# システムプログラミング 1
# 期末レポート

氏名: 重近 大智 (SHIGECHIKA, Daichi)
学生番号: 09501527

出題日: 2020 年 10 月 05 日
提出日: 2020 年 11 月 xx 日
締切日: 2020 年 11 月 16 日

## 1 概要

本レポートでは，2 章に示す 5 つの課題に取り組み，その解答を報告する．使用した MIPS アセンブリ言語のソースコードは，5 章に示す．実行結果は xspim による結果である．

## 2 課題

システムプログラミング 1 の課題は次の 5 つである．解答は 3 章に示す．

1. 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．[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． 5.2 節のコード中の 9 行目の `.data` がない場合，どうなるかについて考察せよ．
3. A.6 節 「手続き呼出し規約」に従って，再帰関数 `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)[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に表 1 に示す 2 つのルーチンを作成すること．
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．

表 1 課題 4 で実装する 2 つのルーチン

| 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 |

# 3　解答と実行結果

## 3.1　課題 1-1

A.8 節 「入力と出力」に示されている方法に示されている方法で実装する場合，空行を除くコードは 25 行である．それに対して，A.9 節 最後「システムコール」に示されている方法で実装する場合，空行を除くコードはたった 12 行となった．これは前者が 1 文字ずつしか表示できないため，1 文字ずつ読み込む処理とアドレスを 1 加える処理（12〜16 行目）がループになっていることと，プリンタを利用する際にプリンタが使用可能な状態になっていることを確認するために 0xffff0008 番地から値を読み，1 になっているか確認する処理 (18〜21 行目) があることに起因する．また jal 命令を用いているため，main を呼び出したときの$ra レジスタの値が破壊されるため，そのコピーをするために move 命令を使っていることも行数が増える原因の一端となっている．後者は，文字列を表示できるため，アドレスに 1 を加えてずらす処理が必要なく，表示するにあたって特定番地の値の確認を取る必要もない．必要なのは，syscall 命令の動作を規定する引数を$v0 レジスタに代入するだけである．そのため，非常に簡潔なコードとなっている．

特に，「入力と出力」に示されている方法で実装する場合は，

## 3.2　課題 1-2

## 3.3　課題 1-3

## 3.4　課題 1-4

## 3.5　課題 1-5

# 4　感想

# 5　作成したプログラムのソースコード

使用したプログラムを以下に添付する．

## 5.1　課題 1-1 で用いたコード

下記は，「入力と出力」に示されている方法で実装した例．

```
 1:      .data                   # データセグメント
 2:      .align 2                # 2 の n 乗の境界上になるまで隙間を空けてくれる
 3: msg:
 4:      .asciiz "Hello World"   # 出力文字情報
 5:
 6:      .text                   # テキストセグメント
 7:      .align 2                # バイトを揃える
 8: main:
 9:      la      $a0, msg        # a0 に msg のアドレスを格納する
10:      move    $s0, $ra        # $s0 に main を呼び出した元のアドレスを格納
11: loop:
12:      lb      $a1, 0($a0)     # a0 の指し先の値を$a1 にロードする
13:      beqz    $a1, end        # a1 が 0 のとき，end へ
14:      jal     print           # print のアドレスにジャンプ（次の命令のアドレスを$ra に）
15:      addi    $a0, $a0, 1     # $a0 に 1 を加える
16:      j       loop            # main のアドレスにジャンプ
17: print:
18:      lw      $t0, 0xffff0008 # 0xffff0008 番地にあるワードを$t0 にロードする
19:      li      $t1, 1          # $t1 に 1 を代入する
20:      and     $t0, $t0, $t1   # $t0 と$t1 の論理積をとって$t0 に代入する（$t0 が 1 か確認する）
21:      beqz    $t0, print      # $t0 が 0 のとき，もう一度準備を試みる
```

```
22:     sw      $a1, 0xffff000c # 0xffff000c 番地に$a1 にあるワードを送る
23:     j       $ra             # 呼び出し元に戻る
24: end:
25:     move    $ra, $s0        # main を呼び出した元のアドレスを$ra に復元
26:     j       $ra             # コンソールに戻る
```

下記は,「システムコール」に示されている方法で実装した例.

```
 1:     .data                   # データセグメント
 2:     .align 2                # バイトを揃える
 3: msg:
 4:     .asciiz "Hello World"   # 出力文字情報
 5:
 6:     .text                   # テキストセグメント
 7:     .align 2                # バイトを揃える
 8: main:
 9:     la      $a0, msg        # msg のアドレスを $a0 に格納
10:     li      $v0, 4          # print_string を指定
11:     syscall                 # システムコールの実行
12:     j       $ra             # コンソールに戻る
```

## 5.2 課題 1-2 で用いたコード

```
 1:     .text
 2:     .align 2
 3:
 4: _print_data:
 5:     la      $a0, ghost
 6:     lw      $a0, 0($a0)
 7:     li      $v0, 1
 8:
 9:     .data
10:     .align 2
11: ghost:
12:     .word 12
13:
14:     .text
15:     syscall
16:     j       $ra
17:
18: main:
19:     subu    $sp, $sp, 24
20:     sw      $ra, 16($sp)
21:     jal     _print_data
22:     lw      $ra, 16($sp)
23:     addu    $sp, $sp, 24
24:     j       $ra
```

## 5.3 課題 1-3 で用いたコード

```
 1:     .text
 2:     .align 2
 3: main:
 4:     move    $s0, $ra        # main を呼んだ戻り先のアドレスを$s0 に保存しておく
 5:
 6:     la      $a0, msg        # msg のアドレスを$a0 にロード
 7:     jal     print_str       # print_str のアドレスにジャンプ（次の命令のアドレスを$ra に）
 8:
 9:     li      $a0, 10         # $a0 に 10 を代入（任意の int 値）
10:
11:     jal     fact            # fact に飛ぶ
12:     move    $a0, $v0        # $a0 に$v0 の値をコピー
13:     jal     print_int       # print_int のアドレスにジャンプ（次の命令のアドレスを$ra に）
14:
15:     la      $a0, msg2       # msg2 のアドレスを$a0 にロード
```

```
16:     jal     print_str       # print_str のアドレスにジャンプ（次の命令のアドレスを$ra に）
17:
18:     move    $ra, $s0        # main を呼んだ戻り先のアドレスを$ra に代入
19:     j       $ra             # コンソールに戻る
20:
21: fact:
22:     subu    $sp, $sp, 32    # 32 バイト確保
23:     sw      $ra, 20($sp)    # $sp + 20 のアドレスにもともとの呼出しアドレスを保存
24:     sw      $fp, 16($sp)    # $sp + 16 のアドレスに$fp を保存
25:     sw      $a0, 24($sp)    # $sp + 24 のアドレスに$a0 を保存
26:     addu    $fp, $sp, 28    # 新しく$fp を設定
27:
28:     li      $t0,   1        # $t0 に 1 を代入（分岐命令用）
29:
30:     bgeu    $a0, $t0, else  # $a0 が$t0 より小さいとき，else へ
31:     li      $v0, 1          # $v0 に 1 を代入
32:     lw      $ra, 20($sp)    # 保存しておいた$sp を復元
33:     lw      $fp, 16($sp)    # 保存しておいた$fp を復元
34:     addu    $sp, $sp, 32    # スタックを開放
35:     j       $ra             # 呼び出し元に戻る
36: else:
37:     addi    $a0, -1         # $a0 から 1 を引く
38:     jal     fact            # 再帰呼び出しをする
39:
40:     lw      $ra, 20($sp)    # 保存しておいた$sp を復元
41:     lw      $fp, 16($sp)    # 保存しておいた$fp を復元
42:     lw      $a0, 24($sp)    # 対応する$a0 の値を復元
43:     addu    $sp, $sp, 32    # スタックを開放
44:
45:     mulo    $v0, $a0, $v0   # $v0 と$a0 をかけて$v0 に代入
46:     j       $ra             # 呼び出し元に戻る
47:
48: print_int:
49:     li      $v0, 1          # $v0 に 1 を代入
50:     syscall                 # システムコールの実行
51:     j       $ra             # 呼び出し元に戻る
52: print_str:
53:     li      $v0, 4          # $v0 に 4 を代入
54:     syscall                 # システムコールの実行
55:     j       $ra             # 呼び出し元に戻る
56:
57:     .data
58:     .align 2
59: msg:
60:     .asciiz "The factorial of 10 is "
61:
62:     .align 2
63: msg2:
64:     .asciiz "\n"
```

## 5.4 課題 1-4 で用いたコード

```
 1:     .text
 2:     .align 2
 3: main:
 4:     move    $s2, $ra        # $s2 に main を呼び出した元のアドレスを格納
 5:     li      $s0, 2          # $s0 に 2 を格納（検索する数）
 6:     li      $s1, 0          # $s1 に 0 を格納（素数の個数カウント）
 7: loop:
 8:     move    $a0, $s0        # $s0 の値を$a0 にコピー
 9:     jal     test_prime      # test_prime のアドレスにジャンプ（次の命令のアドレスを$ra に）
10:     beqz    $v0, r1         # $v0 が 0 ならば，r1 に分岐
11:     addi    $s1, 1          # $s1 に 1 を加える
12:     move    $a0, $s0        # $v0 の値を$a0 にコピー
13:     jal     print_int       # print_int のアドレスにジャンプ（次の命令のアドレスを$ra に）
14:     la      $a0, space      # space のアドレスを$a0 にロード
15:     jal     print_str       # print_str のアドレスにジャンプ（次の命令のアドレスを$ra に）
```

```
16: r1:
17:     bge     $s1, 100, end   # $s1 が 100 以上のとき，end にジャンプ
18:     addi    $s0, 1          # $s0 に 1 を加える
19:     j       loop            # loop のアドレスにジャンプ
20: end:
21:     la      $a0, line       # line のアドレスを$a0 にロード
22:     jal     print_str       # print_str のアドレスにジャンプ（次の命令のアドレスを$ra に）
23:     move    $ra, $s2        # main を呼び出した元のアドレスを$ra に復元
24:     j       $ra             # コンソールに戻る
25:
26: test_prime:
27:     li      $t0, 2          # $t0 に 2 を代入
28:     move    $t1, $a0        # $t1 に$a0 の値をコピー
29: prime_loop:
30:     rem     $t2, $t1, $t0   # $t1 を$t0 で割った余りを$t2 に代入する
31:     beq     $t0, $t1, prime_r1 # $t0 と$t1 が等しいとき prime_r1 のアドレスにジャンプ
32:     beqz    $t2, prime_r2   # $t2 が 0 なら prime_r2 のアドレスにジャンプ
33:     addi    $t0, 1          # $t0 に 1 を加える
34:     j       prime_loop      # prime_loop のアドレスにジャンプ
35: prime_r1:
36:     li      $v0, 1          # $v0 に 1 を代入
37:     j       prime_end       # prime_end のアドレスにジャンプ
38: prime_r2:
39:     li      $v0, 0          # $v0 に 0 を代入
40: prime_end:
41:     j       $ra             # 呼び出し元に戻る
42:
43: print_int:
44:     li      $v0, 1          # $v0 に 1 を代入
45:     syscall                 # システムコールの実行
46:     j       $ra             # 呼び出し元に戻る
47: print_str:
48:     li      $v0, 4          # $v0 に 4 を代入
49:     syscall                 # システムコールの実行
50:     j       $ra             # 呼び出し元に戻る
51:
52:     .data
53:     .align 2
54: space:
55:     .asciiz " "
56:     .align 2
57: line:
58:     .asciiz "\n"
```

## 5.5 課題 1-5 で用いたコード

```
 1:     .text
 2:     .align 2
 3: main:
 4:     move    $s2, $ra        # $s2 に main を呼び出した元のアドレスを格納
 5:     li      $s0, 2          # $s0 に 2 を格納（検索する数）
 6:     li      $s1, 0          # $s1 に 0 を格納（素数の個数カウント）
 7:     la      $s3, prime_array# $s3 に prime_array のアドレスをロード
 8: loop:
 9:     move    $a0, $s0        # $s0 の値を$a0 にコピー
10:     jal     test_prime      # test_prime のアドレスにジャンプ（次の命令のアドレスを$ra に）
11:     beqz    $v0, r1         # $v0 が 0 ならば，r1 に分岐
12:     addi    $s1, 1          # $s1 に 1 を加える
13:     # ここで配列にデータを格納
14:     move    $t0, $s1        # $s1 の値を$t0 にコピー
15:     mulo    $t0, $t0, 4     # $t0 を 4 倍（4byte 区切り）
16:     add     $t0, $t0, $s3   # $t0 と$s3 を足して$t0 に格納
17:     sw      $s0, 0($t0)     # $s0 の値を配列上にコピー
18: r1:
19:     bge     $s1, 100, loop2 # $s1 が 100 以上のとき，loop2 にジャンプ
20:     addi    $s0, 1          # $s0 に 1 を加える
21:     j       loop            # loop のアドレスにジャンプ
```

```
22: loop2:
23:     la      $a0, indicate   # indicate のアドレスを$a0 にロード
24:     jal     print_str       # print_str のアドレスにジャンプ（次の命令のアドレスを$ra に）
25:     #入力受付
26:     jal     read_int        # read_int のアドレスにジャンプ（次の命令のアドレスを$ra に）
27:     # 例外確認
28:     bge     $v0, 101, end   # 入力が 101 以上の場合，終了
29:     ble     $v0, 0, end     # 入力が 0 以下の場合，終了
30:     # 正しい番地を計算
31:     move    $t0, $v0        # $v0 の値を$t0 にコピー
32:     mulo    $t0, $t0, 4     # $t0 を 4 倍（4byte 区切り）
33:     add     $t0, $t0, $s3   # $t0 と$s3 を足して$t0 に格納
34:     # 配列から読みだした値を表示
35:     lw      $a0, 0($t0)     # $s0 の値を配列上にコピー
36:     jal     print_int       # print_int のアドレスにジャンプ（次の命令のアドレスを$ra に）
37:     la      $a0, line       # line のアドレスを$a0 にロード
38:     jal     print_str       # print_str のアドレスにジャンプ（次の命令のアドレスを$ra に）
39:     j       loop2
40: end:
41:     move    $ra, $s2        # main を呼び出した元のアドレスを$ra に復元
42:     j       $ra             # コンソールに戻る
43:
44: test_prime:
45:     li      $t0, 2          # $t0 に 2 を代入
46:     move    $t1, $a0        # $t1 に$a0 の値をコピー
47: prime_loop:
48:     rem     $t2, $t1, $t0   # $t1 を$t0 で割った余りを$t2 に代入する
49:     beq     $t0, $t1, prime_r1 # $t0 と$t1 が等しいとき prime_r1 のアドレスにジャンプ
50:     beqz    $t2, prime_r2   # $t2 が 0 なら prime_r2 のアドレスにジャンプ
51:     addi    $t0, 1          # $t0 に 1 を加える
52:     j       prime_loop      # prime_loop のアドレスにジャンプ
53: prime_r1:
54:     li      $v0, 1          # $v0 に 1 を代入
55:     j       prime_end       # prime_end のアドレスにジャンプ
56: prime_r2:
57:     li      $v0, 0          # $v0 に 0 を代入
58: prime_end:
59:     j       $ra             # 呼び出し元に戻る
60:
61: print_int:
62:     li      $v0, 1          # $v0 に 1 を代入
63:     syscall                 # システムコールの実行
64:     j       $ra             # 呼び出し元に戻る
65: print_str:
66:     li      $v0, 4          # $v0 に 4 を代入
67:     syscall                 # システムコールの実行
68:     j       $ra             # 呼び出し元に戻る
69: read_int:
70:     li      $v0, 5          # read_int(戻り値は$v0 に，1~100 のみ入力可）
71:     syscall                 # システムコールの実行
72:     j       $ra             # 呼び出し元に戻る
73:
74:     .data
75:     .align 2
76: indicate:
77:     .asciiz ">"
78:     .align 2
79: line:
80:     .asciiz "\n"
81:     .align 2
82: prime_array:
83:     .space 400
```

## 参考文献

[1] David A. Patterson，John L. Hennessy，コンピュータの構成と設計　第 5 版 [下]　〜ハードウエアとソフトウエア〜，日経 BP 社，2014.