卒業論文

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ーフェースの設計と実装

氏名：中川 裕樹
学籍番号：2210020311-5
指導教官：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

本研究ではプロッセッサアーキテクチャをソフト/ハード両面から学習する教育システムであるハード/ソフト・コラーニングシステムの拡張のための汎用アセンブラと命令セット定義ツール（以下 ISID と略す）のユーザインターフェースの設計と実装、処理部分との結合を行った。

ハード/ソフト・コラーニングシステムは、アーキテクチャ理解を意識したソフトウェア学習と、実際にプロセッサ設計を行い、また FPGA ボードに搭載、検証することでアーキテクチャ理解を深めるハードウェア学習から構成される。従来のハード/ソフト・コラーニングシステムでは 16 ビット命令セット（MONI）に基づいたシステムであったが、命令セットを独自に定義できる環境を整備するため、独自の命令セットに対応できる汎用アセンブラと、その中で用いる命令セット定義ファイルの作成支援ツールである、ISID の設計と実装を行った。

開発言語は Visual C++を用い、ユーザインターフェースは SDI（Single Document Interface）形式を採用した単一の窓構成となっている。アプリケーションの見せ方を決める View クラス、命令セットを定義するプロパティシートを中心にプログラミングを行い、処理クラスである命令セットクラス、ファイルを保存する Document クラス、SDI のフレームを構成する MainFrame クラスなどとデータをやり取りしながら設計した。

本論文では、まずハード/ソフト協調学習の詳細と、その拡張について述べ、ハード/ソフト・コラーニングシステムの構成要素である、ISID と汎用アセンブラのユーザインターフェースの設計、実装、評価を行う。

目次

1　はじめに 2
1.1　研究背景 2
1.2　研究目的 4
1.3　研究内容 5
2　命令セット定義ツール（ISID）の設計 6
2.1　要求仕様 6
2.2　命令セット定義ツールの機能 7
2.3　命令セット定義ツールの構成画面 7
2.4　命令セット定義ツールの使用法 9
2.5　命令セット定義ツールのモジュール構成 11
3　汎用アセンブラの設計 12
3.1　要求仕様 12
3.2　汎用アセンブラの機能 12
3.3　汎用アセンブラの構成画面 12
3.4　汎用アセンブラのモジュール構成 14
4　VC++によるユーザインターフェースの実装 16
4.1　開発環境 16
4.2　命令セット定義ツール（ISID）のユーザインターフェースの実装 17
4.3　汎用アセンブラのユーザインターフェースの実装 22
5　評価と考察 24
5.1　評価 24
5.2　考察 24
6　おわりに 25
謝辞 26
参考文献 27

図目次

図 1：ハード/ソフト・コラーニングシステムの学習体系 2
図 2：拡張後のハード/ソフト・コラーニングシステムの学習体系 4
図 3：MONI の命令セット定義ファイル例 6
図 4：ISID の構成画面 8
図 5：命令セット定義画面 9
図 6：ISID の使用法フローチャート 10
図 7：ISID のモジュール構成図 11
図 8：汎用アセンブラの構成画面 13

図 9：汎用アセンブラのモジュール構成図 14
図 10：ファイルの関係図 16

表目次

表 1：UserInterfaceView.cpp で作成したメンバ関数 17
表 2：myppage1 で作成したメンバ関数 18
表 3：myppage2 で作成したメンバ関数 19
表 4：myppage3 作成したメンバ関数 20
表 5：myppage4 で作成したメンバ関数 21
表 6：myppage5 で作成したメンバ関数 22
表 7：MainUIView.cpp で作成したメンバ関数 23

1　はじめに

1.1　研究背景

これまで社会の情報化を支えてきた LSI 技術は近年の急速な半導体集積技術の進歩によって大規模化、複雑化し、システム全体を１チップ上に載せるまでの能力に至った。一方、技術的な観点からは、高集積化に伴う問題が生じてきている。１チップ上に数億個レベル以上のトランジスタが載るようになると、設計やテストに膨大な時間が必要となる。[1][2]

システム LSI を、これまでよりはるかに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現在、LSI 設計技術と組み込みソフトウェア設計技術の協調が必要となる。システム LSI の基本構造とその設計手法は、情報工学を学ぶ者の基礎知識であり、ソフトウェア技術者や通信技術者にとっても必須の基礎技術となっている。しかし近年、ハードウェアとソフトウェアの切り分けが難しくなり、ハードとソフトの両面からシステム全体を見ることができる人の数が、社会の需要に追いつかなくなってきている。[3]

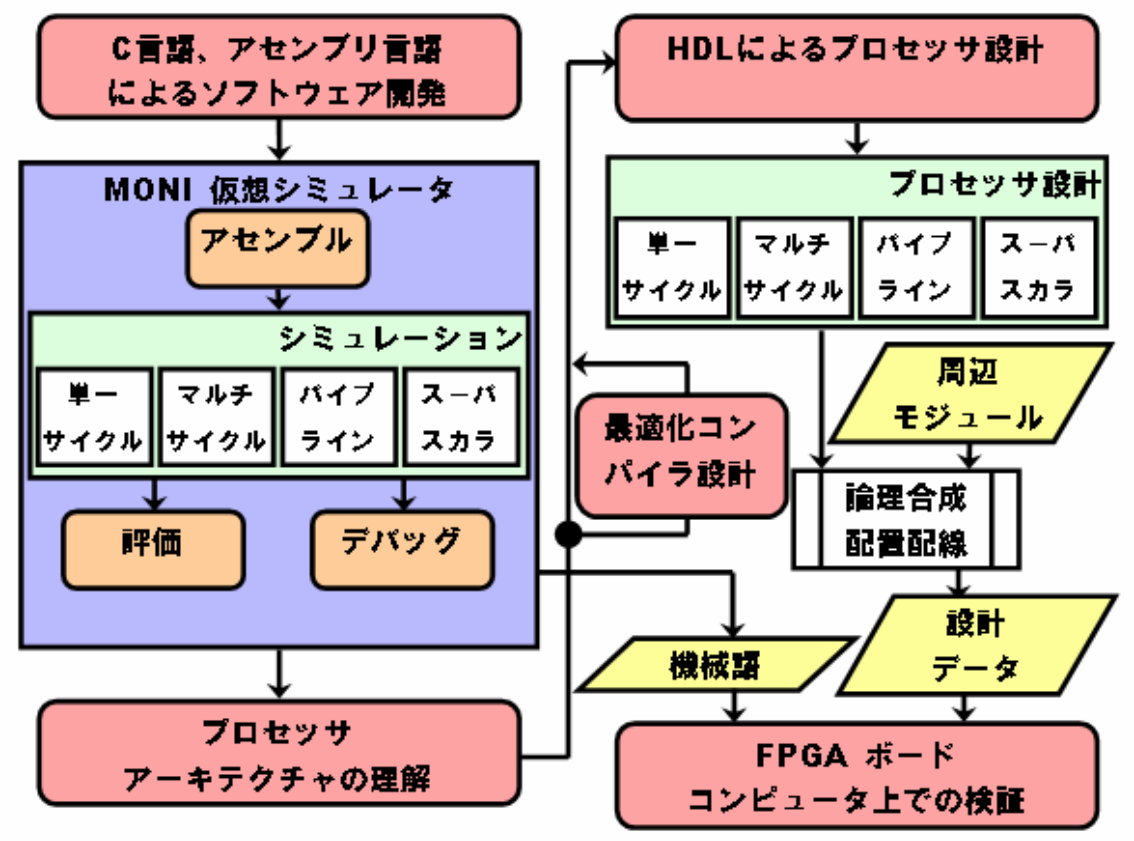

図 1：ハード/ソフト・コラーニングシステムの学習体系

このようにシステム LSI 開発においては、プロセッサ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にとってハードウェアとソフトウェア両方の知識は必要不可欠である。システム LSI を設計する上では、プロセッサアーキテクチャの理解が大前提 で、ハードウェア技術者であっても、開発プロセスや設計効率を意識して作業を進めることが

重要になってくる。ソフトウェア技術者においても、アーキテクチャに適したソフトウェア開発が求められている。これらを解決するために早期の教育が必要となり、大学におけるハードとソフトの関係を密にした教育の必要性がある。このため、我々はハード/ソフト・コラーニングシステムを立ち上げ、改良を行ってきた。

ハード/ソフト・コラーニングシステムとは、アーキテクチャ理解を意識したソフトウェア学習と、実際にプロセッサ設計を行うことでアーキテクチャ理解を深めるハードウェア学習から構成される学習システムである[4][5]。図 1 にオリジナルプロセッサの MONI（16 ビット命令セット）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体系を示す。ソフトウェア面では、プロセッサアーキテクチャ毎の最適化コンパイラや、プログラミングの評価を通して学習、ハードウェア面では、HDL シミュレータを用いて複数のプロセッサアーキテクチャを理解し、その知識を用いてプロセッサ設計を行う。また、設計したプロセッサを FPGA ボードに搭載し、実機検証を行うことで学習する。HDL によるプロセッサ設計とソフトウェア開発を融合させることで、アーキテクチャを意識したプログラミングが行えるようになることがこのシステムの目的である。

図 1 の左側がソフトウェア学習である。学習者は、MONI 仮想シミュレータ上でアセンブリ言語によるプログラミングを行う。MONI 仮想シミュレータでは、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 つから選択し、1 命令、または 1 フェーズごとにシミュレーションを行うことができる。これにより、学習者は MONI プロセッサの内部動作を理解しながらプログラミングができ、ハードウェアを意識したソフトウェアの設計手法が身につけることができる。

右側がハードウェア学習である。HDL を用いて実際に MONI プロセッサの設計を行い、FPGA 上に実装、プロセッサの動作検証を行う。

本システムの特徴をまとめると以下の通りである。

- ソフトウェアシミュレータによる可観測性とハードウェア設計を融合
  - ソフトウェアシミュレータだけでは分からないことを、ハードウェア設計を通して学べる
  - ボードコンピュータでは見えないプロセッサの内部動作をシミュレータで観測可
- プロセッサアーキテクチャを段階的に学習
  - なぜ今スーパースカラや VLIW なのか
  - どのようにプロセッサが発展してきたのか
- アーキテクチャを意識した上でのプログラミング演習
  - プロセッサを高速に動作させるプログラミングを学習
- 同じプログラムを異なるアーキテクチャで動作させ評価可能
  - プロセッサアーキテクチャの評価

1.2　研究目的

以上のような背景を踏まえて、よりハードウェアとソフトウェアを協調的に理解できる学習システムの探求を行う。そこで、ハード/ソフト･コラーニングシステムの検証を行うため、オリジナルプロセッサ SOAR の設計と実装を行った[6]。設計と実装を行った上で以下のような問題点が浮上した。

- 命令セットが MONI にしか対応できていない
- FPGA の実機検証が容易ではない
- 学習環境が充実していない

これらの問題点を解決するために、本システムを以下の 2 つの点で拡張した。図 2 に拡張後のハード/ソフト・コラーニングシステムの学習体系を示す。

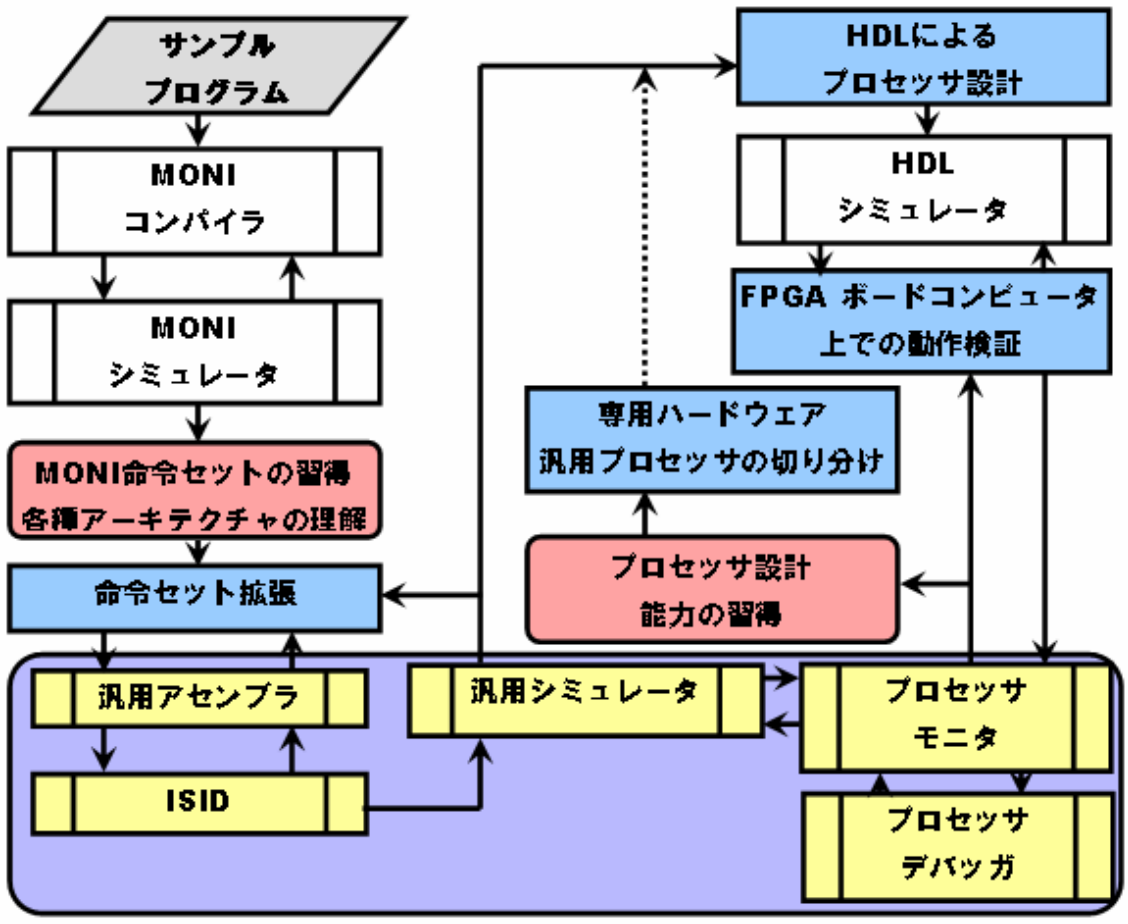

図 2：拡張後のハード/ソフト・コラーニングシステムの学習体系

(1)命令セットを独自に定義できる環境の整備

汎用アセンブラ、汎用シミュレータを開発支援ツールとして加えることで、MONI 基盤のハード/ソフト・コラーニングシステムを、独自の命令セットに対応できるシステムへと拡張した。また、汎用アセンブラ、汎用シミュレータを実現するため命令セットを定義す

る必要があり、本研究では命令定義とその情報のファイル入出力、作成支援ツールとしてISID の設計・実装も行った。

(2) ホスト PC から FPGA の内部を観測・制御

プロセッサデバッガをシステムに加えることで FPGA 内部の観測・制御を実現する。プロセッサデバッガとは、FPGA 上の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変更、またプログラムのステップ実行、プ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。

以下に学習体系の一連の流れを記す。

① ISID を用いて、命令セット定義ファイルを作成する。
② アセンブリ言語をファイルから読み出し、命令セット定義ファイルをもとにアセンブルを行う。
③ 命令セット定義ファイルをもとにシミュレーションを行う。
④ シミュレーションの結果をもとに、HDL によるプロセッサ設計を行う。
⑤ 設計したプロセッサを HDL シミュレータを用いてデバッグを行う。
⑥ プロセッサデバッガを用いて FPGA 上に実装・検証する。
⑦ ①～⑤のフローを繰り返し行い、よりよいプロセッサを設計する。

独自の命令セットに対応できるよう、開発支援ツールとして汎用アセンブラ、汎用シミュレータ、その中で用いる命令セット定義ファイルの作成支援ツールである、ISID を追加した。本研究では、これらの実装により命令セット定義可能なハード/ソフト・コラーニングシステムを実現し、命令セットアーキテクチャ、マイクロアーキテクチャの理解を目標とした学習フローの提案を行い、学部生を対象とした実験授業に取り入れられることを目的としている。

1.3 研究内容

我々は、ハード/ソフト・コラーニングシステムの拡張を行った。従来のシステムは 16 ビット命令セット（MONI）にしか対応できておらず、学習環境の充実という面では不完全といえた。

本研究では、システムの拡張のため、汎用アセンブラ、汎用シミュレータ、ISID の設計・実装を行った。開発言語は Visual C++を用い、私は汎用アセンブラと ISID のユーザインターフェースの設計と実装、命令セットクラス、構文解析クラスとの結合を担当した。

本論文では、まず２章で ISID の設計について述べる。３章では汎用アセンブラの設計について、４章では VC++によるユーザインターフェースの実装について、そして、５章と６章で ISID と汎用アセンブラの評価と考察について述べる。

2　命令セット定義ツール（ISID）の設計

2.1　要求仕様

ISID は、汎用アセンブラの実現のため、命令セット定義ファイル（アーキテクチャ情報、命令セット名、命令長、フィールド、フォーマット、命令の動作、擬似命令）の作成を支援するツールである。命令セット定義ファイルの記述例を図 3 に示す。

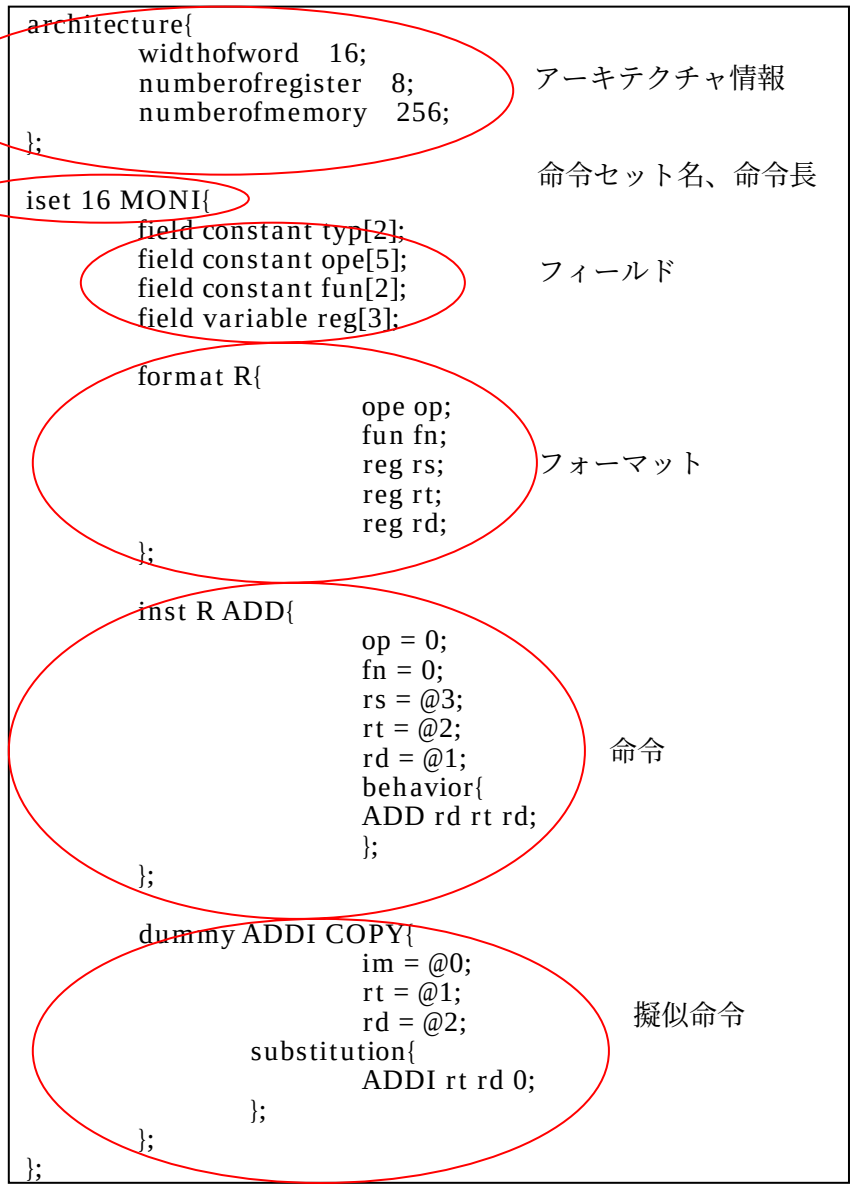

図 3：MONI の命令セット定義ファイル例

図 3 のように命令セット定義ファイルは、アーキテクチャ情報、命令セット名、命令長、フィールド、フォーマット、命令の動作、擬似命令の定義で構成され、一つの命令セットの情報を記述すると数百行になる。このファイルを利用者が作成するのには多くの時間が必要で、間違った箇所があれば汎用アセンブラが正確な処理を行うことができない。ファイルの記述を容易にし、記述ミスをなくすため ISID が必要である。

ユーザインターフェースには、命令セット定義ファイルを出力、変更、保存するためのエディタと、エラーチェックを行い、エラーを出力するためのリストボックス、命令定義のための画面が必要となる。定義の際には　≪命令長→フィールド→フォーマット→命令→擬似命令≫　という順序で行わなければならないため、それぞれの定義画面を分割し、順を追った定義ができるユーザインターフェースが必要となる。また、定義したフィールド情報がフォーマット定義画面のリストボックスに入っていなければならないように、命令セットクラスとの細かな結合が必要で、各所に情報の受け渡しをするためのメソッドが必要となる。命令セットクラスとは、命令セット情報、フィールド、フォーマット、命令、擬似命令クラスのことである。それぞれのクラスに情報の受け渡しに必要なメソッドが用意されている。処理部分に関しては同研究室の富樫氏が実装した。[7]

最後に、書きかけの命令セット定義ファイルを読み込み、途中からの作成、変更ができるようにエディット内の情報を読み込むためのボタンが必要となる。

2.2　命令セット定義ツールの機能

ISID の機能には以下のものがある。

- 命令セット定義ファイルの作成が、容易で正確にできる
- 命令セット定義ファイルの再編集ができる
  - テキスト記述した情報も ISID で読み込むことができる
- 作成した命令セット定義ファイルのエラーチェックを行う

2.3　命令セット定義ツールの構成画面

以上のことを踏まえて、設計した ISID の構成画面を図 4 に、命令セット定義画面を図 5 に示す。

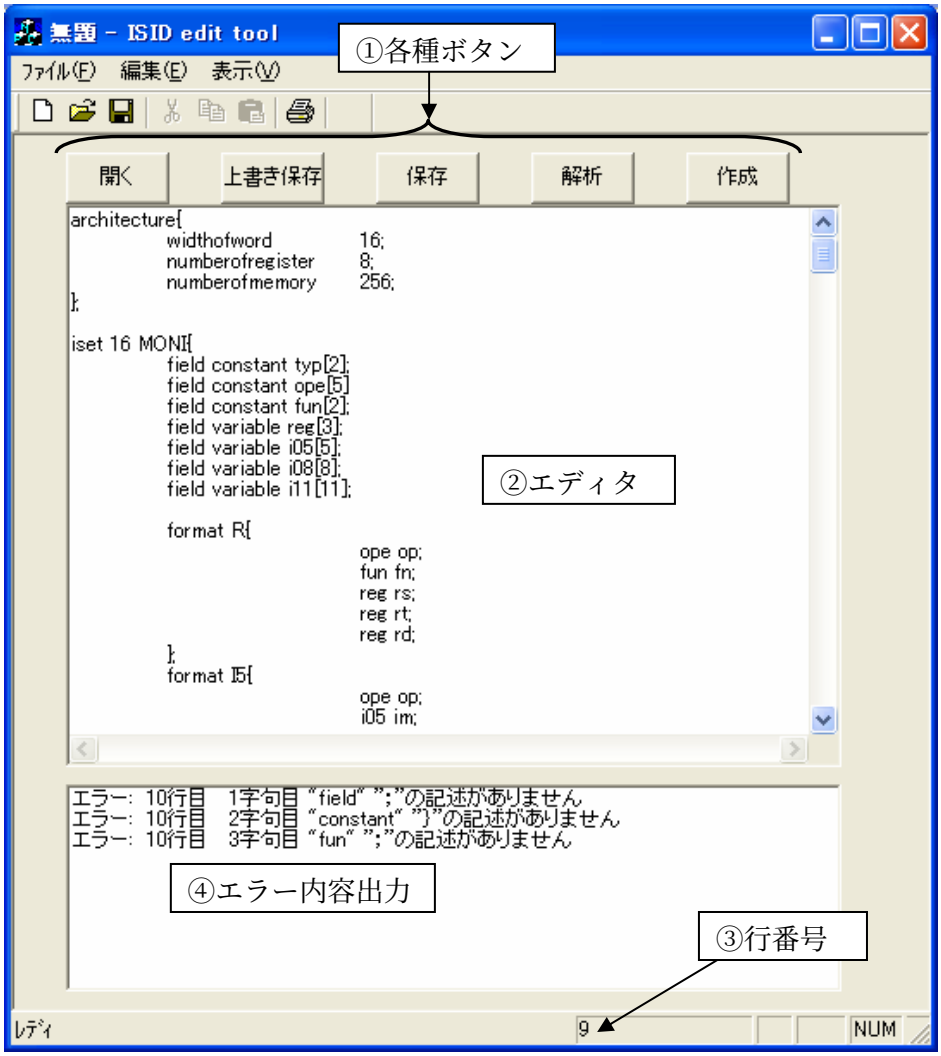

図 4：ISID の構成画面

ISID の構成の詳細について述べる。

① 各種ボタン

[開く]ボタン：既存の命令セットファイルをエディタに出力する。

[上書き保存]、[保存]ボタン：命令セットファイルの保存に用いる。

[解析]ボタン：命令セットファイルのエラーチェックを行う。エラーがない場合は、情報を命令セットクラスに送る。

[作成]ボタン：命令セットファイルを新規に作成する。図 5 に示した命令セット定義画面が現れる。

② エディタ

命令セットファイルの出力に用い、直接入力、変更が可能。

③ 行番号

エディタ上のカーソル位置の行番号を出力する。エラー箇所を特定する際に利用する。

④ エラー内容出力

命令セットファイルのエラーチェックを行う。エラー箇所の行番号、何字句目か、エラー内容を出力する。

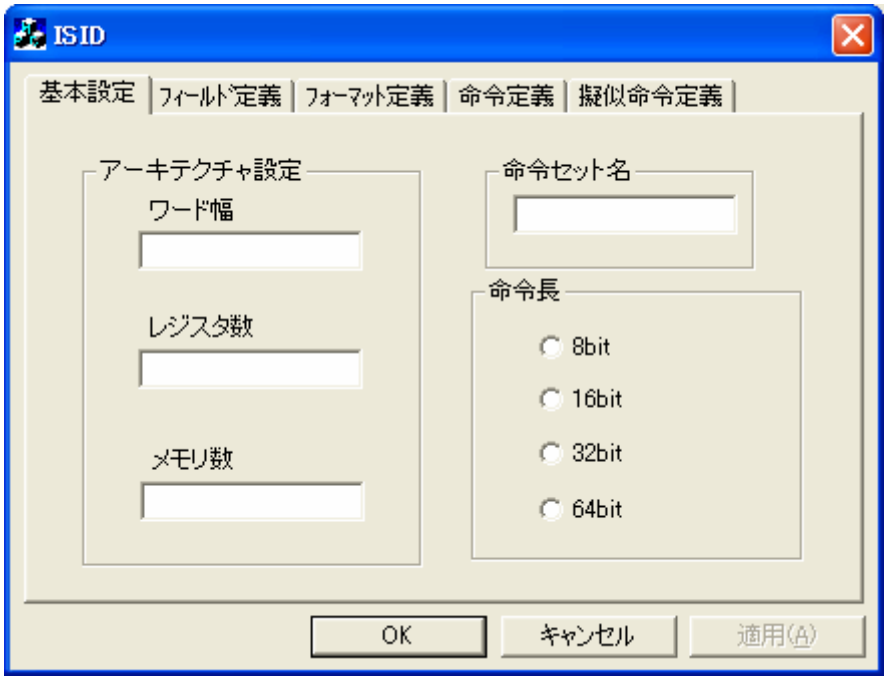

図 5：命令セット定義画面

命令セット定義画面は、基本設定（アーキテクチャ設定、命令セット名、命令長)、フィールド定義、フォーマット定義、命令定義、擬似命令定義画面から構成される。

画面の作成にはプロパティシートを用い、各定義画面をタブで分割することで順を追った定義が可能で、未定義のままタブ移動するとエラーが出力される。

タブ移動、[追加]ボタンのメソッドで命令セットクラスとの結合を行い、情報の受け渡しをしている。詳しい内容は 4.2 章命令セット定義ツール（ISID）のユーザインターフェースの実装で述べる。

2.4 命令セット定義ツールの使用法

命令セット定義ツールのフローチャートを図 6 に示す。

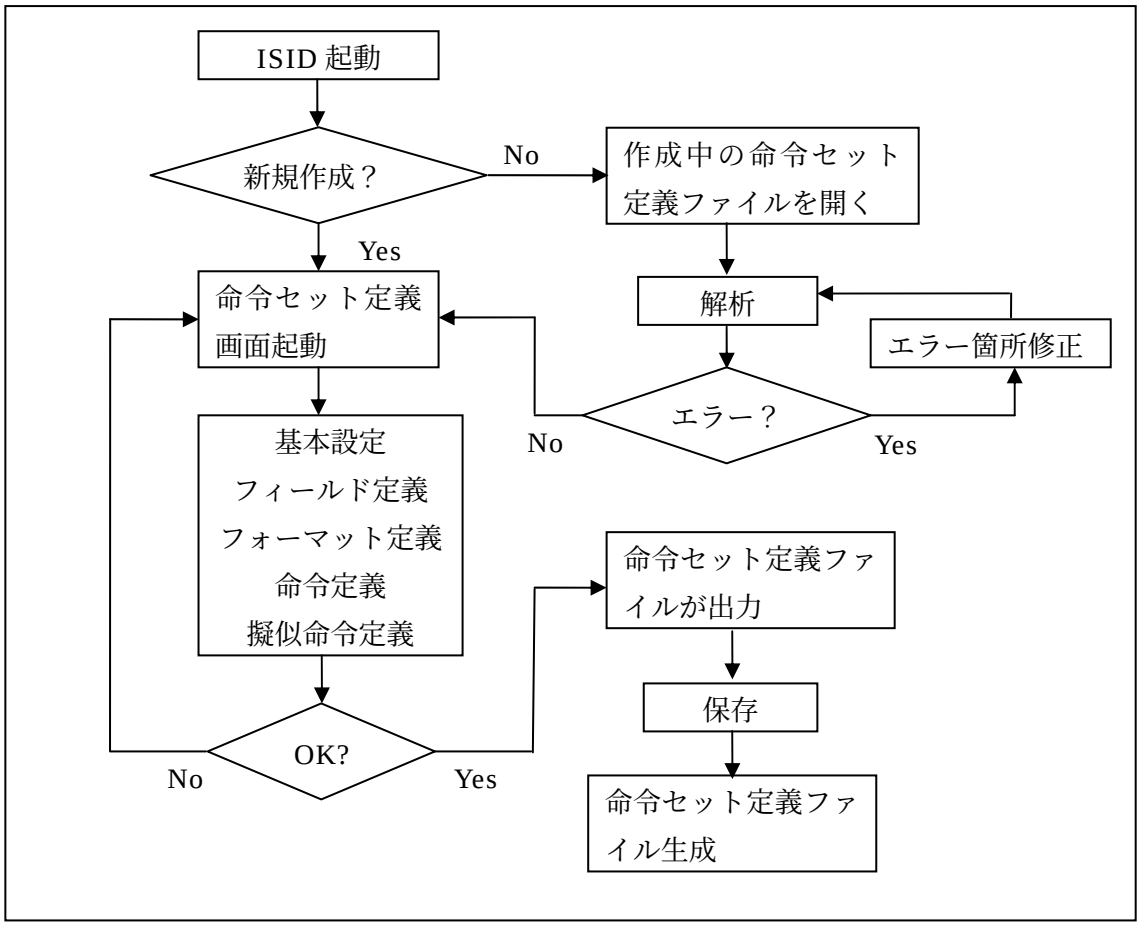

図 6：ISID の使用法フローチャート

(1)新規作成の場合

① ISID を起動、[作成]ボタンを押し、命令セット定義画面に必要事項を入力し、[OK]ボタンを押す。

② 命令セット定義画面が閉じ、エディタに定義した命令セット定義ファイルが出力される。

③ [保存]ボタンでファイル名を入力、保存場所を指定し保存する。

(2)途中から作成の場合

① ISID を起動、[開く]ボタンでエディタに作成中の命令セット定義ファイルを出力する。

② [解析]ボタンを押し、エラーチェックを行い、命令セットクラスにファイルの情報を送る。

③ [作成]ボタンを押すと命令セット定義画面が現れ、開いたファイルの内容が入っているので、残りを定義して[OK]ボタンを押す。

④ [上書き保存]する

2.5　命令セット定義ツールのモジュール構成

ISID のモジュール構成図を図 7 に示す。

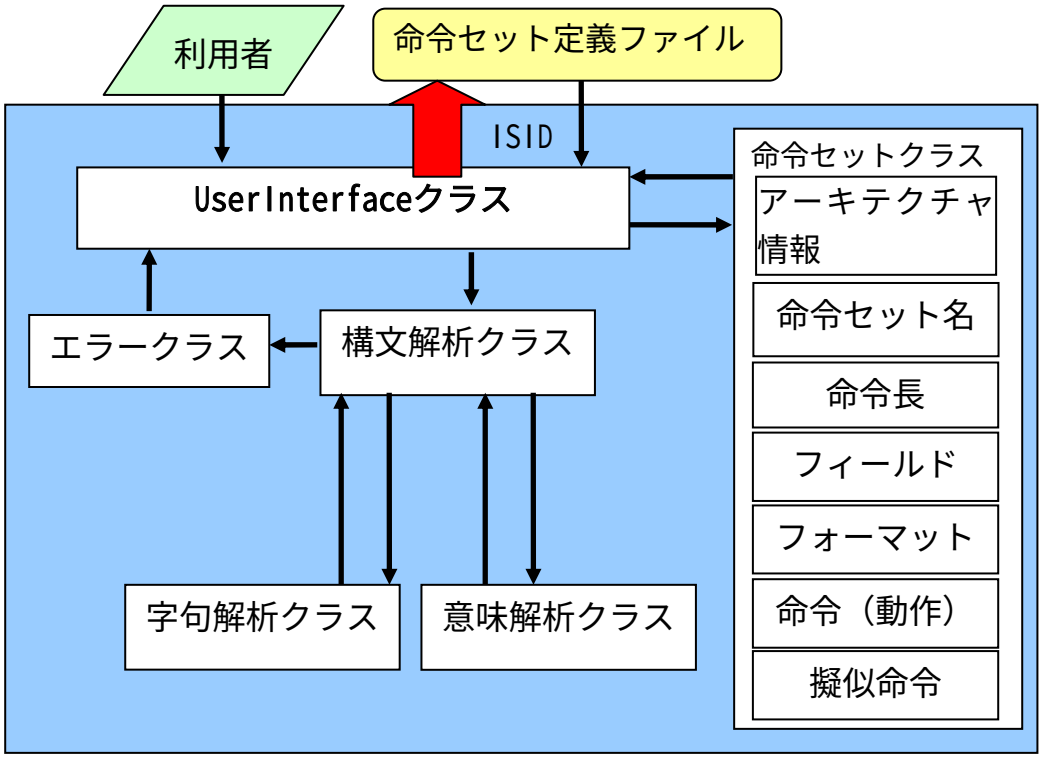

図 7：ISID のモジュール構成図

利用者はユーザインターフェースを用いて、命令セット定義ファイルの読み込み、出力、解析を行う。命令セットクラスには、アーキテクチャ情報、命令セット名、命令長、フィールド、フォーマット、命令、擬似命令の受け渡しに必要なさまざまなメソッドが用意されている。これらを用いて定義された情報を命令セットクラスに登録、また、必要なときには呼び出して出力し定義の支援をしていく。

途中から作成の場合、命令セット定義ファイルを読み込む際に、ファイルにエラーがないかチェックを行ってから命令セットクラスへと情報を登録する必要がある。ユーザインターフェースと構文解析との結合を行う。構文解析、字句解析、意味解析でファイルの文字チェック、文法チェック、ラベルの正当性チェック行い、エラーがあればエラー箇所の行番号、エラー内容を出力する。

3　汎用アセンブラの設計

3.1　要求仕様

設計する汎用アセンブラは、独自の命令セットに対応できるよう、命令セット名、命令長、フィールド、フォーマット、命令の動作、擬似命令が記述された命令セット定義ファイルとアセンブリソースプログラムを読み込んでアセンブルを行うものである。そのためユーザインターフェースには、アセンブリソースを出力するためのエディタ、命令セット定義ファイルを指定するためのボタン、アセンブルを行った際のエラー出力用リストボックスとエラー行を見つけ出すためのエディタ上のカーソル位置の行番号を出力できるようにする必要がある。また、汎用アセンブラのユーザインターフェースをメイン画面とし、そこから ISID と汎用シミュレータを開くボタンも用意する。なによりも、利用者にとって使いやすいユーザインターフェースの実装が課題となる。

3.2　汎用アセンブラの機能

汎用アセンブラの機能には以下のものがある。

- 学習者が定義した独自の命令セットのアセンブリ言語をアセンブル可
- 正確なエラー情報の出力
- 汎用アセンブラのユーザインターフェースをメイン画面とし、そこから ISID と汎用シミュレータを開くことが可能

3.3　汎用アセンブラの構成画面

以上のことを踏まえて、設計した汎用アセンブラの構成画面を図 8 に示す。

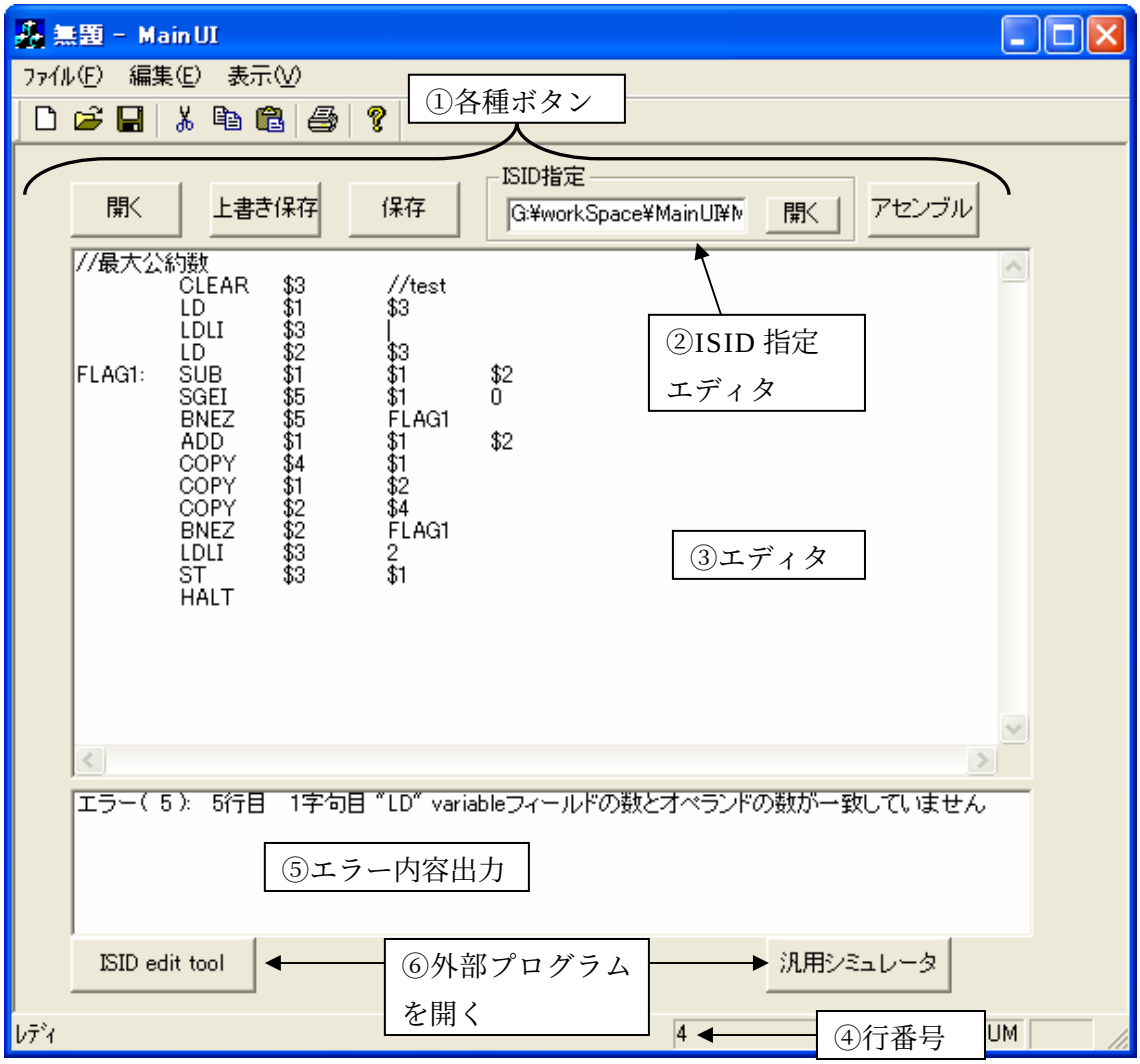

図 8：汎用アセンブラの構成画面

汎用アセンブラの構成の詳細について述べる。

① 各種ボタン

[開く]ボタン：既存の命令セットファイルをエディタに出力する。

[上書き保存]、[保存]ボタン：命令セットファイルの保存に用いる。

[アセンブル]ボタン：ISID を参照し、アセンブリソースプログラムをアセンブルする。エラーがあればリストボックスに内容を出力し、なければ機械語ファイルを作成する。

② ISID 指定エディタ

作成した命令セット定義ファイルを[開く]ボタンで検索し、パスをエディタに出力する。

③ エディタ

アセンブリソースプログラムの出力に用い、直接入力、変更が可能。

④ 行番号

エディタ上のカーソル位置の行番号を出力する。エラー箇所を特定する際に利用する。

⑤ エラー内容出力

アセンブリソースプログラムのエラーチェックを行う。エラー箇所の行番号、何字句目か、エラー内容を出力する。

汎用アセンブラの構成画面には、ISID と汎用シミュレータを起動するボタンを配置し、アセンブリソースプログラムと ISID を読み込んでアセンブルを行うため、命令セット定義ファイルのパスを出力するエディタを配置した。

アセンブルを行い、エラーがない場合には機械語ファイルを生成する画面が現れる。ファイル名を入力、保存場所を指定して操作終了となる。

3.4 汎用アセンブラのモジュール構成

汎用アセンブラのモジュール構成図を図 9 に示す。

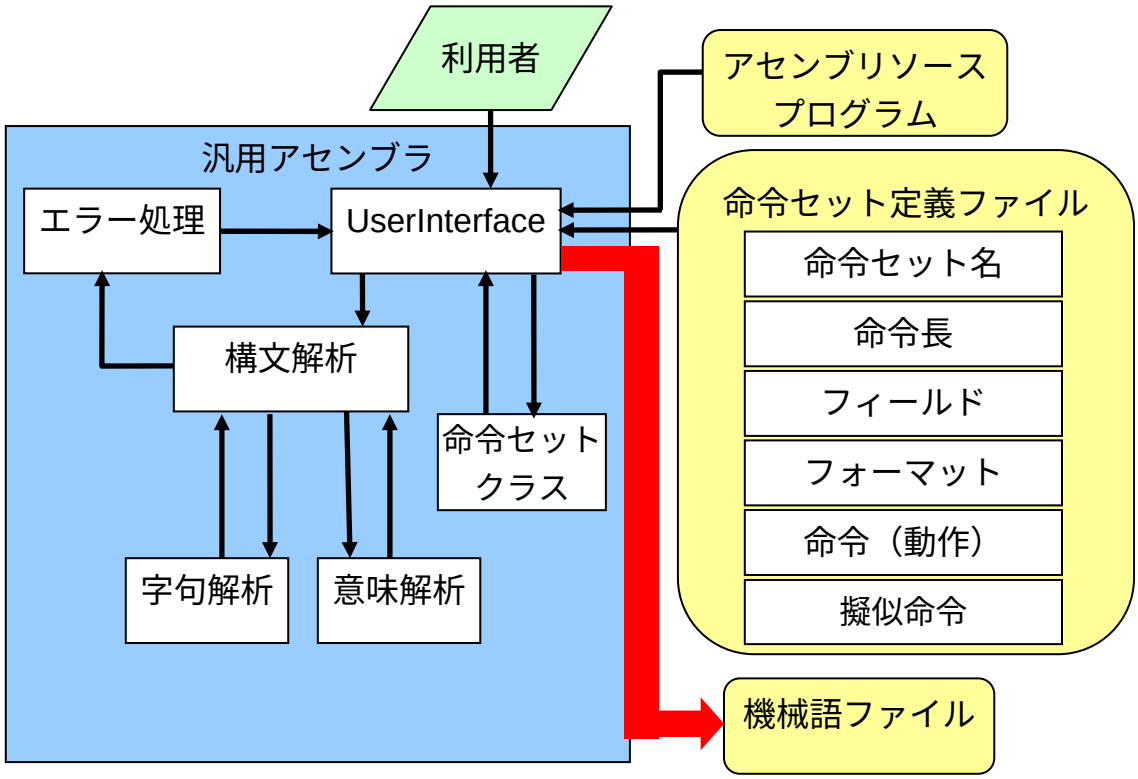

図 9：汎用アセンブラのモジュール構成図

利用者は汎用アセンブラの使用の前に命令セット定義ファイルを作成していないといけ

ない。ユーザインターフェースを用いて、命令セット定義ファイルのパスを読み込み。そのパスから命令セット定義ファイルを命令セットクラスに登録する。エディタ上にアセンブリソースプログラムを出力し、命令セット情報を基に構文解析、字句解析、意味解析でファイルの文字チェック、文法チェック、ラベルの正当性チェック行い、エラーがあればエラー箇所の行番号、エラー内容を出力する。エラーがない場合は機械語ファイルの名前、保存場所を指定して、作成する。

4　VC++によるユーザインターフェースの実装

4.1　開発環境

開発ツールとして VisualC++6.0 を用いた。SDI（Single Document Interface）形式を採用した単一の窓構成となっている。アプリケーションの見せ方を決める View クラス(UserInterfaceView)と命令セットを定義するプロパティシート(myppage1~5)を中心にプログラミングを行い、処理クラスである命令セットクラス、ファイルを保存する Document クラス(UserInterfaceDoc)、SDI のフレームを構成する MainFrame クラスなどとデータをやり取りしながら設計した。

図 10 に各クラスで使用するファイルの関係図を示す。

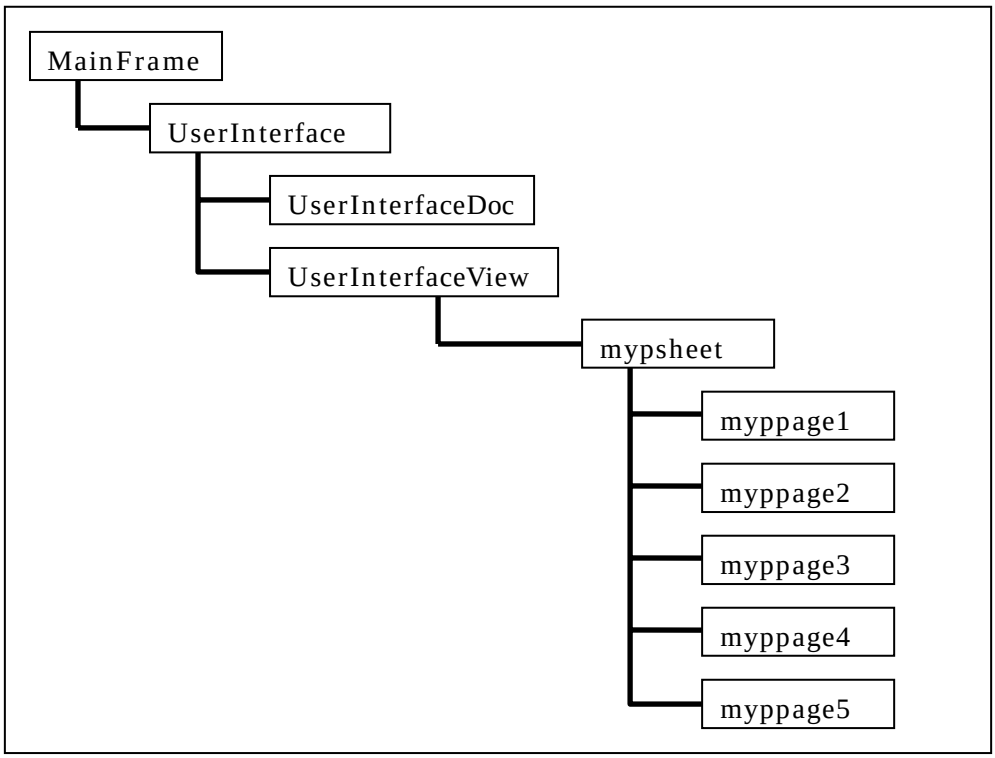

図 10：ファイルの関係図

プロジェクト名

命令セット定義ツール：UserInterface

汎用アセンブラ：MainUI

- MainFrm.cpp：SDI 形式アプリケーションのフレームを制御する
- UserInterface(MainUI)Doc.cpp：ユーザデータを保持する。ファイル編集を含む
- UserInterface(MainUI)View.cpp：ウィンドウの表示方法や見せ方を処理する
- UserInterface(MainUI).cpp：アプリケーションの中心となるプログラム

- UserInterface(MainUI).rc：ユーザが使用するリソースを列挙
- stdAfx.cpp：プリコンパイル済ヘッダーファイルを構築するためのプログラム
- mypsheet.cpp：以下のプロパティページを統括するプログラム
- myppage1.cpp：アーキテクチャ設定、命令セット名、命令長を定義するページ
- myppage2.cpp：フィールドを定義するページ
- myppage3.cpp：フォーマットを定義するページ
- myppage4.cpp：命令を定義するページ
- myppage5.cpp：擬似命令を定義するページ

4.2　命令セット定義ツール（ISID）のユーザインターフェースの実装

ISID とはユーザ独自の命令セットを定義したファイルの作成を支援するツールである。このツールを用いて作成した命令セット定義ファイルを読み込むことで、汎用アセンブラ/シミュレータが動作する。

GUI ベースでの編集のため、流れに沿った定義が可能で入力ミスを減らし、間違いがあった場合でも、解析を行い、エラーチェックが可能である。学習者はこのツールを用いて正確な命令セット定義ファイルを容易に作成することができる。以下では ISID のユーザインターフェースの実装と結合について述べる。

図 4 と図 5 に示したように ISID のユーザインターフェースは 5 つのボタン､エディタ、リストボックス、プロパティシートで構成される。

ISID 起動時に現れる画面は命令セット定義ファイルの出力、解析を行う画面である。表 1 にウィンドウの表示方法や見せ方を処理するプログラムである UserInterface.cpp で作成したメンバ関数を示す。

表 1：UserInterfaceView.cpp で作成したメンバ関数

メンバ関数	メッセージ	処理内容
OnButton1()	開くボタン	ファイル探索画面を出力し、ファイルを開く
OnButton2()	上書き保存ボタン	上書き保存する
OnButton3()	保存ボタン	保存先探索画面を出力しファイルを保存
OnButton4()	解析ボタン	命令セット定義ファイルのエラーチェック、命令セットクラスに情報を渡す
OnButton5()	作成ボタン	命令セット定義画面（プロパティシート）を開く
OnSize(UINT nType, int cx, int cy)		画面のサイズ、ボタンやエディタの配置を決定
OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)	エディタのカーソルが移動	ステータスバーにエディタのカーソル位置の行番号を表示

命令セット画面とは、命令セット定義ファイルのアーキテクチャ情報、命令長、命令セット名、フィールド、フォーマット、命令、擬似命令を定義する画面のことである。命令セット定義画面にプロパティシートを用いることで、多くの情報をわかりやすく表示し、各種の設定をすることができる。また順を追った定義ができるという利点がある。以下に命令セット定義画面についての詳細を述べる。

（1）基本設定画面（myppage1）

アーキテクチャ情報、命令セット名、命令長を定義する画面である。命令長はラジオボタンを用いて、8bit,16bit,32bit,64bit から選択する。アーキテクチャ情報、命令セット名はエディタに数字、文字列を入力する。

画面が現れた操作で命令セットクラスに情報があるか問い合わせる。もしあればその情報を表示する。新規定義 or 変更を行い、タブがほかの画面に移った操作の際に命令セットクラスにこれらの情報を登録する。また、未入力の箇所があれば、エラーウィンドウを出力する。

表 2 に基本設定画面（アーキテクチャ情報、命令セット名、命令長定義画面）で作成したメンバ関数について示す。

表 2：myppage1 で作成したメンバ関数

メンバ関数	メッセージ	処理内容
OnSetActive()	画面出現時	命令セットクラスに情報が登録されていればその情報を表示する
OnKillActive()	画面を失った時	アーキテクチャ情報、命令長、命令セット名を命令セットクラスに登録する 未入力の箇所があればそれに応じたエラーウィンドウを出力

（2）フィールド定義画面（myppage2）

フィールドを定義する画面である。フィールド名、種類（constant or variable）、ビット長を定義する。ビット長のリストボックスの上限は基本設定で定義した命令長である。

画面が現れた操作で命令セットクラスに情報があるか問い合わせる。もしあればその情報を表示する。命令長を受け取り、リストボックスに表示。新規定義 or 変更を行い、追加ボタンが押された際に命令セットクラスにこれらの情報を 1 つずつ登録し、ユーザ用に別に用意したリストボックスへ登録内容を表示する。また、削除ボタンで削除も可能である。

表 3 にフィールド定義画面で作成したメンバ関数について示す。

表 3：myppage2 で作成したメンバ関数

メンバ関数	メッセージ	処理内容
OnSetActive()	画面出現時	命令セットクラスに情報が登録されていればその情報を表示する リストボックスの初期化 ”constant”,”variable”を追加 ビット長を追加
OnButton1()	追加ボタン	選択、記述された情報を命令セットクラスに登録 ユーザ用にリストボックスへ表示 重複した文字列があるとエラーウィンドウ出力
OnButton2()	削除ボタン	命令セットクラスへ削除命令を出す ユーザ用にリストボックスから削除

（3）フォーマット定義画面（myppage3）

フォーマットを定義する画面である。フォーマット名、フィールド名、フィールド要素を定義する。基本設定で定義した命令長が表示されていて、ユーザはフォーマットの総ビット長がこれと同じになるように定義していく。

画面が現れた操作で命令セットクラスに情報があるか問い合わせる。もしあればその情報を表示する。また、命令長の値を受け取り、エディタに表示する。新規定義 or 変更を行い、追加ボタンが押された際に命令セットクラスにこれらの情報を 1 つずつ登録し、ユーザ用に別に用意したリストボックスへ登録内容を表示する。また、削除ボタンで削除も可能である。タブがほかの画面に移った操作の際に各フォーマットの総ビット長と命令長を比べ、一致していないとエラーウィンドウを出力する。

表 4 にフィールド定義画面で作成したメンバ関数について示す。

表 4：myppage3 作成したメンバ関数

メンバ関数	メッセージ	処理内容
OnSetActive()	画面出現時	命令セットクラスに情報が登録されていればその情報を表示する リストボックスの初期化 命令長の表示 フィールドの表示 総ビット長算出 フォーマット名、総ビット長の表示
OnButton2()	追加ボタン	選択、記述された情報を命令セットクラスに登録 ユーザ用にリストボックスへ追加 重複した文字列があるとエラーウィンドウ出力 総ビット長が命令長を超えるとエラーウィンドウ出力
OnButton8()	削除ボタン	命令セットクラスへ削除命令を出す ユーザ用にリストボックスから削除
OnKillActive()	画面を失った時	総ビット長と命令長が一致していないとエラーウィンドウ出力

（4）命令定義画面（myppage4）

命令を定義する画面である。命令名、フォーマット、動作、フィールド要素の値を定義する。フォーマットのリストボックスには定義したフォーマットが入っている。フォーマット選択時にはフィールド要素がリストボックスに表示される。またフィールド要素選択時には、フィールド要素の種類（constant or variable）に応じた引数が表示される。ユーザはこれを選択し、命令を定義していく。

画面が現れた操作で命令セットクラスに情報があるか問い合わせる。もしあればその情報を表示する。また、定義したフォーマットを出力する。新規定義 or 変更を行い、追加ボタンが押された際に命令セットクラスにこれらの情報を 1 つずつ登録し、ユーザ用に別に用意したコンボボックスへ登録内容を表示する。また、削除ボタンで削除、変更ボタンで変更も可能である。

表 5 にフィールド定義画面で作成したメンバ関数について示す。

表 5：myppage4 で作成したメンバ関数

メンバ関数	メッセージ	処理内容
OnSetActive()	画面出現時	命令セットクラスに情報が登録されていればその情報を表示する 定義したフォーマット名を表示
OnSelchangeList1()	フォーマット選択時	定義したフォーマットのフィールド要素を表示
OnSelchangeList3()	フィールド要素選択時	選択したフィールド要素が"constant"か"variable"か判別 "constant"ならビット長分表示 "variable"なら@1～@5 までを表示
OnSelchangeList4()	機械語の値選択時	選択した値を表示
OnSelchangeCombo1()	コンボボックスを選択	選択した命令に応じたフォーマットと値を表示
OnButton1()	追加ボタン	選択、記述された情報を命令セットクラスに登録 命令名の重複を制限 ユーザ用にコンボボックスに追加
OnButton8()	削除ボタン	命令セットクラスへ削除命令を出す ユーザ用にコンボボックスから削除
OnButton2()	変更ボタン	命令セットクラスへ変更命令を出す ユーザ用にコンボボックスを変更

（５）擬似命令定義画面（myppage5）

擬似命令を定義する画面である。擬似命令名、使用する命令、置換命令、フィールド要素の値を定義する。命令のリストボックスには定義した命令が入っている。命令選択時には、命令のフォーマットを表示し、そのフォーマットのフィールド要素がリストボックスに表示される。またフィールド要素選択時には、フィールド要素の種類（constant or variable）に応じた引数が表示される。ユーザはこれを選択し、擬似命令を定義していく。

画面が現れた操作で命令セットクラスに情報があるか問い合わせる。もしあればその情報を表示する。また、定義した擬似命令名をコンボボックスに表示する。新規定義 or 変更を行い、追加ボタンが押された際に命令セットクラスにこれらの情報を 1 つずつ登録し、ユーザ用に別に用意したコンボボックスへ登録内容を表示する。また、削除ボタンで削除、変更ボタンで変更も可能である。

表 6 にフィールド定義画面で作成したメンバ関数について示す。

表 6：myppage5 で作成したメンバ関数

メンバ関数	メッセージ	処理内容
OnSetActive()	画面出現時	命令セットクラスに情報が登録されていればその情報を出力する 定義した命令名をコンボボックスに出力
OnSelchangeList5()	命令選択	定義した命令名、置換命令、フォーマットとそのフィールド要素、値を出力
OnSelchangeList2()	値を選択	選択した値を出力
OnSelchangeList1()	フィールド要素を選択	機械語の値選択リストボックスに"未使用","@1～@5"を出力
OnSelchangeCombo2()	擬似命令名のコンボボックスを選択	定義した擬似命令情報を出力
OnButton7()	追加ボタン	選択、記述された情報を命令セットクラスに登録 擬似命令名の重複を制限 ユーザ用にコンボボックスに追加
OnButton6()	削除ボタン	命令セットクラスへ削除命令を出す ユーザ用にコンボボックスから削除
OnButton9()	変更ボタン	命令セットクラスへ変更命令を出す ユーザ用にコンボボックスを変更

表 1～ 表 6 に記したように、ユーザインターフェースと命令セットクラスとの結合は、プロパティシートのタブ変更（OnSetActive or OnKillActive）と解析ボタン、追加ボタン（OnButton）で行っている。

4.3 汎用アセンブラのユーザインターフェースの実装

汎用アセンブラはユーザ独自の命令セットに対してアセンブルが可能なツールである。命令セット定義ファイルを読み込み、アセンブリソースプログラムの解析を行う。アセンブリソースプログラムを命令セット情報に従って機械語に変換し、そのファイルを作成する。以下では汎用アセンブラのユーザインターフェースの実装と結合について述べる。

図 8 に示したように、汎用アセンブラのユーザインターフェースは 7 つのボタン、アセンブリソース出力エディタ、ISID 指定エディタ、リストボックスで構成される。

表 7 にウィンドウの表示方法や見せ方を処理するプログラムである MainUIView.cpp で作成したメンバ関数を示す。

表 7：MainUIView.cpp で作成したメンバ関数

メンバ関数	メッセージ	処理内容
OnButton1()	開くボタン	ファイル探索画面を出力し、アセンブリソースを開く
OnButton2()	上書き保存ボタン	上書き保存する
OnButton3()	保存ボタン	保存先探索画面を出力しファイルを保存
OnButton4()	アセンブルボタン	アセンブルを行う
OnButton5()	ISID ボタン	ISID を開く
OnButton6()	汎用シミュレータボタン	汎用シミュレータを開く
OnButton7()	開くボタン	命令セット定義ファイルを開く
OnSize(UINT nType, int cx, int cy)		画面のサイズ、ボタンやエディタの配置を決定
OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)	エディタのカーソルが移動	ステータスバーにエディタのカーソル位置の行番号を表示

ISID と汎用アセンブラのユーザインターフェースの一番の違いは、命令セット定義ファイルを読み込み、命令セットクラスに情報を渡すという点である。アセンブリソース用に大きなエディタを１つ配置しているため、命令セット定義ファイルは出力せずに、パスを表示するという形にした。表示されたパスにあるファイルの中身を文字列に格納し、命令セットクラスへと渡した。

また、出力する機械語は、構文解析の機械語生成メソッドにより翻訳された機械語を受け取りファイル生成する流れとなる。

ISID と汎用シミュレータ起動ボタンは、外部プログラムを実行する SellExecute 関数を用いて実装した。SellExecute 関数で ISID、汎用シミュレータの実行ファイルのパスを指定することで、呼び出すことができる。

5　評価と考察

5.1　評価

実装した ISID と汎用アセンブラを同じ研究室の方に使って頂き、感想を伺った。それらを以下にまとめる。

良かった点

- ISID
  - タブで定義画面が区切られているので理解しやすい
  - 定義が不十分だと警告が出てよい
- 汎用アセンブラ
  - 見やすく分かりやすい

改善すべき点

- ISID
  - 個々の設定でどう定義すればいいか分かりづらい
  - 命令定義でどのフィールドにどの引き数を割り当てるかが分かりづらい
  - 命令セット定義ファイル出力エディタの文字が小さい
  - 定義画面で選択されている部分に対してコメントバーなどでヘルプを表示できれば尚良し
  - フィールド定義は、命令長分の四角を選択して定義できると面白い
- 汎用アセンブラ
  - アセンブル結果をファイル出力前に確認できるといい
  - 画面の大きさは変わるがエディタとリストボックスの仕切りが動かない
  - タブで命令セット定義ファイルの内容も見ることができるようにするとよい

5.2　考察

評価の良かった点を見ると、ISID の命令定義画面をプロパティシートを使ったタブ画面を用いたことで順を追った定義や画面の見やすさが実現できたという点から、設計した際の意図が十分に伝わっている印象を受ける。改善すべき点は大変参考になった。先ず、画面配置である。エディタや文字のサイズの違いひとつで、ユーザにとって使いやすいものになる。また、複数のウィンドウ（MDI：Multi Document Interface）形式を採ることで、命令セット定義画面を単独のウィンドウで示し、図の拡大や縮小が出来るようにすればよりわかりやすくなる。今回、ISID、汎用アセンブラの実装はできたが、ユーザにとって使いやすく分かりやすいものにするには、まだまだ改善の余地があることが分かった。

6　おわりに

本研究では、従来のハード/ソフト・コラーニングシステムに命令セットを独自に定義できる環境を整備するため、独自の命令セットに対応できる汎用アセンブラと、その中で用いる命令セット定義ファイルの作成支援ツールである、ISID のユーザインターフェースの設計と実装、命令定義ファイル作成のためのさまざまなメソッドが用意された命令セットクラス、構文解析クラスとの結合を行った。

開発言語は Visual C++を用い、ユーザインターフェースは SDI（Single Document Interface）形式を採用した単一の窓構成となっている。アプリケーションの見せ方を決める View クラス、命令セットを定義するプロパティシートを中心にプログラミングを行い、処理クラスである命令セットクラス、ファイルを保存する Document クラス、SDI のフレームを構成する MainFrame クラスなどとデータをやり取りしながら実装した。

ユーザにとって解りやすく、使いやすいものを作ることを目標としていたため、ISID は命令セットクラスとの結合箇所が多く複雑で、最初は多くのエラーが発生していたが、可能な限りバグ取りをすることができた。最後に、実装した ISID と汎用アセンブラを同じ研究室の方に使って頂き、貴重な意見を頂戴した。その中でも ISID の命令定義、擬似命令定義が複雑な命令セットだと面倒という意見が多かった。汎用性を高めるためすべてユーザが定義できるという形をとっていたが、状況に応じて今後改善が望まれる。

汎用アセンブラ、汎用シミュレータの実装により命令セット定義可能なハード/ソフト・コラーニングシステムは実現できたと思う。これにより、学習環境の整備が行われ、今後、命令セットアーキテクチャ、マイクロアーキテクチャの理解を目標とした具体的な学習フローの提案を行い、学部生を対象とした実験授業に取り入れられることを願う。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導を頂きました、山崎勝弘教授、小柳滋教授には深く感謝いたします。また、本研究に関して貴重なご意見をいただきました中村浩一郎氏、難波翔一朗氏、共同研究者の富樫望氏、西田範行氏、及び命令セット定義ツールと汎用アセンブラの評価に協力していただき、様々な面で貴重な助言を下さった研究室の皆様に深く感謝いたします。

参考文献

[1] 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] 組み込みネット：http://www.kumikomi.net/article/report/2001/16swest/01.html、2006.2

[4] 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

[5] 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

[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] 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.

[8] 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.

[9] 山本信雄　著：プログラマ養成入門講座 VisualC++１、２、３、翔泳社、1999．8

[10] 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.

[11] 浦昭二、原田賢一　著：C 入門、培風館、1994.6

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.