

p88アセンブラ&シミュレータの説明 (for Version 1.0.2, 2007.03.02)

1. はじめに

p88アセンブラは、Alan W. Biermann の「やさしいコンピュータ科学」(アスキー出版局)⁽¹⁾の中で入門学習用に説明されている。このテキストの記述をもとに、アセンブリ言語として動作させることのできるシミュレータを開発した。シミュレータを開発するにあたり、極めてシンプルな仮想マシンとその命令コードを設定した。

以下では、プログラムの書き方、実行方法から、拡張された機能までを説明する。

(1) 原著は "Great Ideas in Computer Science", The MIT Press, 1990.

2. 簡単なプログラムの実行

以下のプログラムをテキストエディタで作成しよう。ラベル "LAB" のある行は、先頭が空白ではなく、必ず "L" で始まらないといけない。その他の行は先頭を空白かタブで始めて、ラベルがないことをはっきりさせなければならない。区切りには空白かタブを使う。文字は大文字でも、小文字でもよい。

```
IN      AX
IN      BX
CMP     AX, BX
JB      LAB
COPY    AX, BX
LAB     OUT    AX
```

ファイルの拡張子は何でもよい（なくてもよい）。仮にファイル "ex.p88" に保存されているとする。

p88シミュレータがカレントディレクトリにあるとすると、以下のコマンドでプログラムを実行できる（% はプロンプト）。このプログラムは端末から2つの数値を入力して（下線で表している）、小さい方を出力する。

```
% ./p88 ex.p88
in ? 67
in ? 43
out: 43
>>> Program terminated. (Halt)
```

プログラムは大文字、小文字を区別しない。ラベル名は先頭が英文字、2文字目以降は英数文字を使うが、7文字までしか認識しない。たとえば、"FUNCTION" と "FUNCTION2"、"FUNCTIONO" は8文字目以降が異なるために、シミュレータでは区別できず、同じものとして扱われる。

命令、レジスタ名、その他システム定義の名前はラベル名としては使えない。システム定義の名前には以下のものがある。

```
COPY, ADD, SUB, MUL, DIV, CMP, JMP, JNB, JB, IN, OUT,
CALL, RTN, PUSH, POP, END, AX, BX, CX, DX, RES, RANDOM
```

数値の記述には、今のところ10進数のみが利用できる。コメントは ";" で書き始め、行末までの任意の文字列がコメントとして無視される。

3. プログラム実行上の注意

上記のプログラムは暗黙のうちに先頭行から実行され、ファイルの末尾で実行が終了していることに注意して欲しい。p88シミュレータでは、プログラムはファイルの先頭（アドレス=0）から実行が開始

される。プログラムは END 命令で終了する。p88シミュレータは、読み込んだプログラムが END 命令で終わっていない場合、自動的に END命令を追加する。

また、メモリ空間外のアドレスにアクセスしようとすると、エラーとなって停止する。その他、0による除算、不正な命令の実行、端末からの割り込みで停止する。従って、プログラムで無限ループを実行して、止まらなくなってしまった場合には Control-C の入力で停止できる。

命令語とデータには区別がなく、命令語も実行中に自由に書き換えができる。

以下のプログラムはテキストに掲載されているものである。ここで ";" から行末まではコメントとして無視される。このプログラムは小文字で記述されているが、大文字で記述した場合と変わりはない。カンマ "," の前後に空白を入れても構わない。

```
;; Chapter 9 sample program in the text
    in      ax
    copy    m1, ax ; m1?
    mul     ax, m1
    out     ax
```

プログラムで使われているラベル "m1" は定義されていないが、そのまま実行してみよう。

```
% ./p88 sample02.p88
!!Undefined Label ... Memory allocated: M1 = [9], Line=3
in ? 100
out: 10000
>>> Program terminated. (Halt)
```

シミュレータが m1 のメモリを自動的に確保し、実行が行われている。p88シミュレータは未定義なラベルがあると自動的にメモリを2バイトずつ割り当てる。この方法で常に実行可能であるわけではないが、テキストに掲載されているいくつかのプログラムをそのまま実行するためにこのような動作をするようになっている。もちろん、実際のアセンブラのプログラムでは、未定義のラベルが残っているのは実行できない。自分でプログラムを記述する際は未定義ラベルが残らないようにすべきであり、END命令も明記しなければならない。

たとえば以下のように記述する。命令の代わりに10進数を記述すると、その値を保存するためのメモリ領域（2バイト）が確保される。

```
;; Chapter 9 sample program in the text
    in      ax
    copy    m1, ax
    mul     ax, m1
    out     ax
    end
m1         0
```

4. 仮想マシンに関する注意

仮想マシンではバイト単位のアдресのみが使われる。原著では命令語がワード単位であることを想定しているが、現在はマイクロプロセッサ向けのアセンブリプログラムの機会が多いと考えられるため、バイトコードによって実装した。ひとつの命令は1～3バイトから構成される。

アドレスはバイト単位だが、データは2バイトからなるワード（16ビット符号付き整数）としてアクセスする。COPY命令による読み出しや格納、算術演算、比較演算はワード単位である。ただし、データをワード境界に揃える必要はない。バイトを扱う命令はない。

アドレス空間は12ビット（4096バイト）であり、ビッグエンディアンで実装されている。

レジスタは AX から DX までの4つが使える。いずれも 16ビット符号付き整数である。

5. ディレクティブ

プログラム内には、機械語命令を生成するための記述だけではなく、アセンブラに対するさまざまな指示を記述できる。これをディレクティブ（擬似命令）と呼ぶ。ディレクティブには、メモリ領域の確保、定数やマクロの宣言、メモリモデルやセグメントの指定など数多くの種類がある。

p88シミュレータには、変数として使うことのできるメモリ領域を確保するためのディレクティブのみが用意されている。

(1) 初期値を持つメモリ領域の確保

整数値、またはラベルを記述することによって、16ビット符号付き整数を初期値として持つメモリ領域を2バイト分確保する。ラベルを付けることもできる。

以下の例では、TABLE というラベルで表されるメモリ位置に初期値として28を持つ2バイトが確保される。引き続き2バイトの初期値は29である。さらに、PTR というラベルで表されるメモリ位置には、初期値として TABLE のアドレスが格納される。

```
TABLE 28
      29
PTR   TABLE
```

(2) 指定した大きさを持つメモリ領域の確保

メモリ領域の大きさをバイト単位で指定して確保するために、RES というディレクティブを新設した。確保した領域の初期値は0である。以下の例では、BUF というラベルで表されるメモリ位置から、20バイト（10ワード）の連続した領域を確保する。

```
BUF   RES      20
```

6. 即値形式について

原著には、1から10までを連続して出力する、以下のようなプログラム例が掲載されている。

```
      COPY     AX,M0
L1     ADD     AX,M1
      OUT     AX
      CMP     AX,M10
      JB      L1
      END
M0     0
M1     1
M10    10
```

一般のプログラムでは、値を1つずつ増やしたり、0と比較したりすることが頻繁に行われる。このプログラムのように値1や値0を保持するメモリ領域を用意しておくのはわずらわしく、プログラムが記述しにくい。そこで、多くのアセンブリ言語には、命令語の一部に値を含めておく即値（イミディエイト）という形式が用意されている。p88アセンブラには、原著では即値形式の命令は用意されていないが、プログラミングの容易さを考えて即値形式を加えた。

即値形式では、アドレスを記述する部分に"#1"のように"#"から始まる整数値を記述する。指定できる数値は8ビット符号付き整数（-128～127）に限定される。この範囲にない数値は上のプログラムのようにメモリに値を保持しておいて利用することになる。即値形式が利用できるのは COPY、CMP、および ADD などの算術演算である。

即値形式を用いて上のプログラムを記述しなおしてみると、次のようになる。COPY 命令ではメモリから値を取り出す代わりに、命令語に含まれる数値をレジスタに直接入れることができる。ADD 命令ではレジスタの値をインクリメントするという動作が簡単に記述できている。

```
      COPY     AX,#0
L1     ADD     AX,#1
      OUT     AX
```

```

CMP      AX,#10
JB       L1
END

```

7. アドレッシングモード

アセンブリプログラムで値を扱う時、その値を指定する方法は何通りか存在する。p88シミュレータでは一般の計算機アーキテクチャでも広く利用されている以下の方法が使える。

(1) レジスタ

レジスタを指定して、そのレジスタに格納されている値を利用する。以下の COPY 命令ではレジスタ BXの値をレジスタAXにコピーしている。

```

COPY     AX,BX

```

(2) 即値

命令語内に指定された値をそのまま利用する。p88シミュレータでは8ビット符号付き整数に限定している。以下の COPY 命令では値0をレジスタAXにコピーしている。

```

COPY     AX,#0

```

(3) 直接指定メモリ

指定されたアドレスに格納されている値をメモリから取り出して利用する。以下ではメモリ位置 HGT から値を取り出してレジスタAXに加算し、その結果をメモリ位置 SUMにコピーしている。

```

ADD      AX,HGT
COPY     SUM,AX

```

(4) レジスタ間接指定メモリ

指定したレジスタに格納されている値をアドレスと見なし、そのアドレスに格納されている値をメモリから取り出して利用する。レジスタから値を取り出した後で、さらにそのアドレスから値を取り出すことから「間接」と呼ばれる。p88アセンブラでは c(AX) という記法を用い、COPY命令でのみ利用できる。"c"と次の"("の間に空白を入れてはいけない。

以下のプログラムは5つのデータを読み込み、読んだのとは逆順に書き出す。レジスタAXには最初、メモリ位置 WORKのアドレスが格納される（メモリ位置 PTRにアドレスが格納されている）。読み込んだ値はレジスタAXではなく、WORKに格納される。繰り返しのたびにレジスタ AXの値は2バイト分ずつ増やされ、値の格納先はWORKの次、その次、というように移動してゆく。

```

LAB1  COPY     AX,PTR
      IN       BX
      COPY     c(AX),BX
      ADD     AX,#2
      COPY     BX,CNT
      ADD     BX,#1
      CMP     BX,#5
      JB      LAB1
      COPY     BX,#0
      COPY     CNT,BX
      SUB     AX,#2
LAB2  COPY     BX,c(AX)
      OUT     BX
      SUB     AX,#2
      COPY     BX,CNT
      ADD     BX,#1
      CMP     BX,#5
      JB      LAB2

```

```

        END
CNT      0
PTR      WORK
WORK     RES      10    ; 5 data

```

(5) ディスプレイスメント付きレジスタ間接指定メモリ

指定したレジスタに格納されている値にディスプレイスメント（変位値）を加えた結果をアドレスと見なし、そのアドレスに格納されている値をメモリから取り出して利用する。p88アセンブラでは `mem+c(AX)` という記法を用いる。ここで `mem` がディスプレイスメントであり、アドレスまたは数値で記述する。COPY 命令でのみ利用できる。

```

COPY     BX,WORK+c(AX)
COPY     BX,2+c(DX)

```

上の例で、ラベル `WORK` がアドレス 100 番地を表し、レジスタ `AX` に整数値 2 が格納されているとすると、レジスタ `BX` にはアドレス 102 番地の値がコピーされる。また、レジスタ `DX` にアドレス 100 番地が格納されているとすると、レジスタ `BX` にはやはりアドレス 102 番地の値がコピーされる。

上記の (4) のプログラムを、このアドレッシング方式を使って書き直してみる。

```

LAB1     COPY     AX,#0
        IN        BX
        COPY     WORK+c(AX),BX
        ADD      AX,#2
        CMP      AX,#10
        JB       LAB1
LAB2     SUB      AX,#2
        COPY     BX,WORK+c(AX)
        OUT      BX
        CMP      AX,#1
        JNB      LAB2
        END
WORK     RES      10    ; 5 data

```

8. スタック操作

p88 シミュレータでは簡単なスタック操作が扱える。スタック操作は原著の p88 アセンブラには存在していないが、ハードウェアレベルでスタック操作が可能となっている場合も多いため、簡単な操作のみを実装した。これにより、サブルーチンコールの実現が容易となった。

(1) スタック領域とスタックポインタ

p88 シミュレータのメモリ空間は 12 ビットであり、命令コードはアドレス 0 から順番に配置される。このメモリ空間の末尾（アドレスの高い）部分がスタック領域として利用され、スタックはアドレスの低い方へ向かって伸張する。

スタックポインタとしてはレジスタ `DX` を使う。レジスタ `DX` はプログラム開始時にはスタックが空の状態を表すように初期化され、その後のスタック操作に伴って値が増減する。プログラムでスタック操作を行わない場合、レジスタ `DX` は他のレジスタと同様に汎用レジスタとして利用できる。

(2) PUSH と POP

PUSH 命令はレジスタを指定して、そのレジスタの値をスタックに格納する。具体的には、まずスタックポインタであるレジスタ `DX` を 2 バイト分デクリメントして、その位置に値を格納する。

POP 命令はスタックから値を取り出して、指定したレジスタにその値を格納する。具体的には、まずスタックポインタであるレジスタ `DX` が指すアドレスから値を取り出し、その後でレジスタ `DX` を 2 バイト分インクリメントする。

以下の例では、サブルーチンコールの前にレジスタ `AX` と `BX` の値をスタックに退避し、その後サブルーチンでの処理によってレジスタ `AX` に戻された値をメモリ `VAL` に保存し、退避しておいたレジスタ `AX` と `BX` の値を元に戻している。PUSH と POP の順番が逆になることに注意。

PUSH	AX
PUSH	BX
CALL	FUNC1
COPY	VAL,AX
POP	BX
POP	AX

(3) CALLとRTN

CALL命令はその時のプログラムポインタの値（CALL命令のアドレス+2）をスタックに退避し、指定されたアドレスに分岐する。

RTN命令はスタックから値を取り出し、その値のアドレスに分岐する。従って、CALL命令に対応するRTN命令を実行すると、CALL命令の直後から実行を再開できる。

9. シミュレータの機能

シミュレータを起動する際、以下のようなオプションを指定できる。

`./p88 [オプション] ソースファイル`

オプション

- t トレースモードで実行
- c 文法チェックのみ（実行しない）
- d 機械語（中間コード）を表示
- h このヘルプを表示

-d オプションと -t を指定して実行させた場合の例（一部）を示す。

トレースモード（-tを指定）で実行すると、実行される命令がすべて表示される。左端の数値は実行される命令のアドレス、右側の数値は命令を実行する直前のレジスタ AX~DXの値、および条件フラグ (CF)の値である。アドレスはラベルではなく、[] で囲んだ数値で表している。

```
% ./p88 -d -t t08b.p88
----- MEMORY DUMP -----
10 00 e1 1d c0 17 20 02 60 0a 90 02 30 02 1d 80
17 f1 60 01 80 0c 00 00 00 00 00 00 00 00 00 00
00
-----
. . . . . 0: COPY AX,#0 0 0 0 4094 NB
. . . . . 2: IN BX 0 0 0 4094 NB
in ? 4
. . . . . 3: COPY [23]+c(AX),BX 0 4 0 4094 NB
. . . . . 6: ADD AX,#2 0 4 0 4094 NB
. . . . . 8: CMP AX,#10 2 4 0 4094 NB
. . . . . 10: JB [2] 2 4 0 4094 B
. . . . . 2: IN BX 2 4 0 4094 B
```

プログラムが正常に END命令で停止するとシミュレータは次の表示を出して終了する。

```
>>> Program terminated. (Halt)
```

括弧内が "Halt" 以外の表示の場合、プログラムは異常終了している。

out of memory	データ領域を実行しようとした
illegal code	不正な命令を実行しようとした
illegal operand	オペランドの形式が正しくない
illegal memory access	メモリ空間外へのアクセスが発生した
div by zero	0で除算しようとした
unexpected EOF	EOF (Control-D) が入力された

また、実行中に端末から割り込み（Control-C）が発生した場合、シミュレータは次のような表示を出して停止する。"PC=" で示されているのは、割り込みが発生した時に実行していた命令のアドレスである。

```
!!>>> Program is interrupted. PC=18
```

10. その他

原著にある RANDOM というアドレスが使える。このアドレスから値を読み込むたびに 0 ～ 32767 の範囲の異なる値（乱数）が得られる。

原著では NO-OP という何もしない命令も紹介されているが、これは「ラベルを付ける場所を確保するためだけに」導入されたものであり、このアセンブラ&シミュレータには実装されていない。このアセンブラではラベルのみの行を記述してひとつのアドレスに複数のラベルを対応させることができる。次の例ではラベル M1 と M2 は同じアドレスを表す。

```
                CMP    AX,CN1
                JB     M1
                ADD    AX,ONE
M1
M2      ADD    BX,TWO
                CMP    BX,CN2
                JNB    M2
```

11. 命令のまとめ

以下で、AX, BX とある位置には任意のレジスタを記述できる。memには数値またはアドレスを表すラベルを記述する。#8 には "#" に引き続いて8ビット符号付き整数（-128 ～ 127の範囲の数値）を記述できる。

COPY AX,BX	レジスタBXの内容をレジスタAXにコピー
COPY AX,mem	アドレスmemのメモリ内容をレジスタAXにコピー
COPY AX,c(BX)	レジスタBXの値をアドレスとするメモリ内容をレジスタAXにコピー
COPY AX,mem+c(BX)	memにレジスタBXの値を加算したものをアドレスとし、その内容をレジスタAXにコピー。
COPY AX,#8	8ビットの符号付き整数をレジスタAXにコピー
COPY mem,AX	レジスタAXの値を、アドレスmemのメモリに格納。
COPY c(BX),AX	レジスタAXの値を、レジスタBXの値をアドレスとするメモリに格納。
COPY mem+c(BX),AX	memにレジスタBXの値を加算したものをアドレスとするメモリに、レジスタAXの値を格納。
ADD AX,BX	レジスタAXにレジスタBXの内容を加える。結果はAXに残る。
ADD AX,mem	レジスタAXにアドレスmemのメモリ内容を加える。結果はAXに残る。
ADD AX,#8	レジスタAXに8ビットの符号付き整数を加える。結果はAXに残る。
SUB、MUL、DIV	これらはADDと同じ形式で、それぞれ減算、乗算、除算を行う。 ただし、0で除算をすると実行時エラーとなってプログラムは停止する。
CMP AX,BX	レジスタAXとレジスタBXの内容を比較。
CMP AX,mem	レジスタAXとアドレスmemのメモリ内容を比較。
CMP AX,#8	レジスタAXに8ビットの符号付き整数を比較。 これらの結果、AXの方が小さい場合、CFはB(Below)になる。AXの方が大きい場合、CFはNB(Not Below)になる。

JMP mem	無条件でアドレスmemに分岐。
JNB mem	CFがNBの場合、アドレスmemに分岐。
JB mem	CFがBの場合、アドレスmemに分岐。
CALL mem	CALL命令の直後のアドレスをスタックに保存し、アドレスmemに分岐。
RTN	スタックから取り出したアドレスに分岐。オペランドなし。
PUSH AX	レジスタAXの内容をスタックに保存する。
POP AX	スタックから取り出した値をレジスタAXに格納する。
IN AX	端末からレジスタAXに整数（16ビット符号付き）を読み込む。
OUT AX	レジスタAXの内容を端末に表示する。
END	プログラムを停止させる。オペランドなし。

ディレクティブ

mem	数値 memを初期値とする2バイト領域を確保する。
RES n	n（数値）バイトの連続するメモリ領域を確保する。初期値は0。

著作権について

このソフトウェアの著作権は荻原剛志にある。自由に利用、再配布して構わないが、このソフトウェアの利用に伴って発生した損害に関して、作者は責任を負わないとする。

(C) 2007, OGIHARA Takeshi