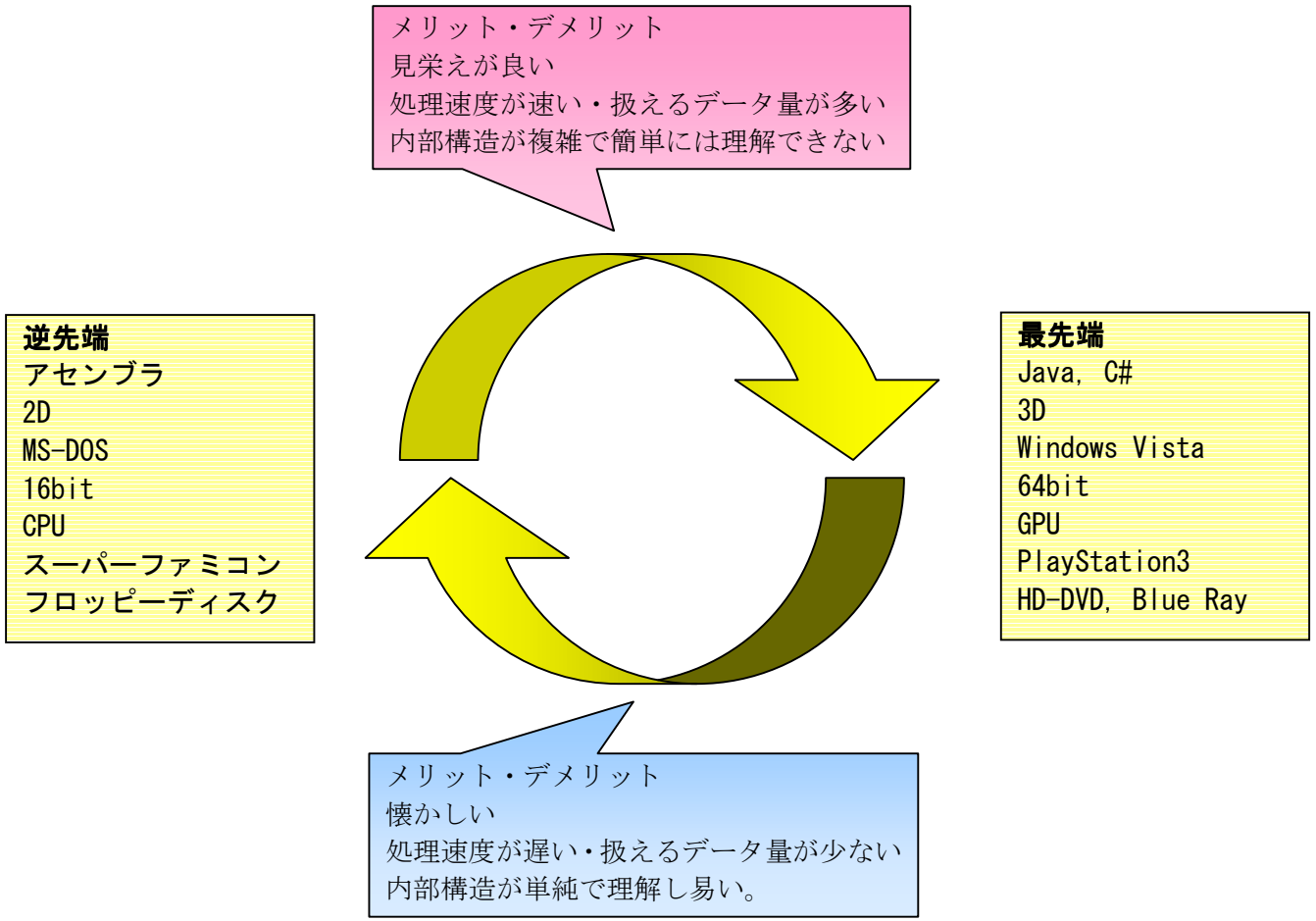
Easy CPU Maker -簡単 CPU 作成ツール-

自由部門 登録番号:20015

時代の逆先端を行け！！



現代の流行であるデータベース・Web2.0・オブジェクト指向・Java・PHP・ネットワーク・3D などから離れ、現在ではすっかり忘れ去られた感のある CPU を簡単に作成できるツールを製作することが私たちの目的である。

なぜ、今の時代に CPU なのか？

近年のコンピュータの発展には目ざましいものがある。

しかし、コンピュータが高機能になればなるほど、内部はブラックボックスと化し、

コンピュータの頭脳である CPU を知らない人たちも増えてきている。

私達は、この状況に危機感を持った。

このままでは、「コンピュータに使われてしまう」。

この状況を改善するためには、昔に戻らなければならない。

すなわち、CPU をいじりながら、CPU を理解すること、

それが現在の私達には必要になっているのである。

私達は、とても簡単な CPU を設計し、それを用いて学習することも考えた。

しかし、本当に CPU を理解するためには、「CPU を作ってしまう」ことが一番である。

一度 CPU を作ってみれば、CPU の単純さが良く分かるはずだ。

私達は、そのように考え、遂に「**Easy CPU Maker**」の企画が出来上がったのである。

対象者

・高校生以上

・CASL程度のCPUやアセンブリ言語の知識がある人

このソフトの対象者はできるだけ広くしたい。しかし、CPUの設計には、どうしてもCPUやアセンブリ言語の知識がある程度必要となる。

よって、C言語などのプログラミング言語やCPUを勉強したことのある人、もしくは高校生以上を対象に設定する。

このソフトで何ができるか？

・個性的なオリジナルCPUの設計

・アセンブリ言語やCPUの学習

・シュミレータのないマイナーCPUの再現

簡単にCPUを設計できる。本物のCPUを設計する際とは違い、内部実装を考える必要がないため、アイデア次第で個性的なCPUを作ることができる。そして、設計したCPU情報を基にして、アセンブラ・仮想マシンを実行でき、自分で作成したアセンブラプログラムを読み込ませて自由に遊ぶことができる。

また、どうしたら処理が早くなるか、こんな設定にしたらどうなるかなど、色々実験することができる。

さらに、アセンブリ言語やCPUに触れるのだから、当然勉強にもなる。

このソフトで作った CPU は、2つとして存在しないため、シュミレータがない。自分しか知らない、そんな世界を再現することもできる。

EasyCPUMaker の内部構造

Easy CPU Maker は大きく分けて、3つのファイルに分かれている。

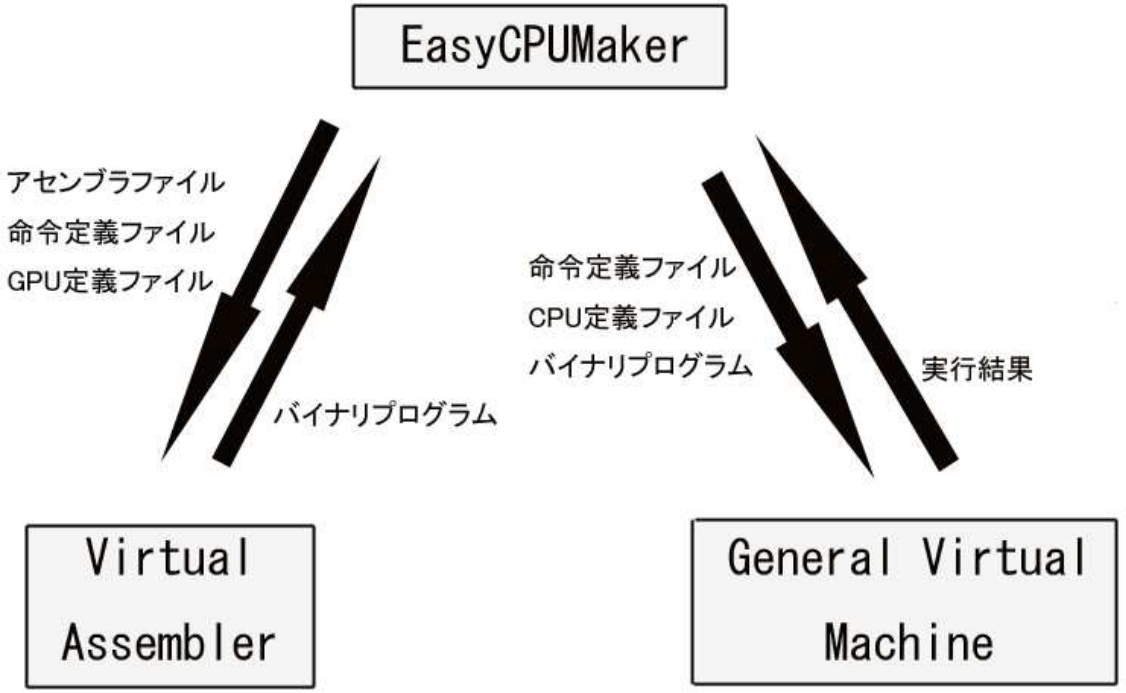
Easy CPU Maker……GUI で操作する仮想 CPU 設計ツール。（メインプログラム）

Virtual Assembler……Easy CPU Maker によって定義された情報をもとにプログラムをアセンブルするツール。

General Virtual Machine……Easy CPU Maker によって定義された情報をもとにマシン語プログラムを実行するツール。

3つのファイルが連携しあうことで、「CPU の設計」「アセンブラの作成」「プログラムの実行」

という3つの大きなプロセスを実現しているのである。



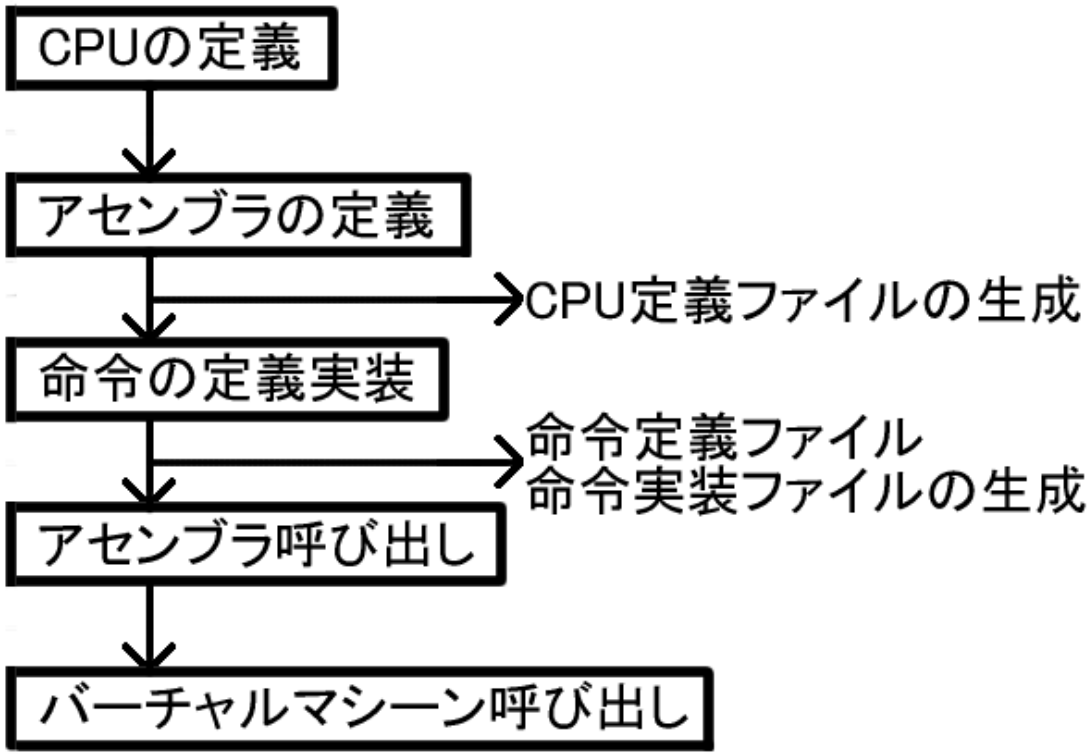
1. EasyCPUMaker

このプログラムは「CPU の設計」を行う。

まず、ウィザード形式で CPU の重要部分の設計を行い、レジスタやメモリなどを定義しておく。その後、CPU の命令を定義し、それらの情報をファイルに保存する。

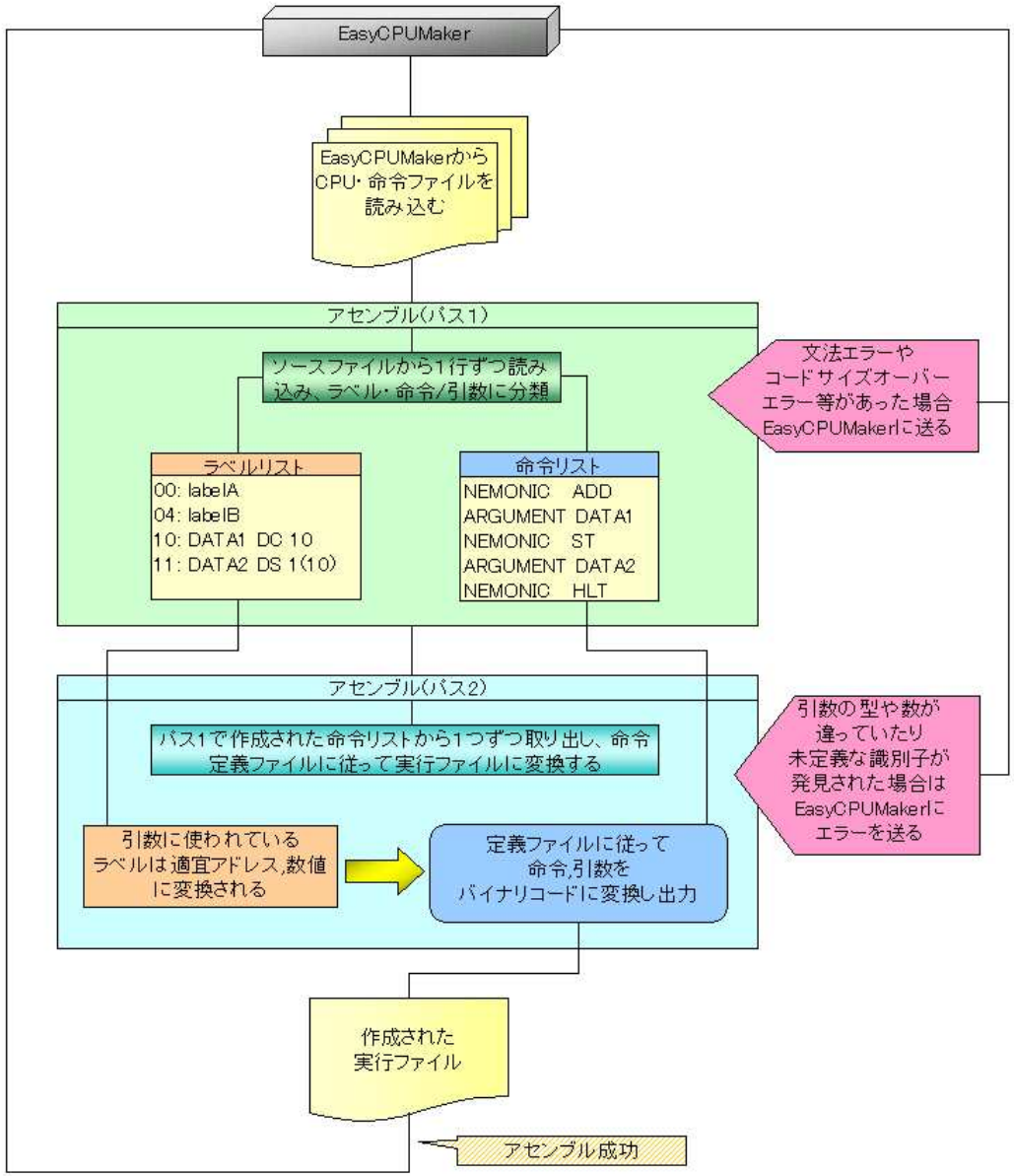
ここで保存したデータは「Virtual Assembler」や「General Virtual Machine」にて使用される。

このプログラムは内部にて「Virtual Assembler」や「General Virtual Machine」を呼び出し、アセンブラプログラムの実行環境にもなる。



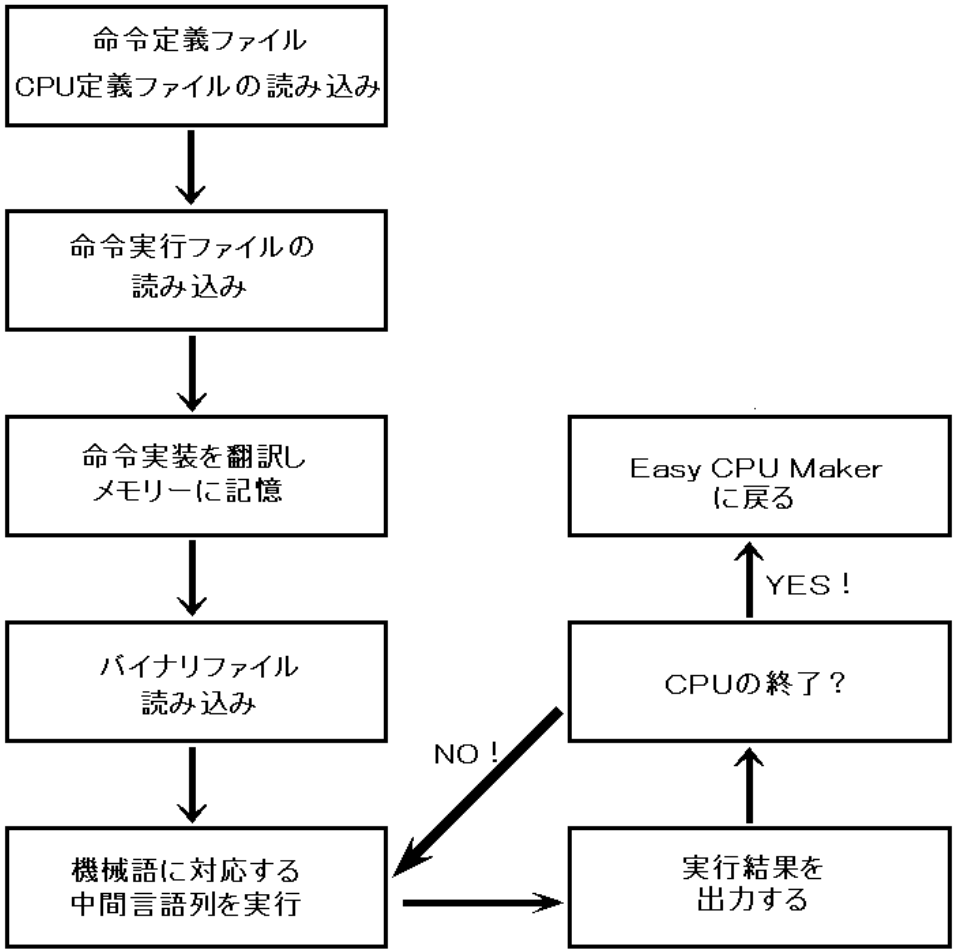
2. Virtual Assembler

このプログラムは「**Easy CPU Maker**」にて定義された情報を基に、渡されたアセンブラファイルをアセンブルする。ある程度の汎用性は犠牲になるが、CPU さえ定義してしまえばアセンブルができるので、自分でアセンブラを作るより、かなり楽になる。



3. General Virtual Machine

このプログラムは「Virtual Assembler」によってアセンブルされたマシン語プログラムを実行する。「Virtual Assembler」を用いないでも、アセンブラを自作してしまえば実行させることもできる。「Easy CPU Maker」によって定義された命令は、一度中間言語に翻訳される。そして、その命令が出てきたら、バーチャルマシンは命令に対応付けられた中間言語を実行する。これによって、一度定義さえしてしまえば、複雑な命令もバーチャルマシン上で実行できる。



CPU の作成例:

この様な CPU を作ってみることにする。

CPU 名「easy」

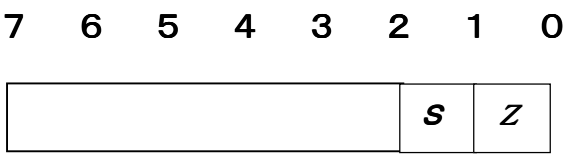
CPU の構成

レジスタはアキュムレータ1個。NOP、HLT以外の命令は全て2バイトであり、

各命令はOPコード1バイト、オペランド1バイトで構成される。

ＯＰコード	ニーモニック	オペランド	説明
０	ＮＯＰ		何もしない
１	ＬＤ	アドレス	レジスタ←メモリの内容
２	ＳＴ	アドレス	メモリの内容←レジスタ
３	ＡＤＤ	アドレス	レジスタ←レジスタ＋メモリの内容
４	ＳＵＢ	アドレス	レジスタ←レジスターメモリの内容
５	ＳＨＬ	数字	レジスタの内容を指定しただけ左シフトする ただし、数字＜＝８である
６	ＳＨＲ	数字	ＳＨＬの右シフト版。 ＳＨＬとＳＨＲは単純な論理シフトである。
７	ＪＭＰ	アドレス	指定したアドレスにジャンプする
８	ＪＺ	アドレス	ゼロフラグが立ったときジャンプ
９	ＪＮＺ	アドレス	ゼロフラグが立たないときジャンプ
１０	ＪＭＩ	アドレス	サインフラグが立ったときジャンプ
１１	ＪＰ	アドレス	サインフラグが立たないときジャンプ
１２	ＣＭＰ	アドレス	アドレスの内容とレジスタを比較して フラグをセット
１３	ＨＬＴ		実行停止

フラグレジスタの定義



0ビット目：ゼロフラグ

1ビット目：サインフラグ

アセンブラの書式：

ラベル：　　命令　　オペランド

CPU の仕様が決まったら、「**Easy CPU Maker**」を用いて、CPU の定義を行う。

1．CPU 名の決定　　この場合は“easy”とする。

2．レジスタの定義　　この場合は汎用レジスタ1個のみを定義する。

3．メモリ長の定義　　今回は8ビットしか使えないので、8ビットとする。

4．OPコード長の定義　　今回は簡単にするため8ビットとする。

5．アセンブラ定義　　今回は省略する。

つぎに、命令の定義を行う。

今回は種類が多いので省略するが、

一つだけ定義例を示す。

例：LD

```
ld reg, &1      // これはレジスタに第一オペランドアドレスの内容を
                // 代入することを意味する。
```

このように、一つの命令は複数の基本命令からなり、この基本命令を実行していくことで VM は動作をしている。「**Easy CPU Maker**」内部では、このように命令を基本命令の列で管理しているが、ソフトでは、GUI 上で分かりやすく編集が可能になっている。

CPU と命令の定義が終わったら、アセンブラプログラムを「**Virtual Assembler**」に渡し、アセンブルする。

アセンブラプログラム例：

アドレス

```
０          ＬＯＯＰ：    ＬＤ　Ｎ
２                       ＡＤＤ　ＳＵＭ
４                       ＳＴ　ＳＵＭ
６                       ＬＤ　Ｎ
８                       ＳＵＢ　ＯＮＥ
１０                     ＳＴ　Ｎ
１２                     ＣＭＰ　ＺＥＲＯ
１４                     ＪＮＺ　ＬＯＯＰ
１６                     ＨＬＴ
１７          ＯＮＥ     ＤＣ　１
１８          Ｎ         ＤＳ　１（１０）
１９          ＳＵＭ     ＤＳ　１（０）
２０          ＺＥＲＯ   ＤＣ　０
```

アセンブル後：

01 12 03 13 02 13 01 12 04 11 02

12 0C 14 09 00 0D 01 0A 00 00

後は、アセンブル後のマシン語プログラムを「General Virtual Machine」で実行すれば良い。

このプログラムの場合、実行すると19番地の変数SUMが55（0x37）になっているはずである。

実行環境

Windows 98/NT/2000/XP

開発環境

Microsoft Visual C++ 6.0
Microsoft Visual C++ .Net 2003
Microsoft Visual C++ 2005