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 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プログラムである。

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、$10, 9, 8, \cdots, 1$ 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
```

このソースコード 2 は、論理演算命令である AND を用いて 1 の数を求め、DX レジスタに保存するプログラムである。

具体的には、1 行目のコメント文でもあるように、80 を 2 進数表記で表したときに 1 の数を求める。AX レジスタをコピーした BX レジスタを 4 行目で AND を用いて、BX レジスタの (2 進数表記で)1 の位が 1 ならば BX レジスタを 1 に、0 ならば BX レジスタを 0 にしている。その後、6 行目でシフト命令である SHR を用いることにより AX レジスタを (2 進数表記で) 右にずらすことと、9 行目でこの一連の動作を繰り返すことで、4 行目の動作を (2 進数表記で) $10, 100, 1000, \cdots$ の位の 1 の数を数えることができる。このままでは、9 行目によってずっと繰り返されるので 7 行目で比較する必要がなくなったら (AX レジスタが 0 になったら) 終了する。

工夫した点としては、9 行目でラベルを繰り返し行ったことである。しかし、これだけでは永遠に繰り返しをしてしまうので、抜け出す条件を書くことに気をつけた。

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 ;;キー入力がなければ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
                ,'$'
69          input db 'INPUT:',0x0d,0x0a,'$'
70          usrmsg db 'User:␣','$'
71          cpumsg db 'CPU:␣','$'
72
73          handsign dw rock,paper,scissors ;;アドレステーブル(ポインタが入った配列)
74          rock db 'Rock',0x0d,0x0a,'$'
75          paper db 'Paper',0x0d,0x0a,'$'
76          scissors db 'Scissors',0x0d,0x0a,'$'
77          crlf db 0x0d,0x0a,'$'
```

このソースコード3は、ユーザーに0から2を入力させ、それに応じて読み込むバイト型データテーブル(db)を変えることによって画面へ文字列を表示するプログラムである。また、今回のソースコードは2回目の授業で書いたもので、1回目の授業後にソースコードを小文字で書いて良いと先生に伺ったので、小文字で書いている。

具体的には、まず、4行目のcallによって27行目(ラベルfirstmsg)にジャンプする。そして、msgというバイト型データテーブルに保存されているメッセージを表示する。次に、5行目のcallによって56行目(ラベルinputfunc)にジャンプする。これにより、数字を入力するとbxレジスタに保存される。そしてラベルrsltuser内の42行目では、アドレステーブル(73行目)と入力してもらった数字を元に、どのデータテーブルを読み込むかを選定している。最後に、ラベルkeywaitでキー入力した時間に応じて数字がbxレジスタに保存されるこれにより、ラベルrsltcpu内でラベルrsltuserと同様な動作でコンピュータの手を決めることができる。

bxレジスタを2回使っているので、指定した順番で表示するために、ユーザーの方を表示してからラベルkeywaitでもう一度bxレジスタに数字を保存し、コンピュータの方を表示することに気をつけた。

また、今回作成したプログラムの問題点として、0から2以外が入力されるとバグが起きてしまうことが挙げられる。この問題点の解決策としてエラーハンドリングを施せばよいと考える。例えば、入力の段階で0から2以外が入力されたときは“Please enter a number from 0 to 2.”を表示し、プログラムが終了するようにすれば良い。

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
```

このソースコード 4 は、call を用いてラベルを繰り返し行うことにより、フィボナッチ数列を求めるプログラムである。

元のソースコード (fib.asm) からの変更点は 8 行目から 15 行目である。このラベル内で add を用いて 2 つ前と 1 つ前を足して ah レジスタに保存している。それを繰り返し行っているが、これだけだと終了しない。そこで 9 行目に注目してほしい。この行で抜け出す条件を書いているが、これは元のソースコード (fib.asm) が 9 つ目のフィボナッチ数までしか求めていない理由と関連している。ah レジスタは 8 ビットなので、255 までしか保存できない。それを超えてしまうとバグがおきてしまうので、元のソースコード (fib.asm) は 9 つ目のフィボナッチ数までしか求めていないと考える。今回のソースコードでも ah レジスタが 255 を超えないように 2 つ前が 128 を超えると終了するようにしている。２つ前が 128 を超えるともちろん 1 つ前も 128 を超えているので、その 2 つを加算したフィボナッチ数は 255 を超えてしまう。したがって、この条件で抜け出すことでバグが起きずに終了することができる。

5　結論

アセンブリ言語から、CPU や周辺機器の動作が分かり、コンピュータが動作する仕組みを本質的に理解することができた。また、授業で行っている C 言語や独学で勉強している Go 言語などの高級言語はとても人間が書きやすいように寄せていると分かった。

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

`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

`http://yu00.hatenablog.com/entry/2015/05/14/214121`

最終閲覧日:2020/6/2