修士論文

ハード/ソフト協調学習のための
コンパイラ学習システム設計と実現

氏　　名　：　井手　純一
学籍番号　：　６１６２０８００１２-0
指導教員　：　山崎　勝弘　教授

提 出 日　：　2010 年 2 月 15 日

立命館大学大学院　理工学研究科　創造理工学専攻

内容梗概

本研究では、プロセッサ周辺のコンピュータ構成知識の習得、ハードウェアとソフトウェアのトレードオフを理解した人材を育成することを目的に、コンパイラ学習システムの設計を行い、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へ実装した。

本論文では、プロセッサ設計によるハード/ソフト協調学習システムの評価と、その評価から考案したコンパイラ学習システムを構築するために設計した MONI コンパイラの構成と、ハード/ソフト学習システム上での学習方法について述べる。

MONI コンパイラは、本研究室で MIPS のサブセットとして定義した教育用命令セットに対応したコンパイラである。単にコンパイラの役割を果たすだけでなく、学習者がコンパイラの仕組みを理解し、これまでのハード/ソフト協調学習システムの学習効果を向上させるために設計した。MONI コンパイラは字句解析部、構文解析部、変数を登録する変数表、及びコード生成部から構成される。字句解析、構文解析部は、コンパイラ・コンパイラである Flex と Bison を用いて生成している。MONI コンパイラは学習者がコンパイラの構成を理解しやすいように、アセンブリコードを生成するだけなく、それぞれトークン列、構文情報、変数登録結果、コード生成部において中間表現から MONI アセンブリコードに変換する処理を一つ一つ確認できるように構築した。

今後、本研究で設計した MONI コンパイラを用いて、学習者がアセンブリ言語の理解促進、プロセッサアーキテクチャの詳細な理解、そしてプロセッサ周辺のコンピュータの構成をハード/ソフト協調学習システムを通して今まで以上の学習効果が得られることを期待している。

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により、LSI の小型化、軽量化と高速化、そして消費電力化が可能となった。携帯電話、自動車、カーナビゲーションシステム、炊飯器、信号機、エレベーター、自動販売機、デジタルカメラ、テレビ、ゲーム機、複写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る。これら組み込み機器の普及は、これからも広がっていくことが確実視されている。そして要求される仕様は年々大規模かつ複雑になり、実装的には小型、低消費電力化が進み、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。このように、高集積システムＬＳＩ技術の進化の中、システム LSI へ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。

半導体製造技術の進歩によって、大規模で複雑なシステムが 1 つのチップ上に構成できるようになったこと、つまり LSI の設計と検証がシステム全体の設計と検証と等価になった。また、組み込み機器のライフサイクルが短くなり，開発期間を短くすることがますます重要となっていることにより、ハード/ソフト協調設計が開発期間短縮に大きく影響する。このような近年の LSI 開発技術はハードとソフトに密接な関係があり、ハードウェアとソフトウェアの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。

以上の背景から、大学の教育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。大学教育の中でも九州工業大学の KITE、広島市立大学の City-1 など、教育用マイクロプロセッサを用いたシステムの開発が進められている[1]-[2]。そこで本研究室では、ハード/ソフト協調学習システム（以下 HSCS）を考案し、開発を進めてきた[4]-[22]。HSCS とは、プロセッサを通してハードウェアとソフトウェアにおける両分野の学習を進めていくことを目的としたシステムである。現在ハード/ソフト協調学習システムを利用し、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を進めている。実際に HSCS の利用を 4 回生の教育課題、または卒業研究の一つとして、アセンブリプログラミングの学習、設計したプロセッサを実機上で検証を行うことでシステムの評価を進めてきた[14][17][18]。評価を進める中で、プロセッサの設計規模と設計したプロセッサを検証する際の FPGA ボードの規模が一つの改善点として挙げられた。近年のプロセッサは、パイプラインやスーパースカラを基本とし、マルチコアプロセッサが主流となっている。そんな中で HSCS でも大規模なプロセッサ設計学習を行ってもらうにはこれまで対応していた FPGA ボードでは FPGA の容量が足りなかった。そこで HSCS のプロセッサ設計支援ツールのプロセッサデバッガを一部追加することで、規模の大きい FPGA ボードでも検証することを可能としている。

本研究では、この HSCS の新たな学習分野に対し、プロセッサ周辺のコンピュータの構成の一つとして、コンパイラ学習システムの開発を行った。学習者にコンパイラの学習を

行ってもらう内容は、コンパイラ全てを学習してもらうのではなく、一部を学習者自信が設計することで全体の構成と役割を理解してもらうことを目的としている。コンパイラ学習システムを実装する意義として、プロセッサ周辺のコンピュータの構成を学習してもらうだけでなく、これまでの HSCS の学習効率向上として 2 つの大きな役割がある。1 つは、コンパイラを使用することによって、HSCS のソフトウェア学習内容を拡大させることである。これまでのソフトウェア学習ではアセンブリ言語のみの学習内容であったが、C 言語からの学習も行えることになる。細かな論理演算子などが使えないアセンブリ言語から始めるより、比較的容易な論理演算子や、一つの命令で複数のアセンブリ命令を扱える C 言語から学習を進めることで、アセンブリ言語のより深い理解をすることができる。2 つ目は、これまで以上のアセンブリ言語の理解がハードウェア学習にも繋がると考える。複雑なアセンブリプログラミングを行うには、対応する命令セットのプロセッサアーキテクチャを詳細に知る必要がある。HSCS では、学習初心者にパイプラインやスーパースカラの複雑なアーキテクチャを意識させることは難しいものであった。今後このコンパイラ学習システムの実装によって、より詳細にアセンブリ言語を理解し、プロセッサアーキテクチャを意識したアセンブリプログラミングを行ってもらうことができる。

本論文では、第 2 章でハード/ソフト協調学習システムの概要と学習内容について説明する。そして本研究で開発したコンパイラ学習システムの設計思想と構成を第 3 章で、システム実現方法を第 4 章で述べる。さらに、本研究で行ったコンパイラ学習システムの設計と HSCS に対する評価を第 5 章で述べ、第 6 章で全体のまとめとする。

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/ソフト協調学習システムとは、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ために考案されたシステムである。システムの構成は、ソフトウェアとハードウェアの両分野を学習するフローからなっており、ソフトウェアの学習フローではアセンブリプログラミングの学習を進めるため、アセンブリ命令毎にプロセッサのデータパスを確認できる仮想アーキテクチャが用意されている。ハードウェアの学習フローでは学習者が独自に命令セットを定義し、FPGA コンピュータ上での検証まで行えるようにプロセッサ設計支援ツールが用意されている。

(1) ソフトウェア学習

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を理解し、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。

(2) ハードウェア学習

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。

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、命令セットアセンブラ、命令セット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。

2.2 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命令セットアセンブラと命令セットシミュ

レータを使用する。また命令セット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。命令セット設計の流れを図 2 に示す。

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。プロセッサデバッガの構成を図 3 に、デバッグコマンドを表 1 に示す。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。

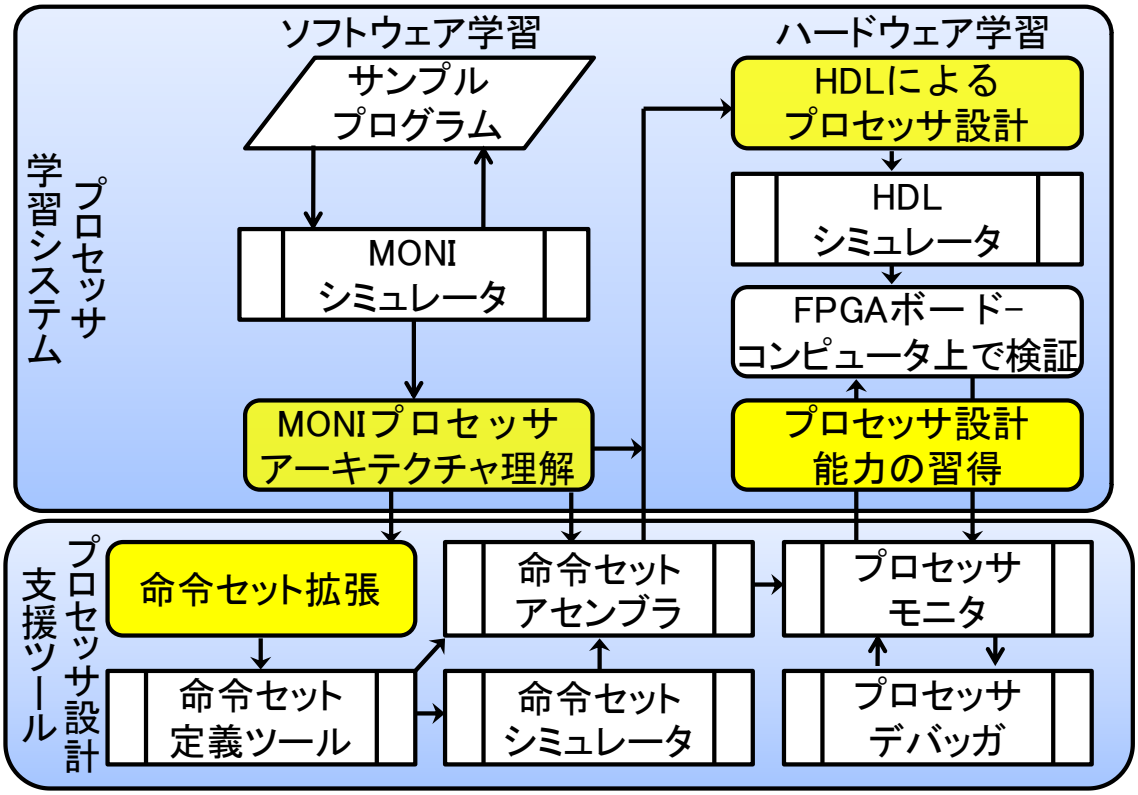

図 1：ハード/ソフト協調学習システム

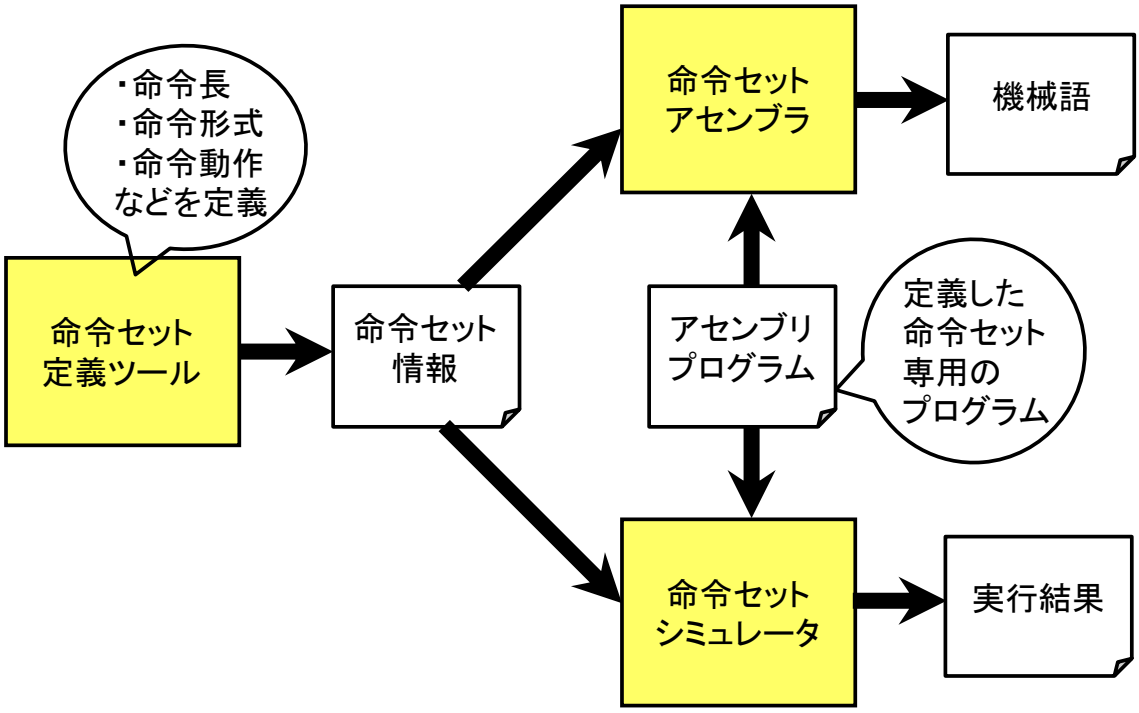

図 2：命令セット設計の流れ

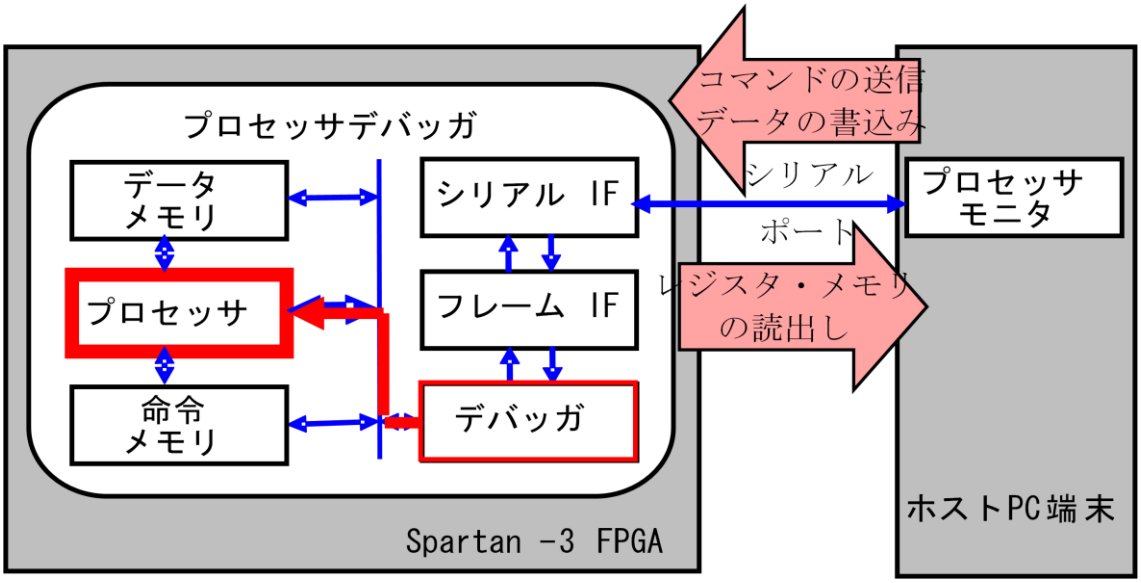

図 3：プロセッサデバッガの構成

表 1：デバッグコマンド一覧

コマンド	ターゲット	意味
write	dm/im/rf	メモリ･レジスタの書き込み
read	dm/im/rf/pc	メモリ･レジスタの読み出し
save	dm/im/rf/pc/bp	メモリ･レジスタ内容を保存
load	dm/im/rf/bp	ファイルからロード
set	pc/bp	PC･ブレイクポイント設定
del	bp	ブレイクポイント削除
list/init	dm/im/rf/pc/bp	メモリ･レジスタの表示/初期化
run all		通常実行
run clk N		Nクロック実行
run bp		ブレイク実行

表 1 において､dm と im はデータメモリと命令メモリを表し､pc はプログラムカウンタ、rf はレジスタファイルを表す。また bp はブレイクポイントの略である。デバッグコマンドには、表 1 以外にプロセッサを停止する halt、プロセッサをリセットする rst、デバックコマンドの内容を表示する help、およびプロセッサモニタを終了させる exit のコマンドが用意されている。

2.3　システムの評価

これまで主に 4 回生の卒業研究として、HSCS を実際に利用して評価を進めてきた。これまで HSCS を利用した 3 人の学習時間を表 2 に示す。

表 2：HSCS を利用した学習時間

学習年度	2006		2007		2008
プロセッサ	MONI		SARIS		PSCSF
サイクル	シングル	マルチ	シングル	マルチ	シングル
ソフトウェア学習	25		18		28
ハードウェア学習	183		230		122

表 2 は、2006 年から 2008 年にかけて HSCS を利用してプロセッサ設計学習を行った 3 人の学習時間を表している。MONI、SARIS 設計者は 2 つのプロセッサアーキテクチャ設計しており、また SARIS、PSCSF 設計者は独自に命令セットを定義してプロセッサ設計を行っている。

この学習時間からわかるように、ソフトウェア学習とハードウェア学習時間では大きな差がある。HSCS では確実に学習時間や学習課題を決定しているわけでもなく、ソフトウェア学習ではアセンブリ言語の学習、ハードウェア学習ではプロセッサ設計を行うので学習時間に差がでるのは当然である。しかし、ここで 3 人の学習者が設計したプロセッサアーキテクチャについて着目した。3 人全員がシングルのみ、またはマルチサイクルの設計学習を行っている。現在社会ではパイプラインやスーパースカラのアーキテクチャが主流であり、その段階の学習までは進むことが難しい現状である。この課題を解決するため、ソフトウェア学習に主点を置いた。学習者はハードウェア学習に入る前に、ソフトウェア学習でアセンブリプログラミングと、MONI シミュレータを用いてアセンブリプログラムのデバッグと同時にプロセッサアーキテクチャのデータパスを理解する。MONI シミュレータにはパイプラインやスーパースカラのアーキテクチャも確認できるように用意されているが、アセンブリ言語を学習しながらハザードやフォワーディングの理解は非常に困難なものであると考える。実際 3 人の学習者はアセンブリプログラミングに重点を置き、シングルのみを理解をしてハードウェア学習に進んでいた。そこで、ソフトウェア学習においてアセンブリ言語の理解とプロセッサアーキテクチャの理解をより近づけることができないかと考えた。シングルサイクルはアセンブリ命令 1 つが 1 クロックであり、それぞれの命令形式に添ったプロセッサの動きを行うだけだが、パイプラインやスーパースカラはハザードを避けるため、アーキテクチャを考慮したプログラミングを行う必要がある。この部分を理解する学習時間を増やすため、始めの段階でアセンブリ言語をもっと理解させて集中して複雑なアーキテクチャの動きを確認してもらうことが必要である。そこで考案したものがコンパイラ環境を HSCS に実装することである。コンパイラ環境を用意するために、アセンブリ言語の初心者でも C ソースコードとアセンブリコードを比較することでアセンブリ言語の理解促進になると考えた。そして、もっとソフトウェア学習の段階で複雑なプロセッサアーキテクチャを理解させる学習時間を増加させるべきであると考え、本研究のコンパイラ学習システムの設計を提案した。

3. コンパイラ学習システムの設計

3.1 設計思想

これまで HSCS では、プロセッサを重点とした学習内容であった。しかし、ハードウェアとソフトウェアのトレードオフをより詳細に理解してもらうためには、もっと幅広くコンピュータの構成を理解する必要がある。そこでプロセッサ周辺の一つであるコンパイラも含めた学習内容にできるように、コンパイラ学習システムを提案した[22]。

コンパイラの対象アーキテクチャを MONI とし、MONI コンパイラの設計を検討した。検討した理由は、これまで HSCS を利用して学習を進めてきた学習者のプログラミング能力に着目したことにある。アセンブリ言語は対象とするプロセッサが実際に実行するコードや、また細かくアーキテクチャを理解する必要があり、プログラミング知識が少ない学生にとってアセンブリプログラミングを行うことは困難であり、学習時間が増加すると考えられる。一方、C 言語は論理演算やアドレス演算が比較的容易であり、1 つの命令が複数のアセンブリ命令に対応しているので、初心者でも理解しやすいプログラミング言語といえる。以上の点から、MONI コンパイラを開発することで以下 4 つの学習効果が期待できると考える。

(1) コンパイラの仕組みの理解

C ソースから目的語を生成するだけでは、コンパイラの構成は理解できない。そこで MONI コンパイラは、字句解析結果によるトークン列、構文解析結果による構文情報など、一つ一つの処理結果を確認しながらコンパイルを進めてもらう。そして設計ファイルを確認してもらうことでコンパイラの構成を理解してもらう。また、学習者が独自の命令セットに対応したコンパイラを用意できるように、MONI コンパイラではコード生成部によって命令セットが対応できるように設計している。学習者が実際にコンパイラを開発するには、多大な学習時間が必要と考えられる。そこで HSCS 学習では、コンパイラだけの学習となってしまわないように、コンパイラの一部であるコード生成部を設計してもらう。

(2) アセンブリプログラミングの理解の促進

MONI コンパイラを開発することにより、C ソースコードと MONI アセンブリコードを比較することで、学習者のアセンブリ言語の理解を早め、複雑なアセンブリプログラムを作成できるようになることが期待できる。

(3) 特徴ある命令セット・アーキテクチャの設計

アセンブリプログラミングをより深く理解することは、命令セットやプロセッサアーキテクチャの理解向上にも繋がると考えられる。MONI コンパイラを使用することでアセンブリ言語の理解を今まで以上に深めることは、MONI 命令セットと MONI アーキテクチャ

をより詳細に理解することであり、ハードウェア学習の際にも複雑な命令セットを考えることができ、学習者自身の特徴あるプロセッサ設計を行えることが期待できる。

(4) アプリケーションに特化したプロセッサ設計

HSCS にコンパイラ学習システムを取り組みことで、今後学習者自身もコンパイラを実際に設計し、言語処理系ソフトウェアの学習も行ってもらう。学習者は MONI コンパイラを使用し、実際にコンパイラを設計することで、C 言語の学習を活かしてアプリケーションを考え、特定のアプリケーションに対して命令セット・アーキテクチャを特化した専用プロセッサの設計学習ができるものと考えている。

3.2　MONI コンパイラの構成

図 4 に MONI コンパイラの構成を示す。

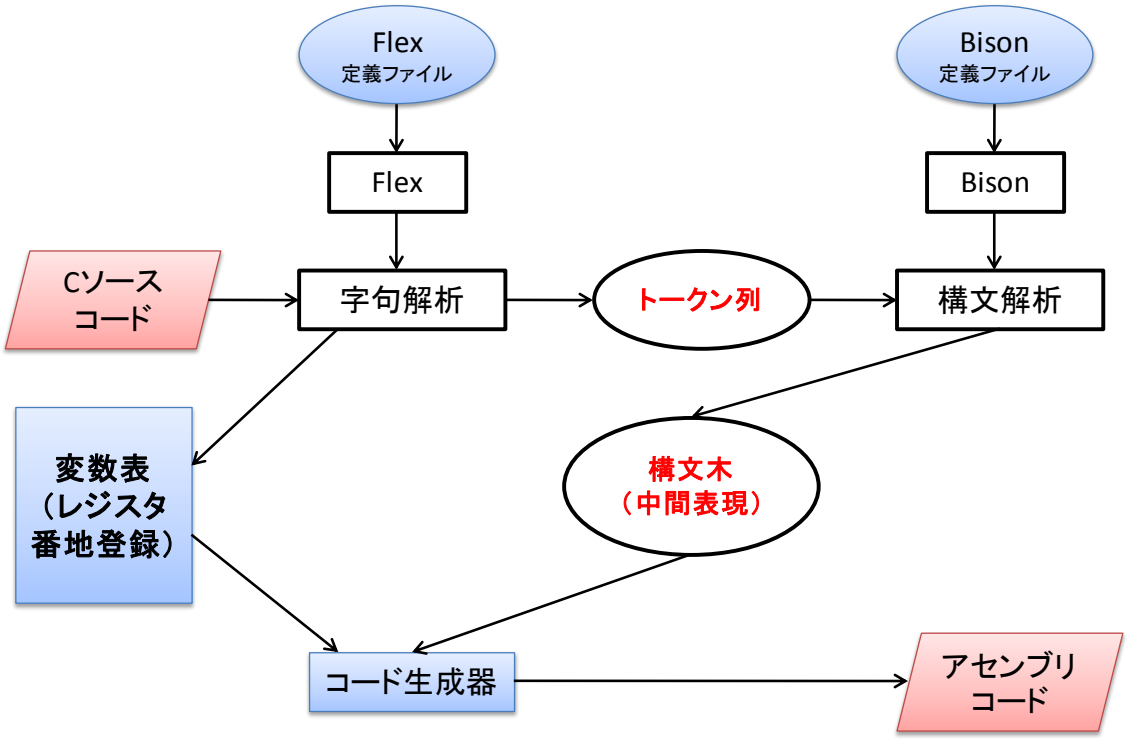

図 4：MONI コンパイラの構成

上図のように、ソースプログラムは、

① 字句解析
② 変数表登録
③ 構文解析
④ コード生成

の順に処理を進め、MONI アセンブリコードを生成する。図 4 のように、学習者が用意した C ソースコードを、Flex を用いて生成した字句解析プログラムに通し、トークン列を生成する。また同時に変数をテーブル表の配列に格納しておく。次に、生成されたトークン列を、Bison を用いて生成した構文解析プログラムに通し構文木を生成する。この構文解析された結果を中間表現とし、コード生成部により初めに変数表に登録した変数名の配列場所をレジスタ番号として受け取り、MONI アセンブリコードを生成する。

3.2.1　字句解析部

字句解析は、ソースコードの文字列の並びをトークンの並びに変換することである。トークンとは意味を持つコードの最少単位であり、キーワード、識別子、シンボル名がトークンである。コンパイラは、ソースファイル上の区切りなどより、1 文字ずつ調べていく必要がある。本研究ではコンパイラ・コンパイラである lex の GNU 版の Flex を用いて自動生成している。

3.2.2　構文解析部

構文解析は、字句解析されたトークン列を受け取り、解析することでプログラムの構造を明らかにし、構文木を生成する。通常のコンパイラでは、この構文木だけでは解析不十分である解析を、意味解析によって構文木と意味規則の対応をとり、中間表現を得る。意味規則とは、例として変数名を実数型と宣言したら、その変数は実数型としてしか使えないことである。本研究では、コンパイラの目的言語を MONI としており、MONI は整数しか扱えないので、細かい意味解析を省略し、構文解析結果を中間表現として扱う。また、字句解析と同様コンパイラ・コンパイラである yacc の GNU 版である Bison を用いて自動生成している。

3.2.3　変数表登録

変数表は、C ソースで定義された変数を配列に格納することで、その要素番号を MONI のレジスタ番号とする。変数表にはハッシュ法を使用している。

3.2.4　コード生成部

中間言語から目的言語を生成する部分である。本研究では構文解析部から生成された解析木と字句解析時に登録した変数名をレジスタ番号とし、MONI アセンブリコードの生成を役割とする。

3.3 対象とする言語

教育用学習用言語として、C のサブセットとなる MONIC を考案した。表 3 に、MONIC の言語仕様を示す。MONI では、自由に記述できる C プログラムに対して、予約語や変数の宣言など全てを実行することは不可能なので、様々な制約を設けた。また、コンパイラを HSCS に実装する目的は汎用コンパイラとしての機能ではなく、学習者がアセンブリ言語の理解促進とコンパイラの構成を理解することである。

表 3：MONIC の言語仕様

項目	説明
名前	英数字、変数名やラベル
整数	10進数の整数
式	変数、整数、算術論理演算子（＋、－、＝等）からなる式。
予約語	if,while,do,for,goto,case

データ型は int 形のみサポートする。構造体、共用体はサポートしない仕様となる。

4. コンパイラ学習システムの実現

4.1 Flex を用いた字句解析部の生成

Flex を用いた字句解析プログラムの生成フローを図 5 に示す。

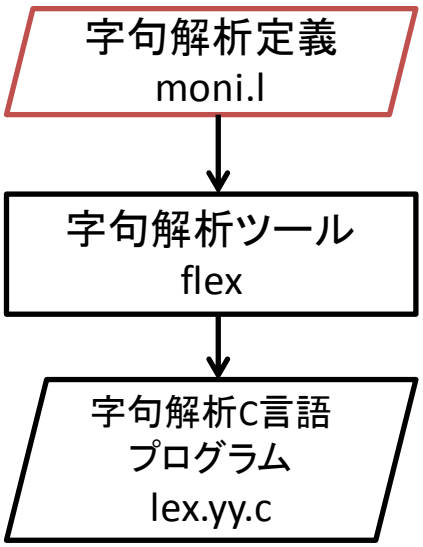

図 5：字句解析プログラムの生成フロー

生成結果である字句解析プログラムは lex.yy.c である。このプログラムを生成するために必要となる情報が、字句解析定義ファイル moni.l である。字句解析定義ルールを図 6 に示す。

```
%{
型や定数の定義
#includeや#define
%}
下の定義部で使うマクロ定義
%%
定義ルール
入力される文字列　{return トークン}
etc ・・・
%%
任意のCプログラム
main関数の定義など
```

図 6：字句解析定義ルール

図 6 のように、%%で仕切られた 3 つのセクションで記述される。上図の中で、C コードが記述できる場所には自由にコメントすることも可能である。また、プログラマは 2 つの方法で、スキャナ(構文解析プログラム)に直接 C コードを含めることができる。第 1 の方法は初めのセクションに C コードを含めることであり。第 2 の方法は最後のセクションにコードを含める方法である。どちらも lex.yy.c(構文解析プログラム)にコピーされるので正当なコードでなければならない。定義ルールは入力とならC ソースコードからトークンを生成する情報を記述する。

定義部における moni.l の一部を図 7 に示す。

```
%%
=               {return(EQ);}
>               {return(LC);}
while           {yylval.n=++n;return(WHILE);}
{variable}   {yylval.s= IDentry(yytext, yyleng);
                                        return(VARIABLE); }
[a-zA-Z][a-zA-Z0-9]*
                {yylval.s=strdup(yytext);return(NAME);}
                                ▪
                                ▪
%%
```

図 7：定義部における moni.l ファイルの一部

図 7 には、入力された文字列が‘＝’、または‘＞’の時は、構文解析プログラムに対しそれぞれ‘EQ’、‘LC’というトークンを返すことを示している。‘while’が入力されると、ポインタ型の yylval.n に整数を格納し、トークン‘WHILE’を返す。yylval.n に格納した値は、分岐先のラベルとして後に使用する。{variable}は、入力される変数を第 1 セクションでマクロ定義している。variable が入力として入ってくると、ポインタ型の yylval.s に変数を代入し、変数表の関数 IDentry にその代入した値を返す。そして構文解析にトークン‘VARIABLE’に渡す。最後の‘[a-zA-Z][a-zA-Z0-9]’は、英数字を表す。英数字が入力されれば、字句解析部は英数字を yylval.s に代入し、yytext に英数字をコピーして、トークン‘NAME’を構文解析に渡す。

トークン列の生成結果例を図 8 に示す。

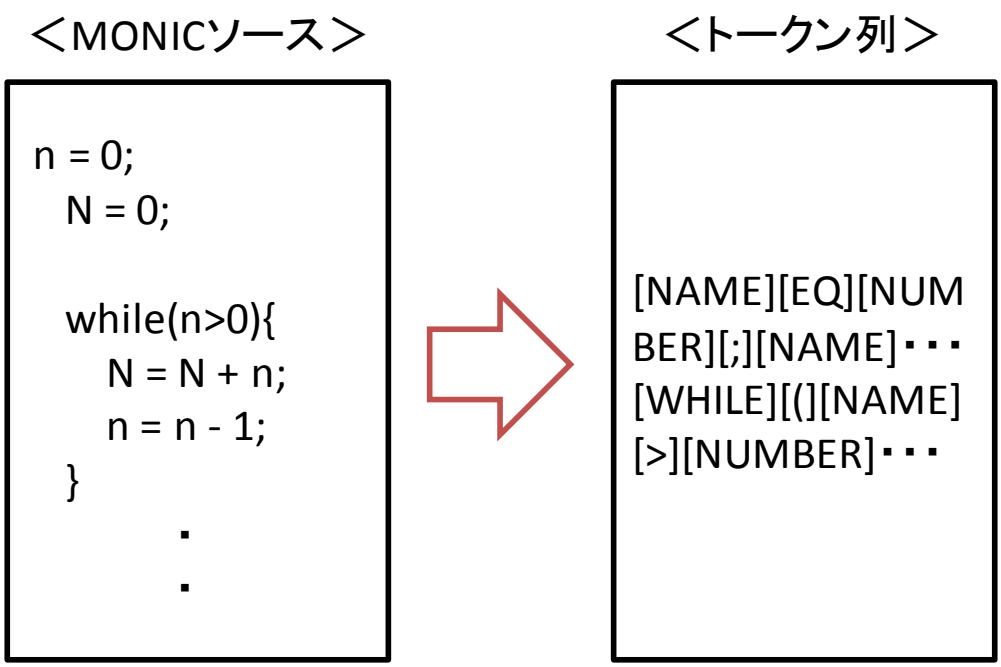

図 8：トークン列生成結果

図 8 では、MONIC ソースから入力される順に、‘n’が[NAME]、‘=’が[EQ]、‘0’が[NUMBER]としてトークンに変換され出力されている。‘while’以降も同様に字句解析部で定義したルールに従いトークンを生成している。

4.2　Bison を用いた構文解析部の生成

Bison を用いた構文解析プログラムの生成フローを図 9 に示す。

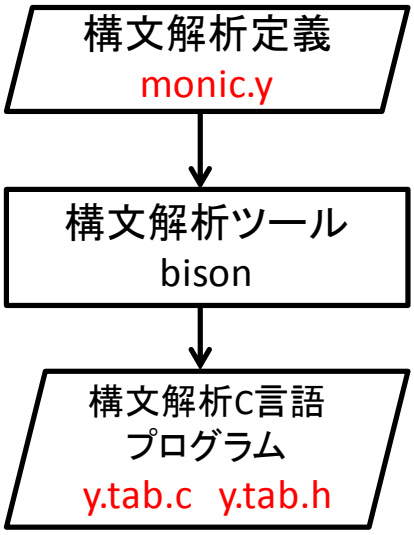

図　9：構文解析プログラムの生成フロー

Flex を用いた字句解析プログラム生成の流れと同様に、構文解析定義情報の monic.y を Bison に与えることで、構文解析プログラムの y.tab.c と y.tab.h を生成する。y.tab.h には、ユーザーが割り当てたトークン番号を、宣言したトークン名に対応させる#define 文を含んでいる。構文解析定義ルールを、図 10 に示す。

```
%{
型や定数の定義
#includeや#define
%}
下の定義部で使うマクロ定義
%%
定義ルール
入力トークン:処理
etc・・・
%%
任意のCプログラム
main関数の定義など
```

図　10：構文解析定義ルール

図 6 と図 10 を比較すればわかるように、構文解析ルールも字句解析定義ルール同様に%%で仕切られた 3 つのセクションで構成される。コメントや C コードの記述においては字句解析定義と同様である。

定義ルールは、入力されたトークンに対し、それぞれ記述された処理を行う。Bison は、BNF(バッカス・ナウア記法)に似た構文規則に基づいてパーサを生成する。定義部の moni.y の一部を図 11 に示す。

```
%%
while:      WHILE {printf(" %02dT:¥n",$1);}
            '(' expr ')'   {printf("¥tSEP %02dF¥n",$1);}
           '{' expr ';' expr ';' '}'
                        {printf("¥tJUMP %02dT¥n %02dF:¥n",$1,$1);}
      ;
expr : VARIABLE      {printf("¥t$%s",$1);}
      | expr > expr   {printf("¥t>");}
                                          ▪
                                          ▪
%%
```

図 11：定義部における moni.y の一部

図 11 には‘while’と‘expr’についての定義ルールを示している。expr は、変数や整数における式、論理演算子の定義である。まず‘while’の処理について説明する。上図の‘while’における記述は、MONIC ソースコードで書かれる while 文「while （条件）（式）」の形にマッチする。トークン‘WHILE’を受け取れば、printf 関数によってループ先のラベルを出力する。次に（条件）に値する expr が入力されれば、その（条件）に対応する中間表現の命令と、(条件) が偽の場合分岐するラベルが生成される。次に (式) に対する expr が入力され、expr の定義による処理が施される。(式) に対する処理が終われば、ループするために無条件分岐の JUMP を生成する。

‘expr’については、expr の中が字句解析部で定義した変数、すなわち VARIABLE であれば、字句解析部で yytext に登録した変数の値を出力する。また上図の‘expr’部の 2 行目の expr ＞ expr は、図 8 の MONIC ソースコードにおける while の条件（n>0）に対応し、そのまま‘＞’を出力するように記述している。

構文解析結果の例を図 12 に示す。

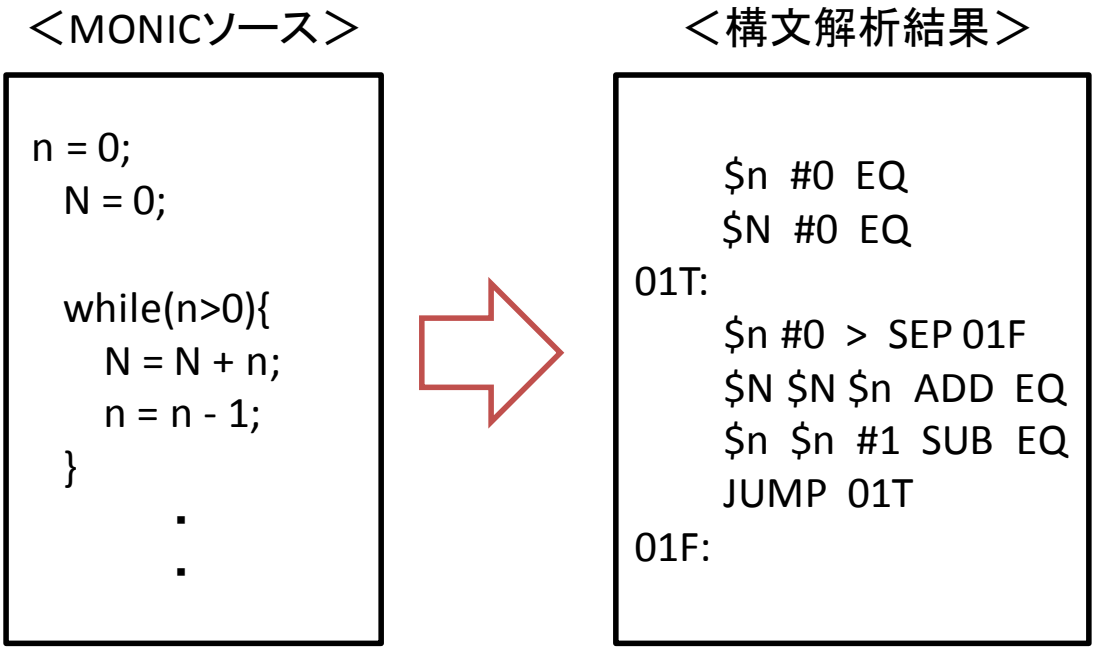

図 12：構文木生成結果

図 12 の結果のように、2 つの例をあげると<n = 0;>が<$n #0 EQ>を生成し、<N = N + n>が<$N $N $n ADD EQ>を生成している。このように後置記法に似た構文情報を生成する。

4.3　変数表の登録

変数表にはハッシュ法を用いた。ハッシュ法にはいくつか技法があるが、本研究における変数表はチェイン方を用いた。ハッシュ表を H、与えられた文字列に対するハッシュ値を k とする時、その文字列を持つポインタを H[k]に入れる。与えられる異なる文字列が、同じハッシュ値に入ることを衝突という。チェイン方では、そのまま同じ位置に格納しようとするが、代わりに連結リストを用意し、データを連結する。このリストが衝突チェインである。変数表の構成を図 13 に示す。

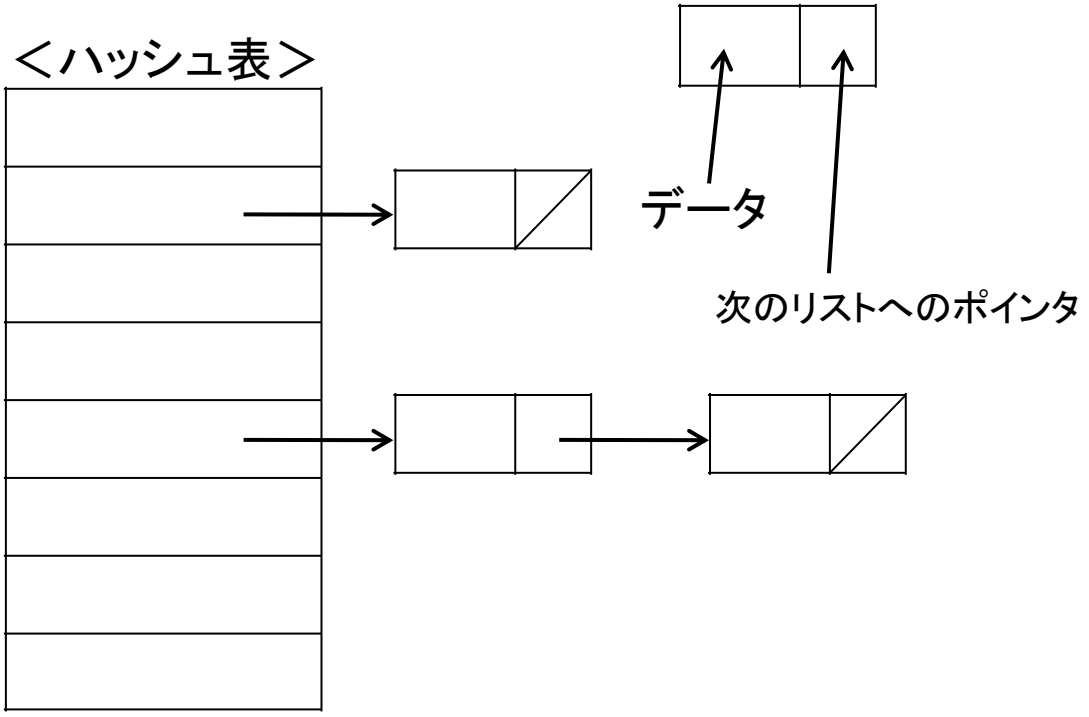

図 13：変数表の構成

図 13 のように、データを保存する領域はハッシュ表とは別の領域に確保されている。本研究では、MONI アーキテクチャの汎用レジスタの数(8 つ)とハッシュ表を合わせている。

このハッシュ表を用いて C ソースコードの変数を変数表に登録し、その登録された変数の要素番号をコード生成部に渡すことでレジスタ番号を決定する。

4.4　コード生成部の生成

コード生成部の役割は、構文解析から生成された中間表現と、変数表に登録された要素番号を用いて MONI アセンブリコードの生成を行う役割を果たす。中間表現から MONI アセンブリコード生成例を図 14 に、生成結果を図 15 に示す。

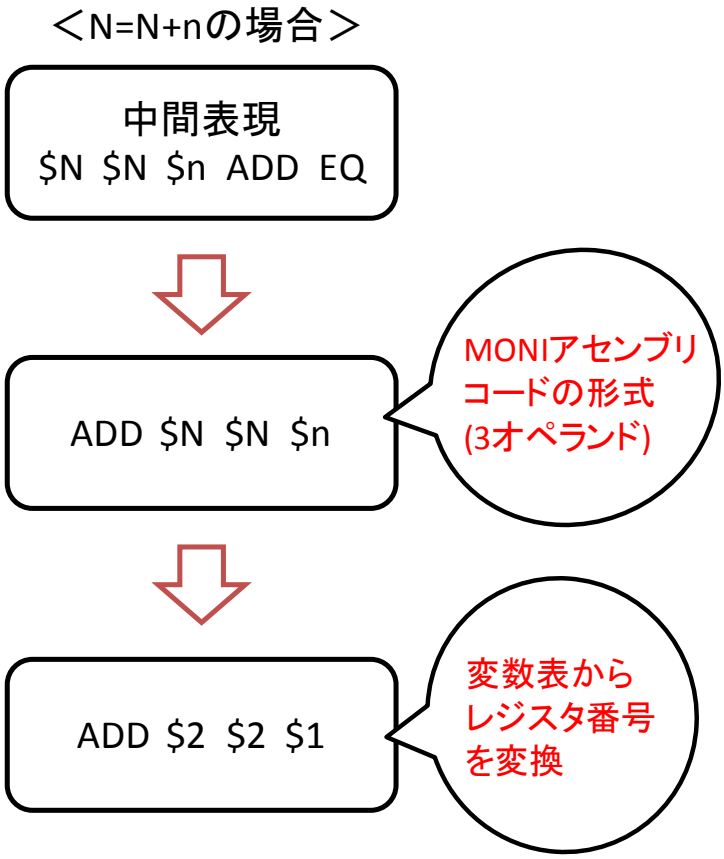

図 14：＜N=N+n>のコード生成例

図 14 のように、構文解析結果を本研究では中間表現として扱う。コード生成部で一度 MONI アセンブリコードの形に直し、変数表に登録した変数の値とレジスタ番号の変換を行うことで MONI アセンブリコードを生成している。中間表現を図 14 の真ん中の形式にしなかったのは、学習者がコード生成部を設計する際、学習者自信の命令セットが 2 オペランドの場合も考慮し、学習者にとってより自由に設計できるように本研究では構文情報をそのままに置いている。

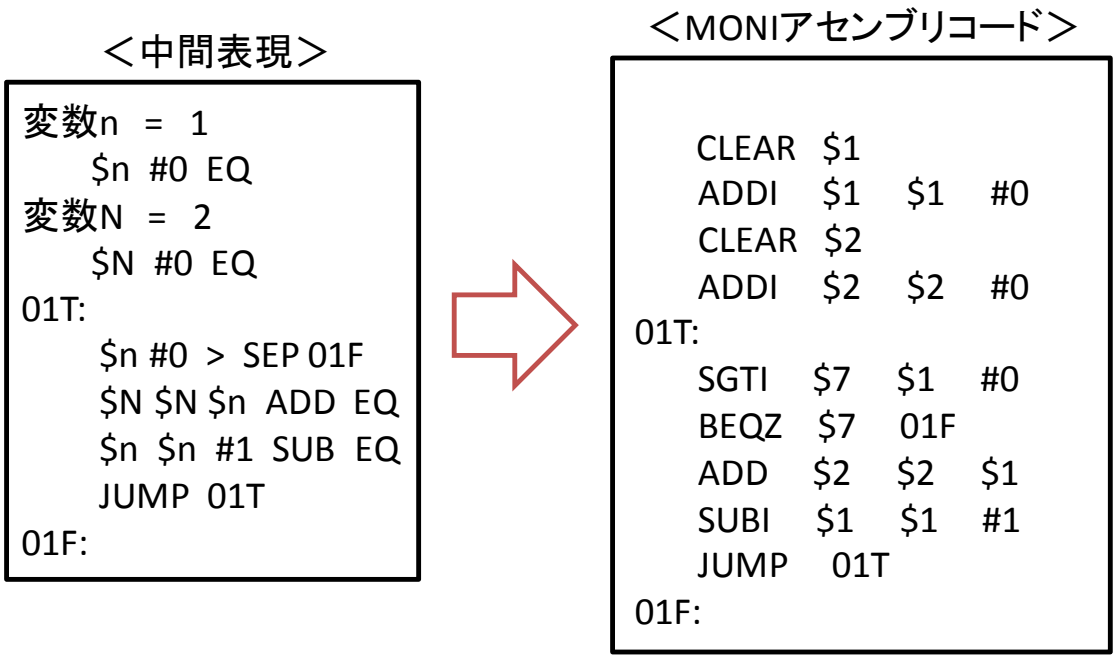

図 15：MONI アセンブリコード生成結果

図 15 は、図左の中間表現から MONI アセンブリコードを生成したものである。ここで<$n #0 > SEP 01F>に着目すると、生成コードは<SGTI $7 $1 #0>と<BEQZ $7 01F>となる。<SGTI $7 $1 #0>は MONI の命令で汎用レジスタ$1 が#0 より大きいなら$7 に 1 を代入する。<BNEZ $7 01F>は、汎用レジスタ$7 が 0 でない場合 01F に分岐する。この時に使用する汎用レジスタ$7 はコード生成部で初めから用意しており、分岐時の条件で使用するレジスタを$7 に統一している。

MONI には、C 言語の算術演算子における乗算、除算、剰余算（*、/、%）に対応する命令が用意されていない。これら演算子が入力された場合はコード生成部で加減算命令を用いてループ処理をさせるアセンブリコードを生成させる。乗算におけるコード生成過程を図 16 に示す。

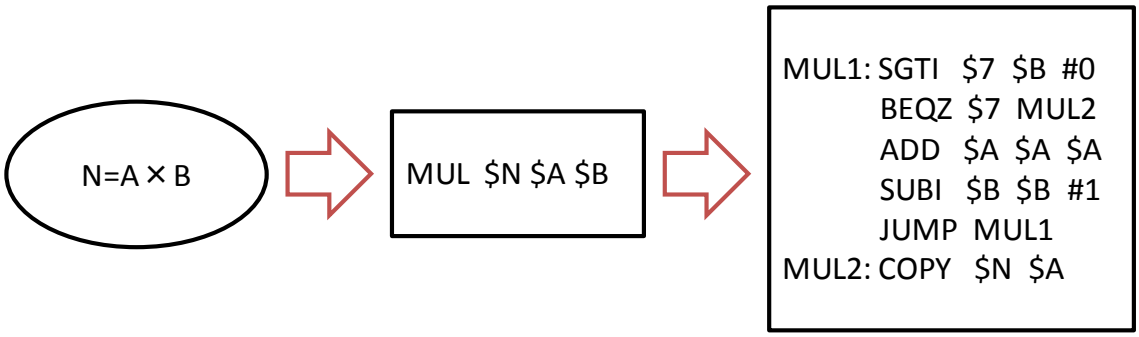

図 16：乗算におけるコード生成過程

＜N=A×B＞の演算式が入力されると‘×’を字句解析部でトークン‘MUL’とし、MULというアセンブリ命令があると考え中間言語を生成する。実際には MONI 命令セットに‘MUL’という命令は無いので、加算、減算、分岐命令を用いたアセンブリコードを生成させる。

5. コンパイラ学習方法とシステムの評価

5.1 学習方法

3 章の設計思想で述べたように、本研究は 4 つの学習効果を促す役割を持つ。HSCS との学習と別々に学習を進めるのではなく、これまでの HSCS の学習フローにおけるソフトウェア学習と同時にコンパイラの学習を進めていく。その学習フローを図 17 に示す。

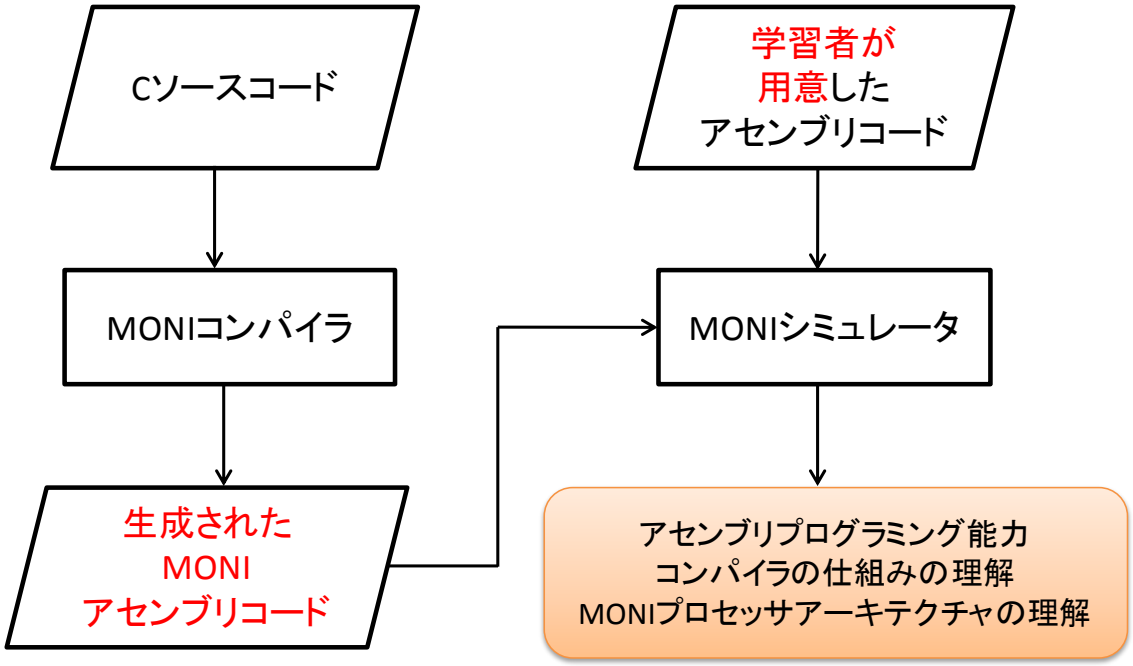

図 17：コンパイラ学習を含んだ HSCS ソフトウェア学習フロー

これまでの学習者は、まず学習者自信でアセンブリプログラミングから初め、MONI シミュレータを用いることでアセンブリ言語のコーディング能力の習得とプロセッサアーキテクチャの理解を行ってきた。本研究のコンパイラ学習システムを実現することで、今後学習者は MONIC の使用に基づいた C プログラミングから学習を進めてもらう。今後の学習の流れは、

① C プログラミング
② MONI コンパイラの使用
③ MONI コンパイラから生成されたアセンブリコードを用いて、MONI シミュレータの使用とアセンブリコードの最適化
④ 学習者自信のアセンブリプログラミング
⑤ MONI シミュレータの使用
⑥ プロセッサアーキテクチャの理解

以上 6 つのステップで学習を進めていく。

第 1 ステップの C プログラミングにおいて、学習者の基礎能力に着目する。これまではプログラミング初心者でもアセンブリ言語からの学習スタートとなっていた。しかしアセンブリ言語は対応している命令セットのプロセッサアーキテクチャを詳細に理解する言語であり、プログラミング初心者ではプログラミングを行っても最適化など考えるまでの学習には行かない。そこで比較的容易な論理演算やアドレス演算を用いて記述できる C プログラミングから始めることで、プログラミング初心者にとっても学習者自信の手書きの C ソースコードと生成されるアセンブリコードを比較できる。また、プログラミング初心者ではなくともこれまで以上のアセンブリ言語のイメージを持つことが可能となる。

第 2 ステップの MONI コンパイラ使用については、単に学習者が用意した C ソースコードを MONI コンパイラに通し、アセンブリコードを生成するのではなく、コンパイラの仕組みをこのステップで学習してもらう。MONIC の仕様で記述したプログラムが、MONIC ソースコード→トークン列→構文情報（中間表現）→MONI アセンブリコードとなる流れを、一つ一つそれぞれ解析していく結果と解析ファイルを確認することで、より細かくコンパイラについての学習をしてもらう。

第 3 ステップでは MONI コンパイラから生成された MONI シミュータでシミュレーションを行ってもらう。ここで、本研究で設計した MONI コンパイラでは最適化処理を導入していないので、最適化処理を追加することによって、より簡潔で効率のいいアセンブリコードにすることができる。1 からアセンブリプログラミングを行ってもらうより、MONI 命令セットを理解しながら少しずつ効率のよいコードに最適化していくことが、アセンブリ言語の理解向上に繋がる。

第 4,5,6 ステップは、これまでの HSCS 学習のソフトウェア学習フローと同じである。1~3 までのステップを活かし、アセンブリ言語の理解をこれまで以上深めることで、プロセッサアーキテクチャの理解に取り組んでもらう。MONI シミュレータでは、現在、シングルサイクル、マルチサイクル、そしてパイプラインとスーパースカラの 4 つのアーキテクチャを学習できるように用意されている。しかしこれまでの学習者の評価では、シングルサイクル以外が難しくて理解できずに、ハードウェア学習に進んでいる。今後は 1~3 のステップを細かく学習してもらうことで、MONI における命令セットアーキテクチャの理解とアセンブリプログラミング能力の習得を効率化させ、時間をかけて複雑なプロセッサアーキテクチャの理解に繋げて頂く。

以上のステップが、HSCS におけるコンパイラ学習システム実装後のソフトウェア学習内容となり、アセンブリプログラミング能力の習得、コンパイラの構成の理解、そしてプロセッサアーキテクチャ理解の 3 つの学習役割を持つ。

5.2　MONI コンパイラの評価

MONI アセンブリプログラムの N までの総和と除算のプログラムにおいて、手書きのコードと MONI コンパイラによって生成されたコードを MONI シミュレータで検証比較した結果を表 4 に示す。

表 4：手書きコードと生成コードの検証結果

		手書きコード	生成コード
Nまでの総和 (N=10)	静的命令数	12	12
	動的命令数	57	59
除算 15/2	静的命令数	15	21
	動的命令数	48	80

表 4 において、検証は MONI シミュレータのシングルサイクルアーキテクチャで行っており、動的命令数が実行クロック数となる。N までの総和は計算が比較的簡易で動的命令数も大きな差はない。しかし除算のプログラムにおいて、動的命令数に大きな差がでている。これは除算の MONIC ソースコードで、

```
        .
        .
D  =  x  /  y ;
M  =  x  %  y ;
        .
        .
```

の計算部分が要因である。計算部分の手書きコードと MONI コンパイラによる生成コードの一部を図 18 に示す。変数 x、y は$0、$1、また D、M は$4、$5 のレジスタを示す。商を$4、剰余を$5 に代入している。

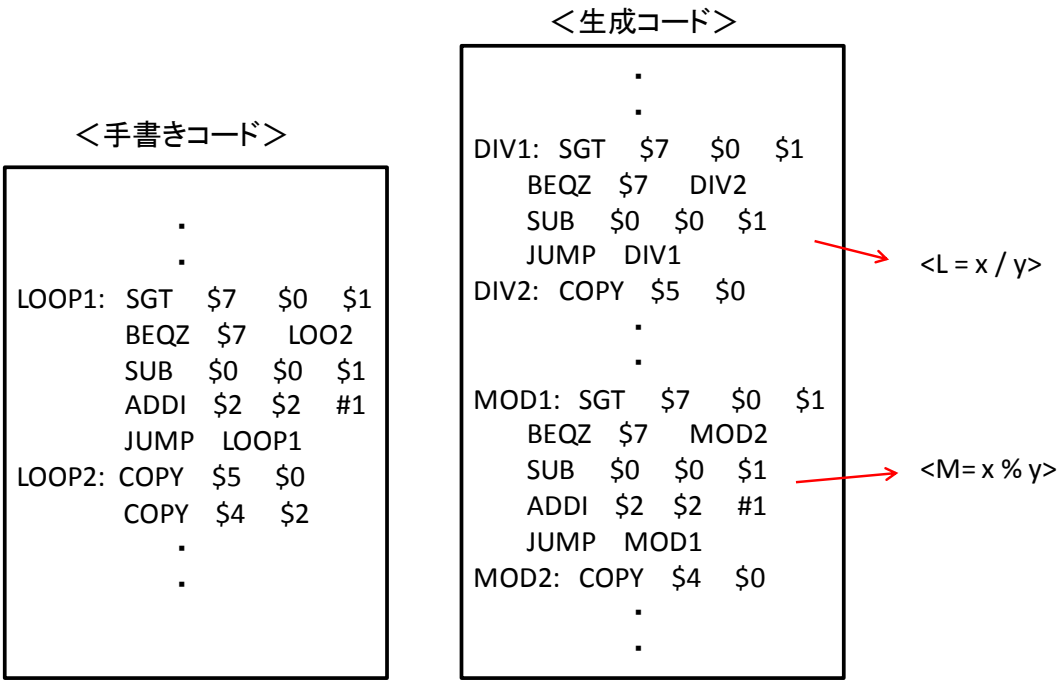

図 18：除算における手書きと生成コードの一部

図 17 のように手書きのコードでは除算と余りを求める計算を一度のループで同時に行うことができるが、MONI コンパイラでは最適化処理を行わないので、入力された MONIC ソースの計算を 1 つずつ行う。このことから始めの除算で 1 つのループ処理、後の余りを求める計算でもう 1 つのループ処理を生成するので命令数が大幅に増える。これは MONI コンパイラを改善すべき点ではなく、学習者自信が最適化を行ってもらう点とする。学習者が用意したソースコードを MONI アセンブリコードに変換し、その MONIC ソースコードの演算処理と比較することでアセンブリ言語の記述法を理解し、最適化処理を自信で行うことでコーディング能力の強化ができる。

6. おわりに

本論文では、ハード/ソフト協調学習のためのコンパイラ学習システムの設計と評価を行った。そして、ハード/ソフト協調学習システムに本研究で設計したコンパイラ学習システムを実装した後の学習方法について検討した。

コンパイラ学習システムにおける MONI コンパイラを使用することで、学習者に対してハードウェアとソフトウェアのトレードオフをより幅広い分野で学習できる環境を提供できる。アセンブリ言語の理解がソフトとハード両分野により効果的な学習効果を得ることを期待している。

今後の課題は、MONI コンパイラを学習者に使用してもらうことが挙げられる。MONIC の言語仕様の改善、学習者がアセンブリ言語の理解促進に繋がるのか、またアセンブリ言語の理解から複雑な命令セット、またパイプラインやスーパースカラの複雑なプロセッサアーキテクチャをスムーズに学習できるのか効果を評価する必要がある。MONI コンパイラの評価から改善を行い、学習者にとってハードウェアとソフトウェア両分野をより効果的に学習できるハード/ソフト協調学習システムを目指す。

謝辞

本研究の機会を与えてくださり、ご指導を頂きました山崎勝弘教授に深く感謝いたします。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卒業生の難波翔一郎氏、志水建太氏をはじめ、様々な面で貴重な助言や励まし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，久我守弘，紫村英智：KITE マイクロプロセッサによる計算機工学教育支援システム，電子情報通信学会論文誌，Vol.J84-D-1，No.6，pp.917-926，2001.

[2] 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.

[3] 松田健，佐藤暁，森健介，堤利幸：教育用マイコン COMET II の C コンパイラ開発(FPGA とその応用及び一般)，情報処理学会研究報告，Vol.2007 No.2，pp.67-72，2007

[4] 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.

[5] 大八木睦：ハードウェア記述言語によるマルチサイクル/パイプラインマイクロプロセッサの設計，立命館大学理工学部情報学科卒業論文，2002.

[6] 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.

[7] 大八木睦：ハード/ソフトコラーニング上でのアーキテクチャ可能なプロセッサシミュレータ(MONI シミュレータ)の使用法，立命館大学理工学研究科, 2004.

[8] 池田修久，中村浩一郎，大八木睦，Hoang Anh Tuan，山崎勝弘，小柳滋：ハード/ソフト・コラーニングシステムにおける FPGA ボードコンピュータの設計，情報処理学会，第 66 回全国大会講演論文集，5T-5，2004.

[9] 大八木睦，池田修久，山崎勝弘，小柳滋：ハード/ソフト・コラーニングシステムにおけるアーキテクチャ選択可能なプロセッサシミュレータの設計，情報処理学会，第 66 回全国大会講演論文集，5T-6，2004.

[10] 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.

[11] 難波翔一郎：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.

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

[13] 難波翔一郎：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.

[14] 志水建太：ハード/ソフト協調学習システム上でのプロセッサ設計とプロセッサデバッガによる検証、立命館大学理工学部情報学科卒業論文、2007.

[15] 榎本雄太：ARM ライクプロセッサの設計と FPGA ボードへの実装の検討、立命館大学理工学部情報学科卒業論文、2007.

[16] 難波翔一郎,志水建太,山崎勝弘,小柳滋：プロセッサ設計支援ツールの設計・実装とハー

ド/ソフト協調学習システムの評価、FIT2007，LC002，2007.
[17] 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電子情報デザイン学科卒業論文，2008.
[18] 井手純一,志水建太,山崎勝弘,小柳滋:学生のよるプロセッサ設計実験に基づいたハード/ソフト協調学習システムの評価，FIT2008，C-009，2008.
[19] 志水，他：プロセッサ設計教育における命令セット定義ツールと命令セットシミュレータの試作，情報処理学会，関西支部大会講演論文集，A-05，2008.
[20] 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.
[21] 宮崎匡史：プロセッサ設計支援ツールを用いた独自プロセッサの設計，立命館大学理工学部電子情報デザイン学科卒業論文，2009.
[22] 井手純一,志水建太,山崎勝弘：ハード/ソフト協調学習のためのコンパイラ開発の検討，情報処理学会，第 71 回全国大会論文集，2k-3，2009.
[23] David A.Patterson/John L.Hennessy 著／成田光彰　訳：コンピュータの構成と設計（上）（下)、日経 BP 社、2006.
[24] 中田育男：コンパイラの構成と最適化，朝倉書店，2005.
[25] 原田賢一：コンパイラ構成法，共立出版，1999.
[26] 中田育男：コンパイラ，オーム社，1995.

付録

(A) Flex 字句解析定義

```
%{
#include <string.h>
#include "y.tab.h"
int n=0;
int N=0;

char *IDentry(char *, int);

%}
variable [a-zA-Z]
div [/]
mod [%]
%%
[ ¥t¥n¥r]
= {return(EQ);}
== {return(EE);}
> {return(LC);}
>= {return(LEC);}
{mod} {return(MOD);}
{div} {return(DIV);}
goto {return(GOTO);}
if {yylval.n=++n;return(IF);}
while {yylval.n=++n;return(WHILE);}
int {return(INT);}
halt {return(HALT);}
scanf {return(SCANF);}
{variable} {yylval.s = IDentry(yytext, yyleng); return(VARIABLE); }
[0-9]+ {yylval.s=strdup(yytext);return(NUMBER);}
[a-zA-Z][a-zA-Z0-9]* {yylval.s=strdup(yytext);return(NAME);}
. {return(yytext[0]);}
%%
int yywrap(){ return(1);}
```

(B) Bison 構文解析定義

```
%{
#include <stdio.h>
#include "moniCodeGen.h"
%}
%union{ char *s, BNZ, JUMP, MOJI;int n , N;}
%token <s> NAME NUMBER VARIABLE
%token <n> IF WHILE
%token EQ GOTO SCANF INT HALT NONE
%destructor {free($$);} NAME NUMBER VARIABLE
%left EQ LC LEC S SE EE
%left MOD DIV
%left '+' '-'
%left '*'
%%
input : statement | input statement
;
statement : '¥n'
                | label | intdef | goto | if | while | scanf | assign
                | none | variable | halt
                | error '¥n'
;
label : NAME ':' {printf("%s:¥n",$1);}
;
intdef: INT intlist ';'
;
intlist: integer
      | intlist ',' integer
;
integer: NAME {printf("%s = 0¥n",$1);}
      | NAME '=' NUMBER {printf("%s = %s¥n",$1,$3);}
      | NAME '=' '-' NUMBER {printf("%s = -%s¥n",$1,$4);}
;
none : NONE {printf("¥t");}
goto: GOTO NAME ';' {printf("¥tJUMP %s¥n",$2);}
;
```

```
if: IF '(' expr ')' GOTO NAME ';' {printf("¥tJNZ %s¥n",$6);}
;
while: WHILE {printf(" %02dT:¥n",$1);} '(' expr ')' {genCode(1);}
{genCodeL($1),monicode("¥tGO¥n");}
    '{' expr ';' expr ';' '}' {printf("¥tJUMP ");} {printf(" %02dT¥n %02dF:¥n",$1,$1);}
;
scanf: SCANF '(' expr ')' GOTO NAME ';' {printf("¥tLD %s¥n",$6);}
;
variable : VARIABLE {printf("¥t$%s¥n",$1);} expr ';'
;
halt : HALT ';' {printf("¥tHALT¥n");}
;
assign : expr ';'
        ;
expr : VARIABLE {genCodeV($1);}
    | NAME {printf("¥t$%s¥n",$1);}
    | NUMBER {genCodeN($1);}
    | expr '+' expr {genCode(3);}
    | expr '-' expr {genCode(4);}
    | expr MOD expr {genCode(10);}
    | expr DIV expr {genCode(11);}
    | expr '*' expr {genCode(12);}
    | expr LC expr {genCode(5),monicode("¥tGO¥n");}
    | expr LEC expr {genCode(6);}
    | expr S expr {genCode(7);}
    | expr SE expr {genCode(8);}
    | expr EE expr {genCode(9);}
    | expr EQ expr {monicode("¥tGO¥n");}

%%
int yyerror(char *s){ printf("%s¥n",s); }
int main(){ yyparse(); }
```