ディスアセンブラ作成

まず学習の第一段階として，ディスアセンブラを作ってもらいます.ディスアセンブラとは,実行ファイル(バイナリ)をアセンブリ言語に直すプログラムのことです.

アセンブリ言語とバイナリは 1:1 に対応しています.

この対応は CPU の仕様によって決まっていて,その仕様書を見ながら，ディスアセンブラを作成してもらいます.

今回学習する CPU は 8086 という x86 アーキテクチャの元となる CPU をもちいます.また学習する OS として minix という OS を用います.

これらの CPU,OS 用にコンパイルされたバイナリをディスアセンブルします.

まず，配布した問題の 1.s をコンパイルしてみましょう.

コンパイルには以下のコマンドを用いてください.

$m2cc -.o 1.s

これで,a.out という実行ファイルが作成されました.

それでは,その中身がどのようなものになっているか確認してみましょう.

今回は hexdump というコマンドを用いますが他のコマンドやバイナリエディタを用いても構いません.

以下の様に使用します.

$hexdump (–C) a.out

```
0000000 01 03 20 04 20 00 00 00 10 00 00 00 26 00 00 00
0000010 00 00 00 00 00 00 00 00 00 00 01 00 70 00 00 00
0000020 bb 00 00 cd 20 bb 10 00 cd 20 00 00 00 00 00 00
0000030 01 00 04 00 01 00 06 00 00 00 20 00 00 00 00 00
0000040 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00
0000050 68 65 6c 6c 6f 0a 6d 65 73 73 61 67 65 00 00 00
```

これが a.out ファイルの中身です.

なお,今回は minix の実行ファイルなので, 以後の説明はその条件のもとでの説明です.

その実行ファイルは大きく分けて,ヘッダ,text(命令),data の 3 つの部分に分かれます.

まず,最初 32byte がヘッダにあたります.今回の例でいうと.

01 03 20 04 20 00 00 00 10 00 00 00 26 00 00 00
00 00 00 00 00 00 00 00 00 00 01 00 70 00 00 00.

の部分がヘッダにあたります.

そのファイルがどのようなものなのか(8086,minix のバイナリですといった情報など),このヘッダから判断できます.

また,上で挙げた text や data の大きさもこのヘッダに含まれています.

ディスアセンブラの作成で必要な情報は text の大きさのみなので,今回は text の大きさを取得してみます.

(他の値がどのような情報か知りたい方は /usr/local/minix2/usr/include/a.out.h を見てください)

text のサイズは 0 バイトから初めて 8~11 の 4 バイトに入っています.この場合は 10 00 00 00 です.

バイナリの並びには２種類あり，リトルエンディアンとビッグエンディアンが存在し,そのエンディアンによってダンプした形が変わってきます.

16 進数で 12345678　というデータの場合

78 56 34 12 のように逆順にメモリに格納する並びを.リトルエンディアンと呼び,12 34 56 78 のようにそのままの並びをビッグエンディアンとよびます.

エンディアンについては,ここでは詳しく解説しませんが

intel 系の CPU ではリトルエンディアンが使われていて,

ヘッダの 8~11byte は 10 00 00 00,

つまり，ヘッダ 32byte の後に 00 00 00 10(10 進数で 16)byte の text があるということです.

ディスアセンブラの作成ではこの text から 1byte ずつ読み取って，その命令を解釈していきます.

上の例の実行ファイルでは

bb 00 00 cd 20 bb 10 00 cd 20 00 00 00 00 00 00

が text に当たるので, このバイナリを解釈して,アセンブラに変換します.

今回は配布したツールにディスアセンブルするプログラムがあるので,実際にディスアセンブルしてみましょう.

$m2cc -.o 1.s　(a.out 作成)

$7run –d a.out

0000: bb0000　　mov bx, 0000

0003: cd20　　int 20

0005: bb1000　　mov bx, 0010

0008: cd20　　int 20

000a: 0000　　add [bx+si], al

000c: 0000　　add [bx+si], al

000e: 0000　　add [bx+si], al

これがディスアセンブル結果です.

みなさんにはこれと同じ物を, 出力を参考にしながら作っていただきます.

とわいえ,なんのヒントもなしで作るのは困難であるため簡単な例でその作り方を説明します.

bb 00 00 : mov bx,0000 を仕様書で確認してみましょう.

Mov　:　Immediate to Register

1	2	3　(byte)
1011wreg	data	data (if w＝1)

これは即値をレジスタにコピーする命令なのですが,今はその命令に関して考える必要はありません.

上の w や reg などが何に用いられているかはデータシートの NOTES:に書いてあります.

w = 1 の時,データを word 長(2byte),w = 0 の時は ,byte 長(1byte)で扱います.

この例で bb0000 を 2 進数に直すと

10111011　00000000　00000000 であり,

1011wreg　data　data (if w=1)

と比べると w = 1 reg = 011 であるわかります.

w = 1 のため,data は word 長(リトルエンディアン)だとわかり,合計 3byte であると判断できます.

また, reg = 011 && w = 1 より

REG is assigned according to the following table

を参照して bx というレジスタ(2byte の memory のようなもの)を用いることがわかります.

このようにして mov bx 0000 だと判定しています.

w,reg 以外にも様々な値があるので, その都度その値がどのような値であるか,下の NOTES を読み理解してください.

このデータシートを用いて，ディスアセンブルするプログラムを作成し，7run –d と同じ出力になるようにしていきます．

ディスアセンブルするプログラムとして，
1.s~3.s, 1.c~6.c, nm の 10 個のプログラムを用意しました．
これらのプログラムをディスアセンブルできるようにしてください．

このディスアセンブラの作成が終わったら今度はそれを利用して，仮想マシン(エミュレータ)を実装していきます．

なお,データシートには誤植があります.

誤 baa → 正 daa ,誤 ssb → 正 sbb ,

Immediate with Accumulator:

誤 000111w → 正 0001110w

データシート主な略語は以下の通りです.

reg　→　register

DISP → displacement([bx+2]の+2 のこと)

mod → mode (r/m の振る舞いを決める)

r/m　→ register or memory(mod によって振る舞いを変える)