

レポート提出票

科目名: 情報工学実験1

実験課題名: 課題2 アセンブリ言語

実施日: 2020年 6月 1日

学籍番号: 4619055

氏名: 辰川力駆

共同実験者: _____

1 実験の目的

機械語と 1 対 1 に対応していながら、人間が読み書きしやすいように作られたアセンブリ言語を学ぶ。アセンブリ言語を用いて PC の CPU である Intel x86 を機械語で直接操作することによって、CPU や周辺機器の動作についての理解を深める。

2 実験の内容

アセンブリ言語に慣れるために、シミュレータを使って (擬似的な) レジスタの値やフラグの値をチェックする。その次に、8086 に対応した MS-DOS という 16 ビット OS を対象として、実際に 8086 をアセンブリ言語で操作する。

3 実験環境

- MacBook Pro(16-inch,2019)
 - ProductName: Mac OS X
 - ProductVersion: 10.15.5
 - BuildVersion: 19F101
 - プロセッサ: 2.6 GHz 6 コア Intel Core i7
 - メモリ: 16 GB 2667 MHz DDR4
- NASM version: 2.14.02
- DOSBox version: 0.74-3

4 結果・考察

4.1 レポート課題 1

課題 1

本課題では“シミュレータ”と“エミュレータ”を用いて実験を行った。“シミュレータ”と“エミュレータ”とは何かについて調査し、レポートにまとめよ。その際に“シミュレータ”と“エミュレータ”の違いを具体例を引用しながら説明せよ。

エミュレータは、1 つのコンピュータシステムが他のシステムと同じように動作することを可能にするハードウェアまたはソフトウェアのことである。シミュレータは、コンピュータのシステムなどを模擬的に再現する機能を持ったハードウェアまたはソフトウェアのことである。つまり、全く同じ動作をするか模擬的であるかの違いである。

具体例あげると、本実験ではシミュレータとして 16bit Assembler Simulator を使用し、エミュレータとして DOSBox を使用した。16bit Assembler Simulator の場合、文字列を出力したり、キー入力をすることはできない。これに対し、DOSBox はこれらができる他に、他の OS(Windows や Linux など) 同様、システムコール全般が扱える。

4.2 レポート課題 2

課題 2

課題 1-2 で作成したプログラムについて、どのようなやり方で 1 から 10 までの和を求めたのかを、適宜ソースコードを引用しながら説明せよ。また、プログラム作成時に工夫した点・気をつけた点についても述べよ。

レポートには作成したソースコード (sum_4619055.asm) を載せること。また、ソースコードの各行に行番号を付けること。

ソースコード 1: sum_4619055.asm

```
1      MOV AX,0
2      MOV CX,10
3      label:
4          ADD AX,CX
5          LOOP label
6      HLT
```

このソースコード 1 では、繰り返しジャンプ命令である LOOP を用いて AX レジスタに 1 から 10 までを逆順に加算したものである。

具体的には、2 行目で LOOP をする回数を CX レジスタに保存し、次の行 (3 行目) に LOOP したいラベル (今回は label) を書き、そのラベル内で繰り返しジャンプ命令である LOOP を書くことで、CX レジスタ回ラベルが実行されている。LOOP 命令では 1 回実行されるたび CX レジスタは -1 されていくので、4 行目で AX レジスタに CX レジスタを加えているが、10, 9, 8, ..., 1 という順番で加えられている。

工夫した点は、上記で述べたように LOOP を用いたことである。1 から 10 を足すプログラムを作るだけであれば LOOP 演算を使わず、4 行目のように加算命令 ADD を用いて 1 から 10 を 10 行に渡り書くこともできたが、コードが簡潔にならないので LOOP を使用した。

4.3 レポート課題 3

課題 3

課題 1-3 において、AX レジスタに保存した数値を 2 進数で表現したときに 1 になるビットの数を求めるプログラムを作成した。どのようなやり方で 1 の数を数えたのか、適宜ソースコードを引用しながら説明せよ。また、プログラム作成時に工夫した点・気をつけた点についても述べよ。

レポートには作成したソースコード (nbits_4619055.asm) を載せること。また、ソースコードの各行に行番号を付けること。

ソースコード 2: nbits_4619055.asm

```
1      MOV AX,80 ;学籍番号は 4619055なので 55+25=80です。
2      loop:
3          MOV BX,AX
4          AND BX,1 ;2進数表記で 1の位が 1ならば
                BX を 1 に、0 ならば BX を 0 にします
```

```

5      ADD DX,BX
6      SHR AX,1
7      CMP AX,0
8      JE loopend
9      CALL loop
10     loopend:
11     HLT

```

4.4 レポート課題 4

課題 4

課題 2-2 において、じゃんけんプログラムを作成した。どのようなやり方でユーザーの手とコンピュータの手を決定したのか、適宜ソースコードを引用しながら説明せよ。また、本課題では、結果を画面に表示する順番を指定した。指定された順番で結果を表示するために工夫した点・気をつけた点についても述べよ。さらに、作成したプログラムの問題点を一つ取り上げ、その解決策について考察せよ。

レポートには作成したソースコード (rps2_4619055.asm) を載せること。また、ソースコードの各行に行番号を付けること。

ソースコード 3: rps2_4619055.asm

```

1      org 0x100
2      mov bx,0 ; アドレステーブル参照のオフセットを示す変数
3
4      call firstmsg ; 最初のメッセージを表示
5      call inputfunc ; 文字列入力
6      call rsltuser ; user の結果を表示
7      call keywait ; キー入力が入ると CPU のじゃんけんがきまる
8      call rsltcpu ; CPU の結果を表示
9
10     ; プログラムの終了
11     mov ax,0x4c00
12     int 0x21
13
14     keywait:
15         add bx,2
16         cmp bx,4
17         jle skip
18         mov bx,0
19     skip:
20         ; キーボードの状態のチェック
21         mov ah,0x06
22         mov dl,0xff
23         int 0x21
24         jz keywait ; キー入力が無ければ keywait へ戻る
25         RET
26
27     firstmsg:

```

```

28         ;最初のメッセージを表示
29         mov ah,0x09
30         mov dx,msg
31         int 0x21
32         RET
33
34     rsltuser:
35         ;user の結果を表示
36         add bx,ax
37         add bx,ax
38         and bx,0x0f
39         mov ah,0x09
40         mov dx,usrmsg
41         int 0x21
42         mov dx,[handsign+bx]
43         int 0x21
44         mov bx,0
45         RET
46
47     rsltcpu:
48         ;CPU の結果を表示
49         mov ah,0x09
50         mov dx,cpumsg
51         int 0x21
52         mov dx,[handsign+bx]
53         int 0x21
54         RET
55
56     inputfunc:
57         ;文字列入力
58         mov ah,0x09
59         mov dx,input ;input メッセージを表示
60         int 0x21
61         mov ah,0x01 ;文字列入力
62         int 0x21
63         mov ah,0x09
64         mov dx,crlf ;改行を表示
65         int 0x21
66         RET
67
68     msg db 'Please␣Input␣Handsign␣(Rock->0,␣Paper->1,␣Scissors->2)',0x0d,0x0a
69         , '$'
70     input db 'INPUT:',0x0d,0x0a,'$'
71     usrmsg db 'User:␣','$'
72     cpumsg db 'CPU:␣','$'
73
74     handsign dw rock,paper,scissors ;アドレステーブル (ポインタが入った配列)
75     rock db 'Rock',0x0d,0x0a,'$'
76     paper db 'Paper',0x0d,0x0a,'$'
77     scissors db 'Scissors',0x0d,0x0a,'$'
78     crlf db 0x0d,0x0a,'$'

```

4.5 レポート課題5

課題5

fib.asm は、フィボナッチ数列を求めるプログラムである。フィボナッチ数列は、 $F_0 = 0$ 、 $F_1 = 1$ として、漸化式

$$F_{n+2} = F_n + F_{n+1} (n \geq 0)$$

によって得られる数列である。fib.asm の場合は $F_0 = 2$ 、 $F_1 = 3$ としている。この fib.asm には同じ処理が繰り返し記述されている箇所や非効率的に書かれた箇所が存在する。そこで、サブルーチン (CALL 命令・RET 命令) および LOOP 命令を用いて、以下のプログラムをより少ないコード量になるように書き換えよ。ただし、結果は DX レジスタに保存することとする。また、プログラムを実行した結果が変わらないようにすること。ファイル名は、fib_4619055.asm とする。レポートには作成したソースコードを載せること。また、ソースコードの各行に行番号を付けること。

レポートでは、作成したプログラムについて、どのような工夫をすることでソースコードをより短いコードに書き換えたのかについて述べよ。また、なぜ fib.asm では、9 つめのフィボナッチ数までしか求めていないのか、考察せよ。説明するときには適宜ソースコードを引用すること。

ソースコード 4: fib_4619055.asm

```
1      ;; フィボナッチ数列を求める
2      org 0x100
3
4      ;; 1つめのフィボナッチ数を計算 (5)
5      mov ah, 2
6      mov al, 3
7      add ah, al
8      loop: ; 2つめ~9つめのフィボナッチ数を計算
9              cmp al, 128 ; 2つ前の数が 128を超えたとき終了する
10             jae end
11             mov bl, ah
12             mov ah, al
13             mov al, bl
14             add ah, al
15             call loop
16
17     end:
18         mov dh, 0
19         mov dl, ah ; 結果をDL レジスタに保存
20         ;; プログラム終了
21         mov ah, 0x4c
22         int 0x21
```

5 結論

参考文献

[1] 東京理科大学工学部情報工学科 情報工学実験 1 2020 年度東京理科大学工学部情報工学科
出版

[2] エミュレータとシミュレータの違いは何ですか - との差

[URL]: <https://ja.strephonsays.com/what-is-the-difference-between-emulator-and-simulator>

最終閲覧日:2020/6/2

[3] LaTeX で色付きソースコードを貼り付け

[URL]: <http://yu00.hatenablog.com/entry/2015/05/14/214121>

最終閲覧日:2020/6/2