# プログラムが
# コンピュータで動く仕組み

## ハードウェア記述言語・CPUアーキテクチャ・
## アセンブラ・コンパイラ超入門

博士(工学) 中野　浩嗣
博士(工学) 伊藤　靖朗
【共著】

コロナ社

# まえがき

本書は，「プログラムがどのような仕組みでコンピュータ上で動いているのか？」という疑問になるべく簡単に答えるのを目的としている。実際に使われているプログラム言語とコンピュータは複雑すぎるため，これらを用いてこの疑問に答えるには多大な時間と労力を必要とする。そこで本書では，必要最小限のごく小さなコンピュータとごく小さなC言語風プログラミング言語を題材に，この疑問に答える。なるべく少ない時間と努力でプログラムがコンピュータで動作する仕組みの本質を理解するのが目的である。

多くの人がC言語などのプログラミングを学び，実際にプログラミングを行っている。しかし，プログラムがどのようにコンピュータ上で動作するのか理解した上でプログラムを作成している人は少ないであろう。プログラムの字面を見てその動作を追っている場合がほとんどであると思われる。プログラム開発者が記述したC言語プログラムは，コンパイラとアセンブラにより機械語プログラムに変換され，コンピュータのCPU上で動作する。どのような機械語に変換されるのかということを意識することにより，プログラム開発者はより効率のよいプログラムを書けるようになることが期待できる。

本書では，プログラムがコンピュータで動作する仕組みを短時間で理解するために，ごく小さなCPUのTinyCPUを設計し，そのCPU上で直接実行できる機械語プログラムの動作シミュレーションを行う。また，TinyCPUの機械語プログラムに変換することを想定したアセンブリ言語TinyASMを導入し，TinyASMプログラムを機械語プログラムに変換するアセンブラを設計する。さらには，C言語風プログラミング言語TinyCを導入し，TinyCプログラムをTinyASMプログラムに変換するコンパイラを設計する。以上により，コンパイラを用いてC言語風のTinyCプログラムをTinyASMプログラムに変換

し，アセンブラを用いて機械語プログラムに変換すれば，TinyCPU 上で動作させることができる。

TinyCPU の設計にはハードウェア記述言語 Verilog を用いる。Verilog の言語仕様は C 言語に似ており，デジタル回路の動作をソフトウェアプログラム的に記述する。論理設計ツールを用いることにより，Verilog による回路記述を実際のデジタル回路に変換することができる。本書では Verilog について基礎から学習するので，Verilog によるデジタル回路設計法を習得することができる。

TinyASM のアセンブラは Perl を用いて記述する。Perl は C 言語に似たスクリプト言語で，文字列処理や連想配列の操作を簡単に記述することができる。Perl の知識がまったくないことを前提に，Perl の基礎を説明するので，アセンブラを理解するにあたり，Perl の事前知識は不要である。

TinyC のコンパイラ設計は，コンパイラ作成ツールである Flex（字句解析ツール）と Bison（構文解析ツール）を用いる。これらも基本から説明するので，コンパイラ設計を理解するにあたり，これらのツールの事前知識も不要である。

一方，本書では，さまざまな回路をハードウェア記述言語 Verilog を用いて設計していくので，デジタル回路設計のある程度の基礎知識が必要である。本書内でも簡単には説明しているが，論理積，論理和，全加算器，セレクタ回路，組み合わせ回路，フリップフロップ，カウンタ回路，メモリ回路，順序回路を知っていると，Verilog でのデジタル回路設計が容易に理解できるだろう。本書のデジタル回路設計の内容については，拙著 1)† と用語を統一しているので，これまでにデジタル回路設計を勉強したことのない読者には，この本をざっと読まれることをお薦めする。

C 言語プログラミングについても，基礎的な知識があることを前提としている。TinyC は C 言語のごく小さなサブセットであり，変数宣言や，if 文，while 文，do 文については C 言語の仕様をそのまま引き継いでいる。また，TinyC の

† この片かっこ番号は，巻末の引用・参考文献の番号を表す。肩付きで表記する場合もある。

コンパイラ設計では，関数呼び出しや return，インクリメント演算（++）を用いるので，C 言語プログラミングにおいてこれらがなにを意味するのかを知っている必要がある。さらに，C 言語を知っている前提で，TinyASM のアセンブラに用いる Perl プログラミングを説明している。

本書の大きな特徴は，つぎの項目を一度に学ぶことができる点にある。

- Verilog によるデジタル回路設計
- プロセッサアーキテクチャ
- 機械語プログラミング
- アセンブリ言語プログラミング
- アセンブラ設計
- Perl プログラミング
- コンパイラ設計

ただし，個々の項目についてその内容を完全に網羅しているわけではない。いずれもごく基礎的で導入的な内容に限っている。本書により，これらの項目に興味をもたれた方は，より詳しく書いた専門書で勉強されることをお薦めする。なお，本書を執筆するにあたり，Verilog については文献 2) を参考にした。また，Flex と Bison については，文献 3) と文献 4) を参考にした。

本書は 7 章で構成される。

1 章は，Verilog を用いた組み合わせ回路の設計法について説明している。半加算器，全加算器，加算器，セレクタ回路などを具体例に，Verilog によるデジタル回路の記述方法を学ぶ。また，Verilog のテストベンチを作成し，Icarus Verilog を用いてシミュレーションを行い，波形表示ツール GTKWave を用いてタイミングチャートを画面表示して，Verilog により設計したデジタル回路が意図したとおりに正しく動作することを確認する。また，TinyCPU の構成部品である算術論理演算回路を設計する。

2 章は，Verilog を用いた順序回路の設計法について説明している。まず，フリップフロップの設計法を説明し，それを拡張することにより，カウンタ回路，スタック回路，メモリ回路を設計している。これら三つの回路は TinyCPU の

構成部品となる。

TinyCPU の設計をボトムアップに行うために，3 章では，TinyCPU の一部の機能だけを実現する回路を設計する。具体的には，式の計算をスタック上で行う演算スタック回路，メモリ回路に格納されている機械語プログラムの機械語コードを順次取り出して命令レジスタに格納する命令フェッチ回路，メモリ回路に格納されている式の計算を行う機械語プログラムを実行する式計算回路，命令レジスタにある 16 ビットの機械語コードの上位 8 ビットをアドレス下位 8 ビットをデータとみなしてメモリ回路への書き込みを行う拡張命令フェッチ回路，の四つの回路を設計する。

4 章では TinyCPU を Verilog で設計する。まず，TinyCPU の構造と機械語命令セットを説明する。TinyCPU はこれまでに設計した回路を構成部品として組み合わせたものである。それらの回路を制御する 15 本の制御線を設定し，その制御線がどのような値をとるかを定義する。その定義に基づいて，TinyCPU の Verilog ソースコードを記述する。ごく簡単な機械語プログラムを対象にシミュレーションを行い，タイミングチャートを表示して，正しく動作することを確認する。

5 章では，TinyCPU 向けのアセンブリ言語 TinyASM の仕様と TinyASM プログラミングを説明する。TinyC の基本構文（代入文，if 文，if–else 文，while 文, do 文）を TinyASM プログラムに手作業で変換するハンドコンパイルの方法を具体例をみながら説明する。そして，コラッツの問題の計算を行う TinyASM プログラムと，ユークリッドの互除法により最大公約数を求める TinyASM プログラムを作成する。これらをハンドアセンブルにより機械語プログラムに変換し，TinyCPU 上での動作シミュレーションを行う。

6 章では，TinyASM プログラムを機械語プログラムに自動的に変換するアセンブラを Perl を用いて設計する。Perl の知識がなくても理解できるよう，Perl の超入門から始める。Perl の超入門では，スカラ変数，リスト，連想配列，パターンマッチ，文字列の置換など，Perl プログラミングの基本を説明する。これらの Perl の機能を用いて，アセンブラを設計する。

7章では，TinyCプログラムをTinyASMプログラムに自動的に変換するコンパイラをコンパイラ作成ツールのFlex（字句解析ツール）とBison（構文解析ツール）を用いて設計する。これらのツールの使い方を学ぶため，後置記法の式の計算を行うプログラムをFlexを用いて設計する。また，中置記法の式の計算を行うプログラムをFlexとBisonの両方を用いて設計する。そして，TinyCの代入文専用のコンパイラを設計し，最後にTinyCコンパイラを設計する。

各章には演習問題があるが，その章までに学んだ知識だけで解くには難しいものもいくつか含まれており，区別のためそのような問題には，問題番号の右肩にアステリスク（*）を付けている。より現実的な回路設計やコンパイラ設計では用いられている内容であり，興味ある読者にはぜひ挑戦してほしい。

なお，本書を執筆するにあたり，Flex と Bison，および GNU C コンパイラ（gcc）は，Windows 上に Linux に似た環境を提供する Cygwin に含まれているものを用いた。また，Icarus Verilog（Verilog シミュレータ）とGTKWave（波形表示ツール）は，Windows 版のものを用いた。Cygwin は`https://www.cygwin.com/`からインストーラをダウンロードすることができる。また，Icarus Verilog と GTKWave の Windows 版インストーラは，`http://bleyer.org/icarus/`からダウンロード可能である†。

本書に掲載したプログラム（リスト連番のあるプログラム）のソースファイルおよび章末演習問題の解答のうち主要なものを，コロナ社ホームページの本書サポートページ `https://www.coronasha.co.jp/np/isbn/9784339029222/` にアップするので，必要に応じてダウンロードあるいは参照してほしい。

最後に，本書を出版するにあたり，お世話いただいたコロナ社に深く感謝を申し上げる。

2021年9月

中野浩嗣，伊藤靖朗

† 本書に示したURLは，2021年6月現在のものである。

# 目　　次


## 1章 Verilog による組み合わせ回路の設計

1.1 Verilog の基本構文とシミュレーション …… 2
1.1.1 Verilog による半加算器の回路記述 …… 2
1.1.2 半加算器のテストベンチ …… 5
1.1.3 Icarus Verilog によるシミュレーション …… 8
1.1.4 モジュールのインスタンス化 …… 10
1.1.5 Verilog の定数表現 …… 16
1.1.6 Verilog の演算子 …… 18
1.1.7 always　　文 …… 22
1.1.8 ビット数可変の組み合わせ回路 …… 24
1.2 セレクタ回路とその応用 …… 27
1.2.1 セレクタ回路 …… 28
1.2.2 7 セグメントデコーダ回路 …… 34
1.2.3 算術論理演算回路 …… 36
演 習 問 題 …… 42

## 2章 Verilog による順序回路の設計

2.1 フリップフロップとカウンタ回路の設計 …… 46
2.1.1 フリップフロップの設計 …… 46
2.1.2 カウンタ回路の設計 …… 52
2.2 ステートマシン回路の設計 …… 55
2.3 スタック回路の設計 …… 59
2.4 メモリ回路の設計 …… 63

演 習 問 題 …… 66

## 3章 TinyCPU の設計の準備

3.1 演算スタック回路 …… 70
3.1.1 中置記法と後置記法 …… 70
3.1.2 演算スタック回路の設計 …… 72
3.2 命令フェッチ回路 …… 76
3.3 式 計 算 回 路 …… 81
3.4 拡張命令フェッチ回路 …… 84
3.4.1 バ ス …… 84
3.4.2 バスを用いた拡張命令フェッチ回路 …… 86
演 習 問 題 …… 90

## 4章 TinyCPU の設計

4.1 TinyCPU の構成部品と機械語コード …… 93
4.2 TinyCPU の構造と動作 …… 97
4.3 TinyCPU の Verilog ソースコード …… 102
4.4 TinyCPU の簡単なプログラム例 …… 106
4.5 TinyCPU のテストベンチとシミュレーション …… 108
演 習 問 題 …… 112

## 5章 アセンブリ言語プログラミング

5.1 アセンブリ言語と C 言語 …… 115
5.1.1 アセンブリ言語 TinyASM …… 115
5.1.2 C 言語風プログラミング言語 TinyC …… 116
5.2 基本構文の TinyASM プログラム …… 117
5.2.1 代 入 文 …… 117
5.2.2 if 文 …… 118

5.2.3 while 文 …… 122
5.2.4 do 文 …… 124
5.3 TinyASM プログラムの例 …… 125
5.3.1 コラッツの問題 …… 125
5.3.2 ユークリッドの互除法 …… 128
演 習 問 題 …… 130

6章 アセンブラの設計

6.1 TinyASM アセンブラ …… 133
6.2 Perl 超 入 門 …… 133
6.2.1 Perl プログラムの基本構造とスカラ変数 …… 133
6.2.2 ファイルの読み出し，リスト，連想配列 …… 135
6.2.3 文字列の置換 …… 138
6.3 アセンブラの設計 …… 139
演 習 問 題 …… 143

7章 コンパイラの設計

7.1 TinyC コンパイラの概略 …… 146
7.2 Flex を用いた後置記法の式の計算 …… 147
7.3 Flex と Bison を用いた中置記法の式の計算 …… 151
7.3.1 Flex による字句解析 …… 151
7.3.2 Bison による構文解析 …… 153
7.3.3 中置記法の式計算プログラムの生成と実行 …… 158
7.4 代入文専用のコンパイラ …… 159
7.4.1 Flex による字句解析 …… 160
7.4.2 Bison による構文解析 …… 161
7.5 TinyC コンパイラ …… 165
7.5.1 Flex による字句解析記述 …… 165
7.5.2 Bison による構文解析記述 …… 167

7.5.3 変数の宣言のための文法規則 …… 170
7.5.4 if 文のための文法規則 …… 171
7.5.5 if–else 文のための文法規則 …… 173
7.5.6 while 文のための文法規則 …… 175
7.5.7 do 文のための文法規則 …… 176
7.5.8 halt 文と out 文のための文法規則 …… 178
7.5.9 TinyC プログラムのコンパイル例 …… 179
演 習 問 題 …… 182

**引用・参考文献** …… 184
**索 引** …… 185

# 1章 Verilogによる組み合わせ回路の設計

◆本章のテーマ

ハードウェア記述言語 Verilog の基本構文と，組み合わせ回路の設計方法を具体的な回路の記述例を参照しながら理解する。また，回路シミュレーションのためのテストベンチの記述法と，Icarus Verilog を用いてテストベンチをコンパイルし，タイミングチャートを表示する方法を学ぶ。セレクタ回路の記述法と，それを応用した算術論理演算回路を設計する。

◆本章の構成（キーワード）

1.1 Verilog の基本構文とシミュレーション

半加算器，全加算器，加算器，モジュール，テストベンチ，タイミングチャート，モジュールのインスタンス化，定数表現，演算子，配線，変数，input 文，output 文，wire 文，reg 文，assign 文，always 文，initial 文，parameter 文，define 文，include 文

1.2 セレクタ回路とその応用

セレクタ回路，7 セグメントデコーダ，算術論理演算回路，case 文，default 文，不定値

◆本章を学ぶと以下の内容をマスターできます

- ☞ Verilog の基本構文
- ☞ assign 文と always 文を用いた組み合わせ回路の記述法
- ☞ シミュレーションのためのテストベンチの書き方
- ☞ Icarus Verilog を用いたシミュレーションとタイミングチャートの表示
- ☞ ビット数が可変の組み合わせ回路の記述法
- ☞ 算術論理演算回路の設計方法

# 1.1 Verilog の基本構文とシミュレーション

Verilog は，回路設計のための記述言語（**ハードウェア記述言語**，hardware description language，**HDL**）である。記述は C 言語と似ており，回路設計者はデジタル回路の動作をソフトウェアプログラム的に記述する。シミュレータを用いて，記述された動作をシミュレーションすることにより，回路設計者の意図した動作となっているかどうかを確認することができる。また，論理合成ツールを用いて Verilog による回路記述を実際のゲート回路やフリップフロップを用いたデジタル回路に変換することができる。Verilog で回路を記述すれば，ゲート回路やフリップフロップを配置した回路の図面を描くことなく，デジタル回路を設計することができる。

## 1.1.1 Verilog による半加算器の回路記述

Verilog による回路記述の最も簡単な例として，**半加算器**を設計する。半加算器は，1 ビットの入力 x と y および 1 ビットの出力 s と c をもつ。

**図 1.1**（a）に示したように，加算 x+y の結果が 2 ビット cs に求められる。ここで，s が合計で c が桁上がりである。出力 s が 1 となるのは，入力 x と y のいずれか一つだけが 1 のときである。また，出力 c が 1 となるのは，入力の両方が 1 のときである。よって，s と c を求める論理式は，つぎのように書くことができる。

```
   x
+  y
-------
c  s
```

| x | y | c | s |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

（a） 半加算器の計算　　（b） 半加算器の真理値表

**図 1.1**　半加算器の計算と真理値表

```
s = x ^ y
c = x & y
```

この二つの論理式で用いられている ^ と & は，C 言語と同じで，それぞれ排他的論理和と論理積を表す。半加算器の**真理値表**（ブール関数の入力と出力の対応表）は図 ( b ) のようになる。

**リスト 1.1** は，s と c の論理式をもとに記述した半加算器の Verilog ソースコードである。Verilog ソースコードを保存するファイルの拡張子は.v を用いる。また，**図 1.2** は，その Verilog ソースコードにより設計される半加算器の構造を表しており，排他的論理和と論理和の計算に，XOR ゲートと AND ゲートをそれぞれ用いている。

**リスト 1.1** 半加算器の Verilog ソースコード ha.v

```
1  module ha(x,y,s,c);  // 半加算器ha，ポートリストx,y,s,c
2
3    input x,y;  // 入力ポートx,y
4    output s,c;  // 出力ポートs,c
5    wire x,y,s,c; // 配線x,y,s,c（省略可）
6
7    assign s=x^y; // 配線sへの継続的代入
8    assign c=x&y; // 配線cへの継続的代入
9
10 endmodule
```

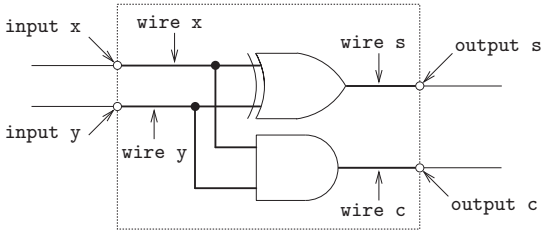


**図 1.2** 半加算器のモジュール ha の構造

Verilog のソースコードは，一つ以上のモジュールから構成される。**モジュール**とは，**入力ポート**や**出力ポート**をもつひとまとまりの回路である。リスト 1.1 の

モジュールは，1 行目の**モジュール宣言** `module` で始まり，10 行目の `endmodule` で終わる。

1 行目では**モジュール名**が `ha` であることを宣言している。それにつづく丸かっこの中は，そのモジュールの**ポートリスト**であり，`x`, `y`, `s`, `c` のそれぞれが入力または出力のためのポートであることを宣言している。モジュールは C 言語の関数と似ており，ポートは C 言語の引数と同様の性質をもち，外部と値のやり取りを行うのに用いられる。ここで，「`//`」の後ろに説明が書かれているが，C 言語と同じく「`//`」から行末までがコメントであり，Verilog のシミュレータや論理合成ツールはこの部分を無視する。また，複数行にわたるコメントの場合は，C 言語と同様に，`/*` 〜 `*/` を用いることができる。

3 行目の **input 文**は，`x` と `y` が 1 ビットの入力ポートであることを意味しており，モジュール外部からのデータを受け取ることができる。

4 行目の **output 文**は，`s` と `c` が 1 ビットの出力ポートであることを宣言しており，モジュール外部にデータを出力することができる。

5 行目の **wire 文**は，`x`, `y`, `s`, `c` がモジュール内部の**ネット**，つまり要素を接続する**配線**であることを宣言している。ここで `x`, `y` は配線でありかつ入力ポートでもあるので，入力ポートの値がそのまま同じ名前の配線の値となる。つまり，入力ポート `x`（`input x`）と配線 `x`（`wire x`）は図 1.2 のように接続していると考える。同様に，配線 `s`（`wire s`）と `c`（`wire c`）の値はそのまま出力ポート `s`（`output s`）と `c`（`output c`）の値となる。wire 文で宣言された配線が入力ポートまたは出力ポートでもある場合，wire 文による配線の宣言（5 行目）は省略できる。今後は，基本的にこのような wire 文は省略することにする。

7 行目と 8 行目の **assign 文**は，等号「`=`」の右辺の式の値が左辺の配線に継続的に代入されつづけることを意味する。つまり，右辺の式の値が変更されると，ただちに左辺の配線に代入される。よって，配線 `s` と `c` に，半加算器として適切な値が継続的に代入されることがわかる。このように assign 文を用いることにより，配線の永続的な接続関係を定義することができる。

### 1.1.2 半加算器のテストベンチ

モジュールが正しいかどうかを確認するには，さまざまな入力に対して出力が意図したとおり得られるかどうかのシミュレーションを行う方法がある，入力が多い場合，すべての入力の組み合わせをシミュレーションすることは，組み合わせ数が膨大になり現実的には不可能なので，正しさの厳密な証明にはならない。しかし，シミュレーションにより明らかなバグはなく，たぶん正しいだろうということを，人の目で確認することができる。

モジュールのシミュレーションを行うために，**テストベンチ**を作成する。テストベンチでは，シミュレーションの対象となるモジュールへの入力ポートに与える値を時系列的に定義する。そして，出力ポートから得られる値の変化をみて，入力と出力の対応が正しいことを確認する。**リスト 1.2** は半加算器のモジュール `ha` を対象とするテストベンチの Verilog ソースコードである。テストベンチも Verilog のモジュールであり，2 行目から 18 行目がテストベンチ `ha_tb` のモジュール本体である。

リスト **1.2** 半加算器のテストベンチ `ha_tb.v`

```
1  `timescale 1ns/1ns  // 1単位時間1ns，精度1ns。今後は省略する
2  module ha_tb;  // 半加算器のテストベンチha_tb
3
4    reg x,y;  // 変数x,yの宣言
5    wire s,c;  // 配線s,cの宣言
6
7    ha ha0(.x(x),.y(y),.s(s),.c(c));  // haのインスタンス化
8
9    initial begin  // ここから実行開始
10     $dumpvars;  // 変数や配線の値の書き出し開始
11     x=0; y=0; #100  // x，yに値を代入し100単位時間待つ
12     x=0; y=1; #100
13     x=1; y=0; #100
14     x=1; y=1; #100
15     $finish;  // 書き出し終了
16   end
17
18 endmodule
```

# 索　　引


【あ】

アセンブラ *133*
アセンブリ言語プログラム *107*
アセンブリコード *94, 107*
アドレス *63*
アドレスバス *87*

【い】

意味値 *152, 161*
インクリメント回路 *42*
インスタンス化 *6*
インスタンス名 *6*

【え】

演算子 *18*
演算スタック回路 *72*

【お】

オペランド *70, 94, 115, 140*

【か】

還　元 *153*

【き】

機械語コード *76, 94, 106*
機械語プログラム *76, 106*
機械語命令 *94*
機械語命令セット *94*
キュー *59*
共用体 *161*

【く】

空規則 *158, 162*
クロック *46*
クロックサイクル *46*
クロック周波数 *46*

【こ】

後置記法 *70, 147*
構文解析記述 *147*
構文解析ツール *147*
コラッツの問題 *125*
コンパイラ *146*

【さ】

算術論理演算回路 *36*

【し】

式計算回路 *81*
字句解析記述 *147*
字句解析ツール *146*
シフト *154*
終端記号 *153*
出力ポート *3, 4*
真理値表 *3*

【す】

スカラ変数 *134*
スタック *59*

【せ】

正規表現 *136*
整数定数 *16*
セレクタ回路 *27*
センシティビティリスト *22*

【た】

タイミングチャート *9*
立ち上がり *46*
立ち下がり *50*

【ち】

中置記法 *70, 151*

【て】

テストベンチ *5*
データバス *87*

【と】

同期書き込み *63*
同期読み出し *67*
同期リセット *66*
トークン *146*
ドントケア *32*

【な】

7 セグメントディスプレイ *34*
7 セグメントデコーダ回路 *34*

【に】

日本語記法 *70*
ニーモニック *94*
入力ポート *3, 4*

【ね】

ネット *4*

【の】

ノンブロッキング代入文 *47*

【は】

ハイインピーダンス *86*
配　線 *4*
バ　ス *84*

ハードウェア記述言語 2
パラメータ 25
半加算器 2
番　地 63
ハンドアセンブル 107
ハンドコンパイル 115

【ひ】

非終端記号 153
左結合 153
左再帰規則 158, 162
ビット拡張 86
非同期読み出し 63

【ふ】

符号拡張 95
プッシュ操作 59
不定値 32, 33
フリップフロップ 49
ブロッキング代入文 47
プロトタイプ宣言 147

【へ】

変数宣言 170

【ほ】

ポップ操作 59
ポートリスト 4

【め】

命令フェッチ 76
命令フェッチ回路 76
命令レジスタ 76
メモリ 63
メモリ回路 63
メモリ形式の機械語プログラム 109, 127, 129

【も】

モジュール 3
——宣言 4
——名 4

【よ】

予約語 146

【ら】

ラベル 107, 115, 133

【り】

リスト 135

【れ】

連想配列 135

【A】

ALU 36
always 文 22
assign 文 4

【B】

Bison 146

【C】

case 文 31

【D】

D 型フリップフロップ 46
default 文 31
define 文 37

【F】

FIFO 59
Flex 146

【G】

genvar 文 25
GTKWave 8

【H】

HDL 2

【I】

Icarus Verilog 8
include 文 37
initial 文 6
input 文 4

【L】

LIFO 59

【M】

module 4

【O】

output 文 4

【P】

parameter 文 25

【R】

RAW 67
read after write 67
reg 文 6

【T】

timescale 文 6
TinyASM 107, 115
TinyC 115

【W】

WAR 67
wire 文 4
write after read 67

【記号】

$dumpvars 7
$finish 7

—— 著 者 略 歴 ——

**中野　浩嗣**（なかの　こうじ）
1987年　大阪大学基礎工学部情報工学科卒業
1989年　大阪大学大学院基礎工学研究科博士前期課程修了（物理系専攻）
1992年　大阪大学大学院基礎工学研究科博士後期課程修了（物理系専攻）
博士（工学）
1992年　株式会社日立製作所基礎研究所研究員
1995年　名古屋工業大学講師
1998年　名古屋工業大学助教授
2001年　北陸先端科学技術大学院大学助教授
2003年　広島大学教授
現在に至る

**伊藤　靖朗**（いとう　やすあき）
2001年　名古屋工業大学工学部電気情報工学科卒業
2003年　北陸先端科学技術大学院大学情報科学研究科博士課程前期修了（情報処理学専攻）
2003年　デンソーテクノ株式会社勤務
2004年　広島大学助手
2007年　広島大学助教
2010年　広島大学大学院工学研究科博士課程後期修了（情報工学専攻）
博士（工学）
2013年　広島大学准教授
現在に至る


# プログラムがコンピュータで動く仕組み

## —ハードウェア記述言語・CPU アーキテクチャ・アセンブラ・コンパイラ超入門—

How Program Runs on Computer
—Introduction to Hardware Description Language, CPU Architecture, Assembler, and Compiler—




2021 年 11 月 25 日　初版第 1 刷発行　★

検印省略

著　者　中野　浩嗣
　　伊藤　靖朗
発行者　株式会社　コロナ社
　　代表者　牛来真也
印刷所　三美印刷株式会社
製本所　有限会社　愛千製本所

112–0011　東京都文京区千石 4–46–10
発行所　株式会社　コロナ社
CORONA PUBLISHING CO., LTD.
Tokyo Japan
振替 00140–8–14844・電話(03)3941–3131(代)
ホームページ　https://www.coronasha.co.jp

ISBN 978–4–339–02922–2　C3055　Printed in Japan　（金）