# 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 11 月 18 日
締切日: 2019 年 11 月 25 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リ言語について，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，システムコールを用いた入出力機能の実装や，アセンブリ指令のもつ役割についての解説，そして，C 言語で記述されているプログラムの再現（スタックを用いた関数の再帰呼び出しの実現，素数を表示するプログラム及びその結果の配列への保存），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
17          li      $t1, 1
18          and     $t0, $t0, $t1
19          beqz    $t0, putc
20          sw      $a0, 0xffff000c
21          addi    $a1, $a1, 1
22          bnez    $a0, putc
23          j       $ra
```

### 2.3 実行結果

```
Hello World
```

### 2.4 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，メモリ上に展開されている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．

### 2.5 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
```

### 2.6 実行結果

```
Hello World
```

### 2.7 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．

### 2.8 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（疎結合になっている）ので，入出力機器が変更されてもプログラムを再利用しやすい，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小さくすることができる．

## 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上に展開される訳ではない．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，xspim では「Can't put data in text segment」というエラーメッセージが表示され，プログラムを実行すると，文字化けした異常な出力結果が得られた．この現象が起きた原因としては，.asciiz が正しく実行されず，msg の中に 11 行目の syscall の命令が展開されているアドレスが格納されてしまっているので，正しい出力が得られない，ということが挙げられる．

## 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         la      $a0, msg
13         jal     print_string
14         li      $a0, 10
15         jal     fact
16         move    $a0, $v0
17         jal     print_int
18         la      $a0, newline
19         jal     print_string
20         j       $s0
21
22 fact:
23         subu  $sp, $sp, 32
24         sw    $fp, 16($sp)
25         sw    $ra, 20($sp)
26
27         addu  $fp, $sp, 28
28         sw    $a0, 0($fp)
29
30         bgtz  $a0, Lthen
31         j     Lelse
```

```
32
33 Lthen:
34          subu  $a0, $a0, 1
35          jal   fact
36
37          lw    $v1, 0($fp)
38          mul   $v0, $v0, $v1
39          j     Lreturn
40
41 Lelse:
42          li    $v0, 1
43          j     Lreturn
44
45 Lreturn:
46          lw    $fp, 16($sp)
47          lw    $ra, 20($sp)
48          addiu $sp, $sp, 32
49          j     $ra
50
51 print_string:
52          li      $v0, 4
53          syscall
54          j       $ra
55
56 print_int:
57          li      $v0, 1
58          syscall
59          j       $ra
60
```

### 4.3 実行結果

```
The factorial of 10 is 3628800
```

### 4.4 プログラムの作成過程に関する考察

このプログラムでは再帰呼び出しを利用して計算結果を求めているので，保持しておかなければならない値（$\$ra$ の値や $\$a0$ の値など）の数が膨大になりレジスタに収まりきらない，という問題があった．この問題を解決するために上記のプログラムでは，レジスタの値をメモリ内のスタックセグメントに一時的に退避させる，という手段をとった．

### 4.5 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.（23-28 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.（30-31 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（34-35 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. $v0 に 1 を格納した後，Lreturn ラベルの場所にジャンプする.（42-43 行目）
8. 直近にスタックに退避させた値（32 バイト分）をレジスタに復帰させた後，$sp の値を加算し，使用済みスタック領域の解放を行う.（46-48 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.（49 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.（37-38 行目）
11. Lreturn ラベルの場所にジャンプする.（39 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:
13     li   $a3, 2
14
15 for:
16     slt  $v0, $a3, $a1
17     beq  $v0, $zero, return1
```

```
18
19      div  $a1, $a3
20      mfhi $v0
21      beq  $v0, $zero, return0
22
23      addi $a3, $a3, 1
24      j    for
25
26  return0:
27      li   $v0, 0
28      j    $ra
29
30  return1:
31      li   $v0, 1
32      j    $ra
33
34  main:
35      move $s0, $ra
36      li   $t0, 1
37      li   $t1, 10
38      la   $t2, 100
39      li   $a1, 2
40      li   $a2, 0
41      j    while
42
43  while:
44      slt  $v0, $a2, $t2
45      beq  $v0, $zero, exit
46
47      jal  test_prime
48      beq  $v0,$t0,then
49      j    default
50
51  then:
52      move $a0, $a1
53      jal  print_int
54      la   $a0, space
55      jal  print_string
56      addi $a2, $a2, 1
57      div  $a2, $t1
58      mfhi $v0
59      beq  $v0, $zero, wrap
60      j    default
61
```

```
62 wrap:
63     la   $a0, newline
64     jal  print_string
65     j    default
66
67 default:
68     addi $a1, $a1, 1
69     j    while
70
71 exit:
72     la   $a0, newline
73     jal  print_string
74     j    $s0
75
76 print_string:
77     li   $v0, 4
78     syscall
79     j    $ra
80
81 print_int:
82     li   $v0, 1
83     syscall
84     j    $ra
```

### 5.4 実行結果

```
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
```

### 5.5 プログラムの作成過程に関する考察

このプログラムでは合計で 100 個もの数字を出力するので実行結果が見にくい，という問題があった．この問題を解決するために，上記のプログラムでは $a2 の値を 10 で割った余りが 0 であれば改行をする，という処理を追加した．

### 5.6 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．(36-40 行目)
2. while ラベルへジャンプする．(41 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 $t0 を比較している)(44-45 行目)
4. $a2 < $t0 なので，ループの中に入り test_primes ラベルにジャンプする．(47 行目)
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．(13 行目)
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_primes ラベルの呼び出し元へ帰る．(19-21 行目)
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_primes の呼び出し元へ帰る．($a1 は素数)(17 行目)
8. $v0 の値が 1 ならば（$a1 が素数ならば），then 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．(52-56 行目)
9. $a2 の値が $s2 の値（10）で割り切れるのであれば，改行する．(57-64 行目)
10. $a1 の値を加算し，3 まで戻る．(68-69 行目)
11. 3 から 10 までを繰り返し，$a2 の値が $t0 以上になった時，プログラムを終了する．(72-73 行目)

## 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: }
```

### 6.3 作成したプログラム

```
 1     .data
 2     .align  2
 3 newline:
 4         .asciiz "\n"
 5
 6 msg_input:
 7         .asciiz "> "
 8
 9 msg_error:
10         .asciiz "please input 0 to 100"
11
12 msg_exit:
13         .asciiz "exit"
14
15 primes:
16         .space 400
17
18     .text
19     .align  2
20
21 test_prime:
22
23     li  $a3, 2
24
25 Lfor:
26     slt $v0,$a3,$a1
27     beq $v0,$zero,return1
28
29     div $a1,$a3
30     mfhi    $v0
31     beq $v0,$zero,return0
32
33     addi    $a3, $a3, 1
```

```
34      j   Lfor
35
36  return0:
37      li  $v0,0
38      j   $ra
39
40  return1:
41      li  $v0,1
42      j   $ra
43
44  main:
45      move    $s0, $ra
46      la      $t0, primes
47      li      $t1, 1
48      la      $t2, 100
49      li      $a1, 2
50      li      $a2, 0
51      j       while
52
53  while:
54      slt     $v0, $a2, $t2
55      beq     $v0, $zero, find
56
57      jal     test_prime
58      beq     $v0, $t1, then
59      j       default
60
61  then:
62      move    $a3, $a2
63      jal     get_address
64      sw      $a1, 0($v0)
65      addi    $a2, $a2, 1
66      j       default
67
68  default:
69      addi    $a1, $a1, 1
70      j       while
71
72  find:
73      la      $a0, msg_input
74      jal     print_string
75      jal     read_int
76      bltz    $v0, error
77      bgt     $v0, $t2, error
```

```
 78     beq     $v0, $zero, exit
 79     move    $a3, $v0
 80     addi    $a3, $a3, -1
 81     jal     get_address
 82     lw      $a0, 0($v0)
 83     jal     print_int
 84     la      $a0, newline
 85     jal     print_string
 86     j       find
 87
 88 error:
 89     la      $a0, msg_error
 90     jal     print_string
 91     la      $a0, newline
 92     jal     print_string
 93     j       find
 94
 95 exit:
 96     la      $a0, msg_exit
 97     jal     print_string
 98     j       $s0
 99
100
101 get_address:
102     addu    $a3, $a3, $a3
103     addu    $a3, $a3, $a3
104     addu    $a3, $t0, $a3
105     move    $v0, $a3
106     j       $ra
107
108 print_string:
109     li      $v0, 4
110     syscall
111     j       $ra
112
113 print_int:
114     li      $v0, 1
115     syscall
116     j       $ra
117
118 read_int:
119     li      $v0, 5
120     syscall
121     j       $ra
```

### 6.4 実行結果

```
> 1
2
> 5
11
> 10
29
> 42
181
> 81
419
> 105
please input 1 to 100
> -100
please input 1 to 100
> 0
exit
```

### 6.5 プログラムの作成過程に関する考察

このプログラムでは，ユーザが想定外の入力を行うことを考えられておらず，プログラム内で確保したメモリ領域以外にもアクセスできてしまう，という問題があった．この問題を解決するために，上記のプログラムでは異常な値が入力された時は処理を分岐しエラーメッセージを表示する，という機能を追加した．

### 6.6 プログラムの解説

上記のプログラムについて，課題 1-4 との変更点の部分を中心に解説する．

大きな違いとして挙げられるのは，素数を見つけた際にその値を出力するのではなく，後のユーザーとの対話処理で使うためにその値を保持している，という点である．以下に，プログラム内の配列に関する操作について詳しく解説する．

1. アセンブラ指令.space であらかじめメモリ上に適当な領域を確保し，その先頭アドレスを primes とする．今回の場合は int 型（4 バイト）を 100 個格納できれば良いので，400 バイトを確保しておく.（15-16 行目）

2. 素数を見つけたタイミングで $\$t0 + \$a2 \times 4$ のアドレスにその素数を格納する.（4 をかけるのは，int 型のバイト数が 4 だから）（63-64 行目）

3. ユーザーから任意の数字（1-100）が入力された際は $\$t0 + (\$v0 - 1) \times 4$ のアドレスにアクセスすることで，任意の配列の要素にアクセスし，そこに格納されている値を取得することができる.（74-75 行目）

## 7　感想

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，処理の分岐を明確にするために，それぞれの分岐先の処理の最後に同じラベルへの `j` 命令を追加して枝分かれしていることを表す，といった工夫を行うと，後から見た時に処理の流れを追いやすくなる，と感じた．