

# システムプログラミング 1

## レポート

氏名: 今田 将也 (IMADA, Masaya)

学生番号: 09430509

出題日: 2019 年 xx 月 xx 日

提出日: 20xx 年 xx 月 xx 日

締切日: 20xx 年 xx 月 xx 日

## 1 概要

本演習では、プログラミングに関する理解を深めるために不可欠なアセンブラと C 言語の境界部分についての演習や MIPS アーキテクチャとアセンブリ言語、アセンブラ特有の記法、また、メモリや入出力、文字と文字列の扱い、レジスタやスタックを用いた手続き呼出の仕組みの演習を行った。具体的には、SPIM という MIPS CPU シミュレータのハードウェア上に C 言語とアセンブリ言語を仕様して文字の表示と入力のためのシステムコールライブラリを作成した。さらに、そのライブラリを使用して `printf` 及び `gets` 相当を今後 C 言語で作成する。本課題で実行した結果は、`xspim` というエミュレータのコンソール上の結果を表示している。

なお、与えられた課題内容を以下に述べる。

### 1.1 課題内容

以下の課題についてレポートをする。プログラムは、MIPS アセンブリ言語で記述し、SPIM を用いて動作を確認している。

**課題 1-1** 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。

**課題 1-2** 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` がいない場合、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

```

課題 1-3 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: }

```

課題 1-4 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。

関数名	概要
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

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: }

```

実行結果（行を適当に折り返している）:

```

 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

```

**課題 1-5** 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．

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: }

```

実行例：

```

> 15
47
> 100
541

```

## 1.2 xspim の実行方法

`xspim -mapped_io&`

でコンソール上で実行後，必要なアセンブリファイルを load し，run することで実行した．

## 2 課題レポート

### 2.1 課題 1-1

#### 2.1.1 作成したプログラム

ソースコード 1: A.8 節「入力と出力」に示されている方法

---

```

1      .text
2      .align 2
3  main:
4      li $a0,72
5  putc:
6      lw $t0,0xffff0008
7      li $t1,1
8      and $t0,$t0,$t1
9      beqz $t0,putc
10     sw $a0,0xffff000c
11     li $a0,101
12  putc2:
13     lw $t0,0xffff0008
14     li $t1,1
15     and $t0,$t0,$t1
16     beqz $t0,putc2
17     sw $a0,0xffff000c
18     li $a0,108
19  putc3:
20     lw $t0,0xffff0008
21     li $t1,1
22     and $t0,$t0,$t1
23     beqz $t0,putc3
24     sw $a0,0xffff000c
25     li $a0,108
26  putc4:
27     lw $t0,0xffff0008

```

```

28     li $t1,1
29     and $t0,$t0,$t1
30     beqz $t0,putc4
31     sw $a0,0xffff000c
32     li $a0,111
33 putc5:
34     lw $t0,0xffff0008
35     li $t1,1
36     and $t0,$t0,$t1
37     beqz $t0,putc5
38     sw $a0,0xffff000c
39     li $a0,32
40
41 putc6:
42     lw $t0,0xffff0008
43     li $t1,1
44     and $t0,$t0,$t1
45     beqz $t0,putc6
46     sw $a0,0xffff000c
47     li $a0,87
48 putc7:
49     lw $t0,0xffff0008
50     li $t1,1
51     and $t0,$t0,$t1
52     beqz $t0,putc7
53     sw $a0,0xffff000c
54     li $a0,111
55 putc8:
56     lw $t0,0xffff0008
57     li $t1,1
58     and $t0,$t0,$t1
59     beqz $t0,putc8
60     sw $a0,0xffff000c
61     li $a0,114
62 putc9:
63     lw $t0,0xffff0008
64     li $t1,1
65     and $t0,$t0,$t1
66     beqz $t0,putc9
67     sw $a0,0xffff000c
68     li $a0,108
69 putc10:
70     lw $t0,0xffff0008
71     li $t1,1
72     and $t0,$t0,$t1
73     beqz $t0,putc10
74     sw $a0,0xffff000c
75     li $a0,100
76 putc11:
77     lw $t0,0xffff0008
78     li $t1,1
79     and $t0,$t0,$t1
80     beqz $t0,putc11
81     sw $a0,0xffff000c
82     j $ra

```

---

## ソースコード 2: A.9 節「システムコール」に示されている方法

---

```

1     .data
2     .align 2
3 str: .asciiz "Hello World"
4
5     .text
6     .align 2
7
8 main: li $v0,4 #print_str のシスコールを$v0 にロード

```

```

9      la $a0,str #プリントする文字列のアドレスをsyscall の引数
10          # $a0 にロードアドレス命令を行う
11      syscall
12      j $ra # $ra レジスタへ戻り、プログラム終了

```

---

### 2.1.2 考察

前者での文字の出力は、野蛮な方法である。計算機ごとに変り得るアドレスの 0xffff000c を意識しつつ使うのは面倒であり、アドレスを知る術がない場合実装するのが不可能である。また、仮に他のプログラムも同時に印刷しようとした場合に競合が発生する可能性もある。このプログラムは印刷が可能になるまで待機してから印刷を行っているが、待たずに印刷するようなプログラムを作成した場合、機器の破壊につながることもあるだろう。

それに比べて、システムコールはカーネルごとに引数の意味が異なって、そのアドレスが変化したとしてもプログラムを変更する必要がなく、他のプログラムとの競合も調整してもらえるため、安全にプログラムを走行することができる。システムコール命令を用いることで安全にユーザプログラムからカーネルやメモリ資源を保護することができ、カーネルに所望の処理を依頼することができる。

## 2.2 課題 1-2

### 2.2.1 実行結果

課題のコードの実行結果

Hello!!

6 行目の .data をコメントアウトした場合の実行結果

X\200}B

### 2.2.2 考察

まず、.data、.text とはメモリ中のどこにデータやテキストを配置するかを制御するためのアセンブラ指令である。本講義で使用した SPIM はテキストとデータのセグメントを分割してメモリ中に並べていくようになっている。しかし、テキストとデータは最終的にどちらも数値であるため、どちらをどこに配置するかアセンブラでは決定できず、プログラマ側で指定する必要がある。また、テキストは通常書き換わることはないのでデータと違って読み込み専用のメモリ上に配置することができる。また、異なるプロセスで同じプログラムを実行する場合でもテキストは同一なので共有することも可能になる。このように、データとテキストを意識して区別することで効率的にプロセスを実行できる。

課題中の 6 行目の .data が無い場合、xspim で load を実行した時点で、.asciiz の "Hello!!\n" がデータセグメントであるため、テキストセグメントに配置することができないというエラー表示が

出る。そして、実行すると X\200}B と表示された。もう一度実行すると、@\207Y^B と異なった表示がされた。 .data がなくなったことで、\$a0 レジスタに msg の示すアドレスの先に "Hello!!\n" ではないデータが存在するようになり、印刷する際にそのアドレスの内容を表示していると考えられる。その内容は実行ごとに変わるため、表示結果も変わっていると考ええる。

## 2.3 課題 1-3

### 2.3.1 実装内容

プログラムは大きく分けて、main と fact 部に分かれる。main 部では、まず手続き呼出規約に従ってスタックを確保した。そして課題のフローに従って、引数を設定し fact を実行後、結果をシステムコールにて印刷し各種のアドレスを復元し、スタックポインタをポップし、プログラムを終了する。fact 部では、まず main 部同様に手続き呼出規約に従ってスタックを確保し、再帰的に引数を渡すことができるように確保した。引数が 0 より大きいなら再帰処理の factsub に jal 命令を実行、0 以下なら 1 を返し、一つ前の fact ルーチンと呼出し、その計算結果に引数をかけていくという処理を再帰的に繰り返した。fact ルーチン終了後は戻りアドレスを復元し、スタックポインタをポップする処理を行いルーチンを終了させている。

### 2.3.2 作成したプログラム

ソースコード 3: 10 の階乗を再帰的に求めるプログラム

```
1      .data
2      .align 2
3  str:
4      .ascii "The factorial of 10 is "
5      .text
6      .align 2
7  print_int:
8      li $v0,1
9      syscall
10     j $ra
11  print_str:
12     li $v0,4
13     syscall
14     j $ra
15  main:
16     subu $sp,$sp,32 #スタックフレームは3 2バイト長で確保をする
17     sw $ra,20($sp) #戻りアドレスを退避させる
18     sw $fp,16($sp) #古いフレームポインタを退避
19     addiu $fp,$sp,28 #新しくフレームポインタを設定
20     #fact を呼び出して戻ってから、syscall で$LC と fact の戻り値をプリントする
21     li $a0,10 #引数は 10
22     jal fact
23     move $t1,$v0 #戻り値をt1 に退避
24     la $a0,str #a0 にテンプレ文のアドレスを記入
25     jal print_str
26     move $a0,$t1 #fact の戻り値を保存した t1 を$a0 に収める
27     jal print_int
28     #退避してあったレジスタを復元したあと呼出側へ戻る
29     lw $ra,20($sp) #戻りアドレスを復元
30     lw $fp,16($sp) #フレームポインタを復元
31     addiu $sp,$sp,32 #スタックポインタをポップする
32     j $ra
33  fact:
34     subu $sp,$sp,32 #スタックフレームは3 2バイト長
35     sw $ra,20($sp) #戻りアドレスを退避させる
```

```

36      sw $fp,16($sp) #古いフレームポインタを退避
37      addiu $fp,$sp,28 #新しくフレームポインタを設定
38      sw $a0,0($fp) #引数を退避させる# 28($sp)にもってきているもの
39      #引数>0かどうかを調べる。
40      #引数<=0なら1を返す。
41      #引数>0ならfactルーチンを呼出(n-1)を計算し、その結果にnをかける
42      #上記を再帰的に繰り返す
43      lw $v0,0($fp) #nをロードしておく
44      bgtz $v0,factsub #引数が0より大きければ再帰処理に飛ぶ
45      li $v0,1 #0以下なら1
46      j return
47 factsub:
48      lw $v1,0($fp) #nをロードする
49      subu $v0,$v1,1 #n-1
50      move $a0,$v0 #a0にn-1に戻る
51      jal fact
52
53      lw $v1,0($fp)
54      mul $v0,$v0,$v1 #n*fact(n-1)
55 return: #return 処理
56      lw $ra,20($sp) #戻りアドレスを復元
57      lw $fp,16($sp) #フレームポインタを復元
58      addiu $sp,$sp,32 #スタックポインタをポップする
59      j $ra

```

---

### 2.3.3 実行テスト結果

```

$ xspim -mapped_io&
The factorial of 10 is 3628800

```

## 2.4 課題 1-4

### 2.4.1 実装内容

#### 2.4.2 作成したプログラム

ソースコード 4: 素数を 100 個表示するプログラム

---

```

1      .data
2      .align 2
3 space:
4      .asciiz " "
5 enter:
6      .asciiz "\n"
7
8      .text
9      .align 2
10 test_prime:
11      subu $sp,$sp,32 #スタックポインター
12      sw $ra,20($sp) # $ra
13      sw $fp,16($sp) #フレームポインター
14      addiu $fp,$sp,28 #フレームポインターのセット
15      li $t0,2 # 1は素数ではないから 2 を初期値にセット
16 prime_for:
17      beq $t0, $a0, return1 # return 1 もし n が素数なら (i==n)
18      bgt $t0, $a0, prime_exit # for ループを抜ける. もし n > i なら.
19      rem $t1, $a0, $t0 # $t1 = n % i nをiで割ったあまり
20      beqz $t1, prime_exit # goto Exit_prime if $t1 == 0
21      addi $t0, $t0, 1 # i++
22      j prime_for # 再びループへ

```



```

23 return1:
24     li $v0,1 #もしnが素数なら1を代入して返す
25     lw $ra,20($sp)
26     lw $fp,16($sp)
27     addiu $sp,$sp,32
28     j $ra #mainへ戻る
29 prime_exit:
30     li $v0,0 #もしnが素数でないなら0を代入して返す
31     lw $ra,20($sp)
32     lw $fp,16($sp)
33     addiu $sp,$sp,32
34     j $ra #ループを抜ける
35 main:
36     subu $sp,$sp,32 #stackpointer
37     sw $ra,20($sp) #$ra
38     sw $fp,16($sp) #framepointer
39     addiu $fp,$sp,28 #set fp
40     li $s0,100 #最大ループ回数 (match<100)
41     li $s1,0 #現在のループ回数 (match)
42     li $s2,2 #チェック用の数値 n (n=2)
43     li $s3,1 #test_prime(n) == $s3
44     li $s4,10 #10個表示されたら改行するため.print \n
45 while:
46     beq $s0,$s1,exit # s1 == 100 ならば(0~99まで),exitに行く
47     move $a0,$s2 # $s2 => $a0 に移動
48     jal test_prime
49     bne $v0,$s3,else # $v0 != 1 test_prime から帰ってきた数値で検証する
50     move $a0,$s2 # 印刷のために,数字を入れる
51     li $v0,1 # 1 はint
52     syscall
53     la $a0, space # 空白を印刷
54     li $v0,4 # 4 は文字列
55     syscall
56     addiu $s1,$s1,1 #現在のループ回数を増加
57     rem $t2, $s1, $s4 # $t2 = n % 10 個表示されたかどうか
58     beqz $t2, print_enter # 改行表示 もし $t2 (個数) == 0
59 else:
60     addiu $s2, $s2, 1 # n = n + 1
61     j while # while ループを繰り返す
62 exit:
63     lw $ra, 20($sp) # Restore return address
64     lw $fp, 16($sp) # Restore frame pointer
65     addiu $sp, $sp, 32 # Pop stack frame
66     j $ra # End this program
67 print_enter:
68     subu $sp,$sp,32 #stackpointer
69     sw $ra,20($sp) #$ra
70     sw $fp,16($sp) #framepointer
71     addiu $fp,$sp,28 #set fp
72     la $a0,enter
73     li $v0,4
74     syscall
75     lw $ra, 20($sp) # Restore return address
76     lw $fp, 16($sp) # Restore frame pointer
77     addiu $sp, $sp, 32 # Pop stack frame
78     j $ra #return

```

---

### 2.4.3 実行テスト結果

```

$ xspim -mapped_io&
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

## 2.5 課題 1-5

### 2.5.1 実装内容

### 2.5.2 作成したプログラム

ソースコード 5: 配列で入力に応じて素数を表示するプログラム

---

```

1 array:
2     .space 400 #400バイト分(100個分)の配列用意
3
4     .data
5     .align 2
6 space:
7     .asciiz " "
8 enter:
9     .asciiz "\n"
10 start:
11     .asciiz "To quit, type 0\n\n"
12
13 mark:
14     .asciiz "\n> "
15 owari:
16     .asciiz "\nGood bye :)\n\n"
17 excep:
18     .asciiz "\nPlease type correct number\n"
19
20
21     .text
22     .align 2
23
24 test_prime:
25     subu $sp,$sp,32 #スタックポインター
26     sw $ra,20($sp) #$ra
27     sw $fp,16($sp) #フレームポインター
28     addiu $fp,$sp,28 #フレームポインターのセット
29     li $t0,2 # 1は素数ではないから2を初期値にセット
30 prime_for:
31     beq $t0, $a0, return1 # return 1 もし n が素数なら (i==n)
32     bgt $t0, $a0, prime_exit # for ループを抜ける. もし n > i なら.
33     rem $t1, $a0, $t0 # $t1 = n % i   nをiで割ったあまり
34     beqz $t1, prime_exit # goto Exit_prime if $t1 == 0
35     addi $t0, $t0, 1 # i++
36     j prime_for # 再びループへ
37 return1:
38     li $v0,1 #もしnが素数なら1を代入して返す
39     lw $ra,20($sp)
40     lw $fp,16($sp)
41     addiu $sp,$sp,32
42     j $ra #main へ戻る

```

```

43 prime_exit:
44     li $v0,0 #もしnが素数でないなら0を代入して返す
45     lw $ra,20($sp)
46     lw $fp,16($sp)
47     addiu $sp,$sp,32
48     j $ra #ループを抜ける
49 main:
50     subu $sp,$sp,32 # stackpointer
51     sw $ra,20($sp) # $ra
52     sw $fp,16($sp) # flamepointer
53     addiu $fp,$sp,28 # set fp
54     li $v0, 4 # syscall of print_string
55     la $a0, start # start ラベルの内容を入れる
56     syscall # 印刷内容 "To quit, type 0"
57     li $s0,100 #最大ループ回数 (match<100)
58     li $s1,0 #現在のループ回数 (match)
59     li $s2,2 #チェック用の数値 n (n=2)
60     li $s3,1 #test_prime(n) == $s3
61     la $a1,array # $a1 に array のアドレスを入れる
62 while:
63     beq $s0,$s1,exit # s1 == 100 ならば(0～99まで),exitに行く
64     move $a0,$s2 # $s2 => $a0 に移動
65     jal test_prime
66     bne $v0,$s3,else # $v0 != 1 test_prime から帰ってきた数値で検証する
67     li $t4, 4 # For array を増加(4バイト単位)
68     addu $a1, $a1, $t4 # $a1 = $a1 + 4
69     sw $s2, 0($a1) # $s2 (素数)=>$a1 が指すアドレスの中の先頭にいれる
70     addiu $s1,$s1,1 # 現在のループ回数を増加
71 else:
72     addiu $s2, $s2, 1 # n = n + 1
73     j while # go to Loop
74 exit:
75     li $v0, 4 # syscall of print_string
76     la $a0, mark # 印字内容 ">"
77     syscall # 印刷
78     la $a1, array # Initialize $a1
79     li $v0, 5 # For syscall of read_int
80     syscall # 何番目の素数かを入力する
81     beqz $v0,end # 0か文字なら終了
82     bltz $v0,error # 負ならエラー
83     bgtu $v0,$s0,error # 100より大きくてもエラー
84     move $t3, $v0 # 入力された値$v0を$t3に
85     addu $t3, $t3, $t3 # $t3 = $t3 * 2
86     addu $t3, $t3, $t3 # $t3 = $t3 * 2 4バイト分になる
87     addu $a1, $a1, $t3 # $a1 = $a1
88     lw $a0, 0($a1) # $a1 のアドレスから4バイト(word)取り出して $a0 に代入 (load)
89     li $v0, 1 # For syscall of print_int
90     syscall # Print prime
91     j exit
92 error:
93     li $v0, 4 # for syscall of print_string
94     la $a0, excep # Print error message
95     syscall # print
96     j exit
97 end:
98     li $v0, 4 # for syscall of print_string
99     la $a0, owari # good,bye
100    syscall # print
101    lw $ra, 20($sp) # Restore return address
102    lw $fp, 16($sp) # Restore frame pointer
103    addiu $sp, $sp, 32 # Pop stack frame
104    j $ra # End this program

```

---

### 2.5.3 実行テスト結果

```
$ xspim -mapped_io&  
To quit, type 0  
> 100  
541  
> 14  
43  
> -1000  
Please type correct number  
> 199  
Please type correct number  
> 0  
Good bye :)
```

## 3 感想