システムプログラミング1
期末レポート

氏名: 山田　敬汰 (Yamada，Keita)
学生番号: 09430559

出題日: 2019 年 10 月 7 日
提出日: 2019 年?月??日
締切日: 2019 年 11 月 18 日

1　概要

本演習では，C 言語で書かれたプログラムを機械語に変換する際に必要不可欠となるアセンブラについて，いくつかの演習課題を解くことを通じて学びを深めた．具体的には，入出力機能の実装やアセンブリ言語内に登場する用語の解説，そして，C 言語で記述されているプログラムの再現（関数の実装，素数を表示するプログラム及びその保存方法の改造），を行った．以下に，今回の授業内で実践した 5 つの演習課題についての詳しい内容を記述する．

2　課題 1-1

2.1　課題内容

教科書 A.8 節 「入力と出力」に示されている方法と，A.9 節 最後「システムコール」に示されている方法のそれぞれで ”Hello World” を表示せよ．両者の方式を比較し考察せよ．

2.2　作成したプログラム（システムコール無し）

```
 1         .data
 2         .align 2
 3 msg:
 4         .asciiz "Hello World"
 5
 6         .text
 7         .align 2
 8 main:
 9
10         la      $a1, msg
11
12         .text
13         .align 2
```

```
14 putc:
15          lb      $a0, 0($a1)
16          lw      $t0, 0xffff0008         # $t0 = *(0xffff0008)
17          li      $t1, 1                  # $t1 = 1
18          and     $t0, $t0, $t1           # $t0 &= $t1
19          beqz    $t0, putc               # if ($t0 == 0) goto putc
20          sw      $a0, 0xffff000c         # *(0xffff000c) = $a0
21          addi    $a1, $a1, 1
22          bnez    $a0, putc
23          j       $ra                     # return
```

2.3 プログラムの解説

1-4 行目では，出力する文字列をあらかじめメモリ上に展開している．

10 行目では，先ほどメモリ上に展開した文字列の先頭アドレスをレジスタ $\$a1$ に読み込む．

15 行目では，$\$a0$ に，次に表示する文字のアドレスとして $\$a1$ の値を格納する．

16-18 行目では入出力機器の状態を取得し，入出力が可能かどうかを判定している．（最下位ビットの値との AND 演算を行う）

19 行目では，入出力機器が使用可能な状態でない場合に，*putc* ラベルまで戻る様にしている．（入出力可能になるまで何度も判定を行っている）

20 行目では，入出力機器の出力用のアドレスを指定し $\$a0$ の値を出力する．

21-23 行目では，$\$a1$ の持つアドレスを一つずらし，次の文字のアドレスを格納した後，$\$a0$ の値が 0 でなければ（*.asciiz* を使っているので，文字列の末尾のアドレス内には 0 が格納されている）*putc* ラベルに戻る．

2.4 作成したプログラム（システムコール有り）

```
 1          .data
 2          .align 2
 3  msg:
 4          .asciiz "Hello World"
 5
 6          .text
 7          .align 2
 8  main:
 9          la      $a0, msg
10          li      $v0, 4
11          syscall
12          j       $ra
13
```

2.5 プログラムの解説

1-8 行目についてはシステムコール無しの場合と違いはないので，説明を省略する．

9-10 行目ではシステムコールを呼び出すための下準備を行っている．具体的には，システムコールに与える引数である $a0 に文字列の先頭アドレスを与え，どのシステムコールを呼ぶかを制御するレジスタ $v0 に,「文字列を出力する」ことを意味する 4 を格納する

11 行目では実際にシステムコールを呼び出し，先ほど設定した $a0 と $v0 の値を見て,「$a0 を先頭アドレスとした文字列を出力する」という動作を OS が行う．

この時，入出力機器へのアクセスは全て OS を介して行っているので，プログラム側から直接入出力機器にアクセスすることはない．

2.6 考察

両者を比較した時に，システムコールを利用しないプログラムでは，入出力処理を行う際にアドレスを固定値で指定しているのに対し，システムコールを利用するプログラムでは，入出力処理はシステムコールにより OS 側に一任されているのでプログラム内で固定値のアドレスが指定されていない，という違いがある．この違いによって，システムコールを利用する方のプログラムは入出力装置についての情報を持っていなくても動作するので，入出力機器が変更されてもプログラムを（疎結合になっている）という利点がある．その他にも，OS 側が許可した動作しかできないようになっているので（各種機器とプロセッサ間を仲介し，互いが直接アクセスしないようにしている），アドレスを直接指定するよりもセキュリティ上のリスクを減らすことができる．

3 課題 1-2

3.1 課題内容

アセンブリ言語中で使用する .data， .text および .align とは何か解説せよ．下記コード中の 6 行目の .data がない場合，どうなるかについて考察せよ．

```
 1:          .text
 2:          .align  2
 3: _print_message:
 4:          la      $a0, msg
 5:          li      $v0, 4
 6:          .data
 7:          .align  2
 8: msg:
 9:          .asciiz "Hello!!\n"
10:          .text
11:          syscall
12:          j       $ra
13: main:
14:          subu    $sp, $sp, 24
15:          sw      $ra, 16($sp)
16:          jal     _print_message
17:          lw      $ra, 16($sp)
18:          addu    $sp, $sp, 24
19:          j       $ra
```

3.2 解答

.data，.text および .align とは「アセンブラ指令」と呼ばれるもので，主にプログラムの実行前に行われる前処理を制御するものである．つまり，これらの記述は機械語としてメモリに変換される訳ではない．その中でも，.data と .text はメモリ中のどこに機械語を配置するかを制御している．具体的には .data はデータをデータセグメントに配置することを指示し，.text はデータをテキストセグメントに配置することを指示している．何故，データセグメントとテキストセグメントを区別する必要があるのかというと，データセグメントの内容には読み込みと書き込みが両方行われるのに対し，テキストセグメントの内容は読み込みしか行われない（値が不変）ので，同一プログラムを他のプロセスで実行する時にテキスト部分を共有することができたり，読み込み専用領域にデータを置くことができる，というメリットがあるからである．

また .align はオプションとして整数 n を指定し，データが 2^n ずつの領域に確保される様に余白を取るための命令である．何故余白を取る必要があるのかというと，MIPS は 32 ビット（4 バイト）ずつデータを CPU とやり取りするため，余白を取って（この場合は $n = 2$）データを綺麗に整列させておくことで，アクセス回数を減らすことが可能となり，システムの効率化に繋がるからである．

最後に，上記のコード中の 6 行目の .data がない場合にどうなるかというと，?????????

4 課題 1-3

4.1 課題内容

教科書 A.6 節 「手続き呼出し規約」に従って，関数 fact を実装せよ．(以降の課題においては，この規約に全て従うこと) fact を C 言語で記述した場合は，以下のようになるであろう．

```
 1: main()
 2: {
 3:   print_string("The factorial of 10 is ");
 4:   print_int(fact(10));
 5:   print_string("\n");
 6: }
 7:
 8: int fact(int n)
 9: {
10:   if (n < 1)
11:     return 1;
12:   else
13:     return n * fact(n - 1);
14: }
```

4.2 作成したプログラム

```
1         .data
2         .align 2
3 msg:
```

```
 4         .asciiz "The factorial of 10 is "
 5 newline:
 6         .asciiz "\n"
 7
 8         .text
 9         .align 2
10 main:
11         move    $s0, $ra
12         li      $s1, 1
13         la      $a0, msg
14         jal     print_string
15         li      $a0, 10
16         jal     fact
17         move    $a0, $v0
18         jal     print_int
19         la      $a0, newline
20         jal     print_string
21         j       $s0
22
23 fact:
24         subu  $sp, $sp, 32
25         sw    $fp, 16($sp)
26         sw    $ra, 20($sp)
27
28         addu  $fp, $sp, 28
29         sw    $a0, 0($fp)
30
31         bgtz  $a0, Lthen
32         j     Lelse
33
34 Lthen:
35         subu  $a0, $a0, 1
36         jal   fact
37
38         lw    $v1, 0($fp)
39         mul   $v0, $v0, $v1
40         j     Lreturn
41
42 Lelse:
43         li    $v0, 1
44         j     Lreturn
45
46 Lreturn:
47         lw    $fp, 16($sp)
```

```
48          lw    $ra, 20($sp)
49          addiu $sp, $sp, 32
50          j     $ra
51
52 print_string:
53          li      $v0, 4
54          syscall
55          j       $ra
56
57 print_int:
58          li      $v0, 1
59          syscall
60          j       $ra
61
```

4.3 プログラムの解説

ここでは $fact$ 関数内で行われている処理に対応している部分について，処理の流れを追いながら詳細に解説する．

1. $\$sp$ の値を減算し，スタックを 32 バイト分確保した後，そこに $\$fp$ と $\$ra$，$\$a0$ の値を退避させる．(24-29 行目)
2. $\$a0$ の値を確認し，その値が 0 かどうかで処理を分岐する．(31-32 行目)
3. $\$a0$ の値が 0 より大きいので，$Lthen$ ラベルの場所にジャンプする．
4. $\$a0$ の値を一つ減らし，その後 $fact$ を再帰的に呼び出す（35-36 行目）
5. 1 から 4 までの処理を $\$a0$ の値が 0 になるまで繰り返す．
6. $\$a0$ の値が 0 になった時，3 のタイミングで $Lthen$ ラベルでは無く $Lelse$ ラベルの場所にジャンプする．
7. $\$v0$ に 1 を格納した後，$Lreturn$ ラベルの場所にジャンプする．(43-44 行目)
8. 直近にスタックに退避させた値（32 バイト分）をレジスタに復帰させた後，$\$sp$ の値を加算し，使用済みスタック領域の解放を行う．(47-49 行目)
9. $\$ra$ レジスタに格納されている，4 の $fact$ にジャンプする命令の次の命令のアドレスに移動する．(50 行目)
10. スタックに退避しておいた $\$a0$ の値を $\$v1$ に格納し，$\$v0$ の値を自身と $\$v1$ の積の値に上書きする．(38-39 行目)
11. $Lreturn$ ラベルの場所にジャンプする．(40 行目)
12. 8 から 11 までの処理を繰り返す．

13. 一番最初にスタックに退避させた $\$ra$ には一番最初に $fact$ を呼び出した時の次の命令のアドレスが格納されているので，このプログラムで確保したスタック領域を解放しきった後，9 のタイミングで 17 行目の命令に移動し，$fact$ の一連の処理が終了する．（計算結果は $\$v0$ に格納されている）

5 課題 1-4

5.1 課題内容

素数を最初から 100 番目まで求めて表示する MIPS のアセンブリ言語プログラムを作成してテストせよ．その際，素数を求めるために下記の 2 つのルーチンを作成すること．

- test_prime(n) n が素数なら 1，そうでなければ 0 を返す
- main() 整数を順々に素数判定し，100 個プリント

5.2 C 言語で記述したプログラム例

```
 1: int test_prime(int n)
 2: {
 3:   int i;
 4:   for (i = 2; i < n; i++){
 5:     if (n % i == 0)
 6:       return 0;
 7:   }
 8:   return 1;
 9: }
10:
11: int main()
12: {
13:   int match = 0, n = 2;
14:   while (match < 100){
15:     if (test_prime(n) == 1){
16:       print_int(n);
17:       print_string(" ");
18:       match++;
19:     }
20:     n++;
21:   }
22:   print_string("\n");
23: }
```

5.3 作成したプログラム

```
1     .data
```

```
 2     .align  2
 3
 4 newline:
 5         .asciiz "\n"
 6 space:
 7         .asciiz " "
 8
 9     .text
10     .align 2
11
12 test_prime:     # int test_prime(int n = $a0)
13
14 li $a3,2     # for のループ数カウンタ
15
16 for:
17       slt $v0,$a3,$a1         # if !(i < n)
18       beq $v0,$zero,return1 #   return 1
19
20       div $a1,$a3     # $a0 =
21 mfhi $v0     # $v0 = n % i
22 beq $v0,$zero,return0 # if ($v0 == 0) return 0
23
24       addi $a3, $a3, 1 # $a3 = $a3 + 0
25       j     for
26
27 while:
28 slt $v0,$a2,$a0     # if !(i < n)
29 beq $v0,$zero,return1 #   return 1
30
31       jal   test_prime
32       beq   $v0,$s1,match
33
34       addi $a1, $a1, 1 # $a1 = $a1 + 1
35 j while     # goto while
36
37 match:
38       move    $s2, $a0
39       move    $a0, $a1
40       jal     print_int
41       la      $a0, space
42       jal     print_string
43       move    $a0, $s2
44
45       addi   $a1, $a1, 1 # $a1 = $a1 + 0
```

```
46        addi   $a2, $a2, 1 # $a2 = $a2 + 0
47        j       while
48
49 main:
50        move    $s0, $ra
51        li      $s1, 1
52        la      $a0, 100
53        li      $a1, 2      # while のループ数カウンタ
54        li      $a2, 0      # match 数
55        jal     while
56        la      $a0, newline
57        jal     print_string
58        j   $s0 # jump to $s0
59
60
61 return0:
62        li $v0,0
63        j $ra
64
65 return1:
66        li $v0,1
67        j $ra
68
69 print_string:
70        li      $v0, 4
71        syscall
72        j       $ra
73
74 print_int:
75        li      $v0, 1
76        syscall
77        j       $ra
```

5.4 プログラムの解説

上記のプログラムの内容について，処理の流れを追いながら詳細に解説する．

1. 各種変数の初期化を行う．($a0：表示する素数の上限，$a1：while ループのループ数カウンタ，$a2：見つかった素数の数)（52-54 行目）
2. *while* ラベルジャンプする．(55 行目)
3. 見つかった素数の数が表示する上限を超えているかどうか確認する．($a2 と $a0 を比較している)
4. $a2 < $a0 なので，ループの中に入り *test_prime* ラベルにジャンプする．

5. $a1 が素数かどうかを確かめるために必要な変数 $a3（除数）を初期化する．
6. $a1 の値が $a3 で割り切れるかを確かめ，割り切れた場合は $v0 に 0 を格納し，*test_prime* ラベルの呼び出し元へ帰る．
7. 割り切れない場合は $a3 の値を加算し続けながら割り切れるかどうかを確かめ，割り切れないまま $a3 の値が $a1 の値までたどり着いた時，$v0 に 1 を格納し，*test_prime* の呼び出し元へ帰る．（$a1 は素数）
8. $v0 の値が 1 ならば（$a1 が素数ならば），*match* ラベルにジャンプした後，$a1 の値を画面に出力し，$a2 の値を加算する．
9. $a1 の値を加算し，3 まで戻る．
10. 3 から 9 までを繰り返し，$a2 の値が $a0 以上になった時，*while* ラベルの呼び出し元へ帰り，プログラムを終了する．

6 課題 1-5

6.1 課題内容

素数を最初から 100 番目まで求めて表示する MIPS のアセンブリ言語プログラムを作成してテストせよ．ただし，配列に実行結果を保存するように main 部分を改造し，ユーザの入力によって任意の番目の配列要素を表示可能にせよ．

6.2 C 言語で記述したプログラム例

```
 1: int primes[100];
 2: int main()
 3: {
 4:   int match = 0, n = 2;
 5:   while (match < 100){
 6:     if (test_prime(n) == 1){
 7:       primes[match++] = n;
 8:     }
 9:     n++;
10:   }
11:   for (;;){
12:     print_string("> ");
13:     print_int(primes[read_int() - 1]);
14:     print_string("\n");
15:   }
16: }
```

7　感想

今回の課題を進めていく上での感想としては，アセンブリ言語でプログラムを書こうとすると簡単な処理でも冗長なソースコードを書かなければならないので，コンピュータの動作を一歩一歩丁寧に追えるようになっていることを実感するとともに，改めて C 言語のような高級言語やそれをアセンブリ言語に変換してくれるコンパイラのありがたみを感じることができた．

また，アセンブリ言語でプログラムを書いていると，j 命令や jal 命令などで，コードを行ったり来たりすることが多々あり，何も考えずにプログラムを書いていると，処理の流れを追うのが困難になってしまうので，適切なラベル名の命名やどこに何の処理を書くべきかを考えながらプログラムを実装することが大事だと感じた．具体例としては，特定のラベル内からし呼び出されないラベルの接頭辞として L を追加したり，if-else の分岐を明確にするために，処理の流れは変わらないが j 命令を追加する，といった工夫を行った．