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

A.8 節「入力と出力」に示されている方法

```
        .text
        .align 2
main:
        li      $a0,72
putc:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc
        sw      $a0,0xffff000c
        li      $a0,101
putc2:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc2
        sw      $a0,0xffff000c
```

```
        li      $a0,108
putc3:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc3
        sw      $a0,0xffff000c
        li      $a0,108
putc4:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc4
        sw      $a0,0xffff000c
        li      $a0,111
putc5:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc5
        sw      $a0,0xffff000c
        li      $a0,32

putc6:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc6
        sw      $a0,0xffff000c
        li      $a0,87
putc7:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc7
        sw      $a0,0xffff000c
        li      $a0,111
putc8:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc8
        sw      $a0,0xffff000c
        li      $a0,114
```

```
putc9:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc9
        sw      $a0,0xffff000c
        li      $a0,108
putc10:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc10
        sw      $a0,0xffff000c
        li      $a0,100
putc11:
        lw      $t0,0xffff0008
        li      $t1,1
        and     $t0,$t0,$t1
        beqz    $t0,putc11
        sw      $a0,0xffff000c
        j       $ra
```

A.9 節「システムコール」に示されている方法

```
        .data
        .align 2
str:    .asciiz "Hello World"

        .text
        .align 2

main:   li  $v0,4   #print_str のシスコールを$v0 にロード
        la  $a0,str #プリントする文字列のアドレスを syscall の引数
                    #$a0 にロードアドレス命令を行う
        syscall
        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ます．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出来、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

.data がない場合の実行結果

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 がない場合，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，印刷する際にそこ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出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 ルーチンを呼出し、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

```
        .data
        .align 2
str:
        .ascii "The factorial of 10 is "
        .text
        .align 2
print_int:
        li      $v0,1
        syscall
        j       $ra
print_str:
        li      $v0,4
        syscall
        j       $ra
main:
        subu    $sp,$sp,32  #スタックフレームは３２バイト長で確保をする
        sw      $ra,20($sp) #戻りアドレスを退避させる
        sw      $fp,16($sp) #古いフレームポインタを退避
        addiu   $fp,$sp,28  #新しくフレームポインタを設定
        #fact を呼び出して戻ってから、syscall で$LC と fact の返り値をプリントする
        li      $a0,10      #引数は 10
        jal     fact
        move    $t1,$v0     #返り値を t1 に退避
        la      $a0,str     #a0 にテンプレ文のアドレスを記入
        jal     print_str
        move    $a0,$t1     #fact の返り値を保存した t1 を$a0 に収める
        jal     print_int
        #退避してあったレジスタを復元したあと呼出側へ戻る
        lw      $ra,20($sp) #戻りアドレスを復元
        lw      $fp,16($sp) #フレームポインタを復元
        addiu   $sp,$sp,32  #スタックポインタをポップする
        j       $ra
fact:
        subu    $sp,$sp,32  #スタックフレームは３２バイト長
        sw      $ra,20($sp) #戻りアドレスを退避させる
        sw      $fp,16($sp) #古いフレームポインタを退避
        addiu   $fp,$sp,28  #新しくフレームポインタを設定
        sw      $a0,0($fp)  #引数を退避させる# 28($sp) にもってきているもの
        #引数＞０かどうかを調べる。
        #引数＜＝０なら１を返す。
        #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
        #上記を再帰的に繰り返す
        lw      $v0,0($fp)  #n を road させておく
```

```
        bgtz    $v0,factsub #引数が 0 より大きければ再帰処理に飛ぶ
        li      $v0,1       #0 以下なら 1
        j       return
factsub:
        lw      $v1,0($fp)  #n をロードする
        subu    $v0,$v1,1   #n-1
        move    $a0,$v0     #a0 に n-1 に戻る
        jal     fact

        lw      $v1,0($fp)
        mul     $v0,$v0,$v1 #n*fact(n-1)
return: #return 処理
        lw      $ra,20($sp) #戻りアドレスを復元
        lw      $fp,16($sp) #フレームポインタを復元
        addiu   $sp,$sp,32  #スタックポインタをポップする
        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

```
        .data
        .align 2
space:
        .asciiz " "
enter:
        .asciiz "\n"

        .text
        .align 2
test_prime:
        subu    $sp,$sp,32 #stackpointer
        sw      $ra,20($sp) #$ra
        sw      $fp,16($sp) #flamepointer
        addiu   $fp,$sp,28  #set fp
        li      $t0,2   # 1 is not prime
prime_for:
        beq     $t0, $a0, return1   # return1 if n is prime number (i==n)
```

```
        bgt     $t0, $a0, prime_exit    # break the for if n > i
        rem     $t1, $a0, $t0       # $t1 = n % i
        beqz    $t1, prime_exit     # goto Exit_prime if $t1 == 0
        addi    $t0, $t0, 1         # increment i
        j       prime_for           # loop again with incremented i
return1:
        li      $v0,1       #if number is prime
        lw      $ra,20($sp)
        lw      $fp,16($sp)
        addiu   $sp,$sp,32
        j       $ra
prime_exit:
        li      $v0,0       #if number is not prime
        lw      $ra,20($sp)
        lw      $fp,16($sp)
        addiu   $sp,$sp,32
        j       $ra
main:
        subu    $sp,$sp,32 #stackpointer
        sw      $ra,20($sp) #$ra
        sw      $fp,16($sp) #flamepointer
        addiu   $fp,$sp,28  #set fp
        li      $s0,100     #number of max(match<100)
        li      $s1,0       #number of loop(match)
        li      $s2,2       #number to check and print(n=2)
        li      $s3,1       #test_prime(n) == $s3
        li      $s4,10       #if 10,print  \n
while:
        beq     $s0,$s1,exit # s1 == 100 => exit
        move    $a0,$s2     # $s3 => $a0
        jal     test_prime
        bne     $v0,$s3,else    # $v0 != 1
        move    $a0,$s2        # put the prime number
        li      $v0,1
        syscall
        la      $a0, space      # " "
        li      $v0,4
        syscall
        addiu   $s1,$s1,1   #n++
        rem     $t2, $s1, $s4    # $t2 = n % i
        beqz    $t2, print_enter     # goto printenter if $t2 == 0
else:
        addiu   $s2, $s2, 1     # n = n + 1
        j       while            # go to Loop
```

```
exit:
        lw      $ra, 20($sp)  # Restore return address
        lw      $fp, 16($sp)  # Restore frame pointer
        addiu   $sp, $sp, 32  # Pop stack frame
        j       $ra # End this program
print_enter:
        subu    $sp,$sp,32 #stackpointer
        sw      $ra,20($sp) #$ra
        sw      $fp,16($sp) #flamepointer
        addiu   $fp,$sp,28  #set fp
        la      $a0,enter
        li      $v0,4
        syscall
        lw      $ra, 20($sp)  # Restore return address
        lw      $fp, 16($sp)  # Restore frame pointer
        addiu   $sp, $sp, 32  # Pop stack frame
        j       $ra #return
```

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

```
array:
        .space 400

        .data
        .align 2
space:
        .asciiz " "
enter:
        .asciiz "\n"
start:
        .asciiz "To quit, type 0\n\n"

mark:
        .asciiz "\n> "
owari:
        .asciiz "\nGood bye :)\n\n"
excep:
        .asciiz "\nPlease type correct number\n"


        .text
```

```
        .align 2

test_prime:
        subu    $sp,$sp,32 #stackpointer 32bytes
        sw      $ra,20($sp) #$ra
        sw      $fp,16($sp) #flamepointer
        addiu   $fp,$sp,28  #set fp

        li      $t0,2   # syokiti
prime_for:
        beq     $t0, $a0, return1       # return1 if n is prime number (i==n)
        bgt     $t0, $a0, prime_exit    # break the for if n > i
        rem     $t1, $a0, $t0           # $t1 = n % i
        beqz    $t1, prime_exit         # goto Exit_prime if $t1 == 0
        addi    $t0, $t0, 1             # increment i
        j       prime_for               # loop again with incremented i
return1:
        li      $v0,1       #if (number is prime)
        lw      $ra,20($sp)
        lw      $fp,16($sp)
        addiu   $sp,$sp,32
        j       $ra
prime_exit:
        li      $v0,0       #if (number is not prime)
        lw      $ra,20($sp)
        lw      $fp,16($sp)
        addiu   $sp,$sp,32
        j       $ra


main:
        subu    $sp,$sp,32 #stackpointer
        sw      $ra,20($sp) #$ra
        sw      $fp,16($sp) #flamepointer
        addiu   $fp,$sp,28  #set fp

        li    $v0, 4          # for syscall of print_string
        la    $a0, start      # print Start
        syscall               # print info

        li      $s0,100     #number of max(match<100)
        li      $s1,0       #number of loop(match=0)
        li      $s2,2       #number to check and print(n=2)
        li      $s3,1       #if(test_prime(n) == 1)
```

```
        la      $a1,array   #$a1,array

while:
        beq     $s0,$s1,exit    # s1 == 100 => exit
        move    $a0,$s2         # $s2 => $a0
        jal     test_prime

        bne     $v0,$s3,else    # $v0 != 1

        li      $t4, 4          # For array increasing
        addu    $a1, $a1, $t4   # $a1 = $a1 + 4
        sw      $s2, 0($a1)     # Put prime number into array

        addiu   $s1,$s1,1       #n++

else:
        addiu   $s2, $s2, 1     # n = n + 1
        j       while           # go to Loop
exit:
        li      $v0, 4          # for syscall of print_string
        la      $a0, mark       # Mark of ">"
        syscall                 # print mark
        la      $a1, array      # Initialize $a1
        li      $v0, 5          # For syscall of read_int
        syscall                 # read_int for number of prime

        beqz    $v0,end
        bltz    $v0,error
        bgtu    $v0,$s0,error

        move    $t3, $v0        # Put argument into $t3
        addu    $t3, $t3, $t3   # $t3 = $t3 * 2
        addu    $t3, $t3, $t3   # $t3 = $t3 * 2
        addu    $a1, $a1, $t3   # Add address to fetch

        lw      $a0, 0($a1)     # $a0 = *($a1)
        li      $v0, 1          # For syscall of print_int
        syscall                 # Print prime

        j       exit


error:
        li    $v0, 4            # for syscall of print_string
```

```
        la    $a0, excep        # Print error message
        syscall                 # print info
        j     exit


end:
        li      $v0, 4          # for syscall of print_string
        la      $a0, owari
        syscall                 # print

        lw      $ra, 20($sp)    # Restore return address
        lw      $fp, 16($sp)    # Restore frame pointer
        addiu   $sp, $sp, 32    # Pop stack frame

        j       $ra             # End this program
```

3 感想