 作成された ISID プログラムは正しいか
- 格納された情報を正しく出力できているか
- 解析は正しく動作しているか
- エラー情報は正しいか

以上の点について、SOAR、MONI 命令セット（ISID プログラム）に対して、正常に動作していることを確認した。

5.2　汎用アセンブラの各機能の正当性の検証

汎用アセンブラの各機能の正当性を検証するために ISID で定義した命令セットに対する複数のアセンブリコード用いて、汎用アセンブラの動作検証を行った。動作検証を行うために、汎用アセンブラの機能について再考した。以下に汎用アセンブラの機能を示す。

○ 独自の命令セットに対するアセンブリコードの解析
○ 機械語の出力
○ シミュレーションコードの出力

上記の項目において、汎用アセンブラの動作検証を行う。検証に用いたアセンブリコードは、SOAR[6]、MONI[10]それぞれの最大値・最大公約数・挿入ソートである。汎用アセンブラの動作検証について説明していく。

汎用アセンブラの正当性の検証を行うために、テスト項目の選定を行った。以下に、テスト項目について示す。

- 解析は正しく動作しているか
- エラー情報は正しいか
- 翻訳された機械語は正しいか
- 出力されたシミュレーションコードは正しいか

以上の点について、SOAR,MONI 命令セットに対する複数のアセンブリコードで、正常に動作していることを確認した。

5.3 汎用性の検証

ISID と汎用アセンブラの汎用性の検証を行った。まず、命令セットアーキテクチャを構成する要素の探索を行った。以下に命令セットアーキテクチャの構成要素を示す。

○ 命令長
○ 命令形式（命令フォーマット）
○ 命令セット
○ MPU の構成様式
○ レジスタ
○ アドレッシングモード
○ メモリアーキテクチャ
○ 擬似命令（マクロ命令）

上記の視点から汎用性の検証を行う。表 33 に各項目の検証結果を示す。

表 33：命令セットアーキテクチャの構成要素に対する検証

命令セットアーキテクチャ構成要素	検証結果
命令長	8、16、32、64bit に対応 命令長は固定
命令形式	自由に設定可能
MPU の構成様式	RISC に対応、CISC には未対応
命令セット	自由に設定可能
レジスタ	汎用レジスタは自由に設定可能 専用レジスタは用意してあるもののみ使用可 ＜用意してあるレジスタ＞ プログラム・カウンタ（PC） スタック・ポインタ（SP）

	＜用意していないレジスタ＞ インデックス・レジスタ アキュームレータ ステータス・レジスタ（フラグ・レジスタ）
アドレッシングモード	＜設定可能なモード＞ ディスプレースメント付レジスタ間接モード インデックス付レジスタ間接モード ディスプレースメント付 PC 相対モード メモリ間接 PC 相対モード インデックス付 PC 相対モード ロード/ストア・アドレス指定モード 絶対アドレッシングモード イミディエイト（即値）モード ＜設定不可能なモード＞ 直接アドレスモード （メモリとアドレスのワード幅が不一致の場合） メモリ間接モード
メモリアーキテクチャ	命令メモリとデータメモリで構成 単体のメモリでも使用可能
擬似命令（マクロ命令）	１命令のみにオペランドを変えて使用可能

命令長は 8、16、32、64bit から選択し、固定である。そのため、RISC に対応するが、CISC には対応することができない。

命令セット、命令形式は自由に設定可能であるが、汎用シミュレータを動作させるためには、表 1 に示されている 21 種類の命令を組み合わせてできる命令でなければならない。

レジスタに関しては、汎用レジスタは自由に設定可能である。専用レジスタに関しては、汎用シミュレータ上にプログラム・カウンタ、スタック・ポインタが用意されており、これらを用いて命令の動作記述を定義する。インデックス・レジスタ、アキュームレータ、ステータス・レジスタ（フラグ・レジスタ）に関しては用意されていない。そのため、汎用レジスタを適宜利用して補う形となる。

アドレッシングモードに関しては、代表的なものに関して検証を行った。ISID 上で意識的に決定する項目はないが命令の動作記述を定義する際に決定する。RISC に対応しているため、設定不可能なアドレッシングモードがある。

擬似命令に関しては、１命令には対応している。しかし、マクロ命令のような複数の命令を用いた命令を作成することができない。

これらが ISID、汎用アセンブラ上で設定できる項目となる。

5.4　評価

ISID と汎用アセンブラ機能性を評価するために、これらを使用してもらった上でアンケートを行った。それぞれの評価について説明していく。

（1） ISID の評価

ISID を使用してもらい機能性の評価を行った。機能性の評価のアンケートを行うために、質問する項目の選定を行った。以下にアンケートでの質問項目を示す。

① RISC プロセッサに対して、作成したい命令セットを実現できるか
② 命令セットアーキテクチャを定義する情報は妥当か
③ エラー情報はわかりやすいか
④ どのように改善すべきか

上記の質問項目に対するアンケートの結果を以下に示す。

① RISC プロセッサに対して、作成したい命令セットを実現できるか
- 実現できるが、1 命令ごとの定義が細かい
- 複数の命令を定義するのは、面倒である
- ある程度知識があれば可能
- 命令と擬似命令の違いがわからない

② 命令セットアーキテクチャを定義する情報は妥当か
- 定義すべき項目が多い
- わかりにくい表現が含まれている
- 学習用として十分である
- 命令と命令の動作記述が混同してしまう

③ エラー情報はわかりやすいか
- わかりやすい
- 英語を併記すればベスト

④ どのように改善すべきか
- GUI を使用中にバグが出るときがあるので、安定した動作ができるように
- 解析に関してはほぼ完璧ではないか

ISID を用いて、学習者が定義したい命令セットを実現できるレベルに到達した。しかし、汎用性を求めた結果、定義すべき項目多くなってしまった。汎用性を保ちながら、また汎用性の範囲を選定しながら、わかりやすく定義できるものにしていく必要がある。また、ただ定義を容易にするだけではなく、学習者の理解を促す必要がある。汎用性、機能性、学習者の理解のバランスが重要である。

擬似命令に関して、命令と１対１でしか対応ができていないため、擬似命令と命令の違いが明確になっていないことが判明した。

（2） 汎用アセンブラの評価

汎用アセンブラを使用してもらい機能性の評価を行った。機能性の評価のアンケートを行うために、質問する項目の選定を行った。以下にアンケートでの質問項目を示す。

① 定義した命令セットに対して正しいアセンブル結果を得られるか
② エラー情報はわかりやすいか
③ どのように改善すべきか

上記の質問項目に対するアンケートの結果を以下に示す。

① 定義した命令セットに対して正しいアセンブル結果を得られるか
- 正しい結果を得られる

② エラー情報はわかりやすいか
- エラー情報は的確である
- バグがある箇所は推定できる
- エラー表示は親切である

③ どのように改善すべきか
- アセンブラは間違いを起こしてはいけないので、十分なほどにデバッグをすること
- ISID との連携

アセンブラは安定に動作することが重要である。現時点では、正常に動作しているが、どのようなバグが発生するか予測できない。しっかりデバッグを行い、バグを無くしたい。

5.5 考察

（1） 検証に関する考察

ISID は、命令セットアーキテクチャの設計を行えるツールとなった。また、汎用アセンブラは定義した命令セットをアセンブルでき、MONI 基盤のハード/ソフト・コラーニングシステムの拡張の礎を築くことができたのではないかと思う。さらなる発展、よりよい学習システムを目指し、改善すべき点について考えた。以下に重要度を考慮して、改善点を示す。

＜重要な改善点＞

○ 使用するレジスタの選択
○ アドレッシングモードの選択
○ マクロ命令にも対応

学習者にレジスタとアドレッシングモードの理解を促すために必要である。また、マクロ命令にも対応させ汎用性を広げたい。

＜改善してもよい点＞

○ 命令長の可変にも対応

○ CISC にも対応

命令長の可変、CISC にも対応については、プロセッサの理解を促すという点に重点をおき、RISC を対象とした学習システムであるため、早急に改善する必要はないかと考える。しかし、CISC のプロセッサを構築したい学習者も存在するかもしれないので、将来的には改善してもよいと考える。

（2） 評価に対する考察

命令セットアーキテクチャを定義するには問題がないが、定義すべき項目が多く、命令セットアーキテクチャを理解していなければ難しい学習システムであることがわかった。学習者にとって有用な情報を選定し、汎用性の範囲を選定し直す必要がある。汎用性と機能性の調和のとれたシステムの探求を目指す。

また、擬似命令と命令の違いが明確ではないという問題がある。これは、擬似命令と命令が１対１対応となっているためである。マクロ命令として、複数の命令を組み合わせて実行できるシステムへと拡張したい。また、命令の動作記述と命令の区別がつきづらい点を解消する必要があるのではないかと考える。

さらに、命令と命令の動作記述に関して混同してしまう点を解消する必要があるのではないかと考える。アセンブリコードの仕様と命令の動作記述（汎用シミュレータ上で動作可能な命令＜表 1＞）の仕様が似た形になっている。表 34 に、アセンブリコードの１命令と命令の動作記述の 3 オペランド方式の ADD を例として示す。

表 34：命令と命令の動作記述の例の比較

	アセンブリコードの１命令	命令の動作記述
記述例	ADD $1 $2 $2	ADD rd rt rs;
動作内容	$1 = $2 + $2	rd = rt + rs

上記のように似た構文となっているため混同しやすく、学習者の理解を妨げる可能性がある。命令の動作記述の構文を動作内容のように演算子で表現するのが良いのではないかと考える。

6　おわりに

本研究では、ハード/ソフト・コラーニングシステムを、プロセッサ設計に関して自由度の向上と学習環境の充実を目的に拡張を行った。その構成要素である ISID、汎用アセンブラを設計・実装し、検証・テストを行った。

命令セット情報の定義が行える ISID と定義した命令セット用のアセンブラとして使える汎用アセンブラを導入することにより、ハード/ソフト・コラーニングシステムのソフトウェア学習の環境が充実した。

今後の展望として、ISID と汎用アセンブラについて第５章で述べた改良を行いたい。命令セットアーキテクチャの作成の補助だけではなく、命令セットアーキテクチャを体系的に理解できるような学習システムを目指す。そして、学習システムの提案だけではなく、実際の教育現場で導入されるようなシステムへと発展し、プロセッサアーキテクチャを理解した学生が、将来のシステム LSI の開発の最前線で活躍できるようにすることがこのシステムの終着点であると考える。本システムが更なる発展を遂げ、終着点に到達することを願っている。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導をいただきました山崎勝弘教授、小柳滋教授に深く感謝いたします。

また、本研究の共同研究者である中村氏、難波氏、Hoang 氏、中川氏、西田氏に深く感謝いたします。

最後に、ISID と汎用アセンブラの評価にご協力いただいた高性能計算研究室の修士課程の皆様に心より感謝いたします。

参考文献

[1] 末吉敏則：「FPGA/PLD 最前線」－歴史から最新動向まで－、5 都市 FPGA カンファレンス 2005、pp. 9-60、2005.

[2] 安浦寛人：システム LSI 時代の設計技術の動向と課題、日本セロックシカテクノロジーセミナー、2005.

[3] 古川達久：FPGA 上でのソフト・マクロ CPU によるハードウェア/ソフトウェア分割手法の研究、立命館大学理工学研究科修士論文、2005.

[4] 梅原直人：ハード/ソフト最適分割を考慮した AES 暗号システムと JPEG エンコーダの設計と検証、立命館大学理工学部情報学科卒業論文、2005.

[5] 小林真輔：コンフィギャラブル・プロセッサを理解する－コンパイラ検証のための基礎知識、Design Wave Magazine、2003. 12.

[6] 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.

[7] 中村浩一郎：FPGA ボードコンピュータシステムの開発、立命館大学理工学部卒業論文、2004.

[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/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.

[10] 池田修久、中村浩一郎、Tran So Cong、難波翔一朗：RC100 を用いた FPGA ボードコンピュータ　設計仕様書 Ver0.7.3.2、立命館大学理工学研究科　高性能計算研究室・コンピュータシステム研究室、2004.

[11] 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ーフェースの設計と実装、立命館大学理工学部情報学科卒業論文、2006.

[12] 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情報学科卒業論文、2006.

[13] 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.

[14] John L.Hennessy,Dabid A. Patterson 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.

[15] 清水尚彦　著：コンピュータ設計の基礎知識―ハードウェア・アーキテクチャ・コンパイラの設計と実装、共立出版、2003.

[16] 中森章　著：マイクロプロセッサ・アーキテクチャ入門、CQ 出版、2004. 4.

[17] 小林優　著：改訂・入門 Verilog HDL 記述－ハードウェア記述言語の速習＆実践

[18] 愛甲健二　著：アセンブリ言語の教科書、データハウス、2005. 7.

[19] 林晴比古　著：高級言語プログラマのためのアセンブラ入門、ソフトバンククリエイ

ティブ、2005．11.
[20] 山本信雄　著：プログラマ養成入門講座 Visual C++１、２、３、翔泳社、1999．8
[21] 林晴比古　著：新 VisualC++6.0 入門　ビギナー、シニア編、 ソフトバンクパブリッシング、 2001.
[22] 浦昭二、原田賢一　著：C 入門、培風館、1994．6
[23] 西村克信、 額田多政、 天野秀晴：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、電子情報通信学会技術研究報告、Vol.99、No.532(CPSY99106-116)、pp.61-68、2000．1.
[24] 高橋隆一、児島彰、上土井陽子、吉田典可：マイクロコンピュータ設計教育環境 City-1 FPGA 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、Vol.97、No.17(DA-83)、pp41-48、1997．2.
[25] 田中康一郎、小羽田哲宏、久我守弘、末吉敏則：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、Vol49(ARC-100)、pp59-66、1993．6
[26] 末吉敏則、小羽田哲宏、野崎貴弘、田中康一郎、久我守弘：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、Vol94、No50(ARC-106)、pp25-32、1994．6.
[27] 土江竜雄、佐々木敬泰、弘中哲夫、児島彰：教育研究用スーパスカラ・プロセッサ・シミュレータ Mikage の概要、情報処理学会研究報告、Vol96、No.80(ARC-119)、pp107-112、1996．8.
[28] 井上弘士、中垣憲一、大内正英、久我守弘、末吉敏則：教育用 RISC 型マイクロプロセッサ DLX－FPGA とそのラピッドシステムプロトタイピング、電子情報通信学会技術研究報告、Vol.95、No.25(ICD95 11-22)、 pp.71-78、1995．4
[29] 今井慈郎、富田眞治、古川善吾、井面仁志、白木渡、石川浩、大和田昭邦：計算機システム教育のためのビジュアルシミュレータ VisuSim、情報処理学会研究報告、Vol.2001、No.34(CE-59)、pp77-84、2001．3.
[30] 大角圭吾、小久保政樹、西野洋介、早川栄一、システムソフトウェア教育支援環境「港」における実装レベルの OS 学習支援システム、電子情報通信学会技術研究報告、Vol.2005、No.15(CE-078), pp.7-12 、2005.2.
[31] 原田実、塩見彰睦：教育用マイクロプロセッサ SEP4 を用いた設計演習の提案,情報処理学会全国大会講演論文集、Vol.65、No.4、pp4.277-4.278、2003．3