QtSpim によるアセンブリプログラミング演習 ❶

1. はじめに

科目 “計算機構成” では，学期末試験の他にアセンブリプログラミングの課題を課している．自分の書いたアセンブリプログラムのデバッグや正しく動くかどうかの確認に QtSpim を使う．まずは，習うより慣れろとうことで，配布する 3 つのプログラムを QtSpim で実行する．

2. 演習の準備

2.1 演習で使うアセンブリプログラムのコピー作業

本日の演習で使うアセンブリプログラムは GitHub() で公開しているので，各自コピーすること．アセンブリプログラムを入手するには，`https://github.com/tomisawa/ComputerOrganization` から，右の緑色の `Clone or download` を選択し，さらに `Download Zip` を選択すればよい．

ここでは，ディストップに作成したフォルダ H29COEX にコピーしたものとして話しを進める．

ディストップのディレクトリ H29COEX に，次の 4 つのファイルがあることを確認すること．なお，拡張子 “.s” は表示されない場合もある [*1]．

`fact-exit.s`, `fact-break.s`, `fact-loop.s`, `H29Qtspim.pdf`(本資料)

2.2 QtSpim の設定

図 1 にディスクトップにある QtSpim のショートカットアイコンを示す．このアイコンをクリックして QtSpim を起動する．起動したら**図 2** のように Settings を選択する．図 2 にように，“Accept Pseudo Instructions” だけをチェックし，それ以外はチェックを外す．**特に，“Load Exception Handler” のチェックを外すことを忘れないこと．**

図 1 QtSpim のショートカットアイコン

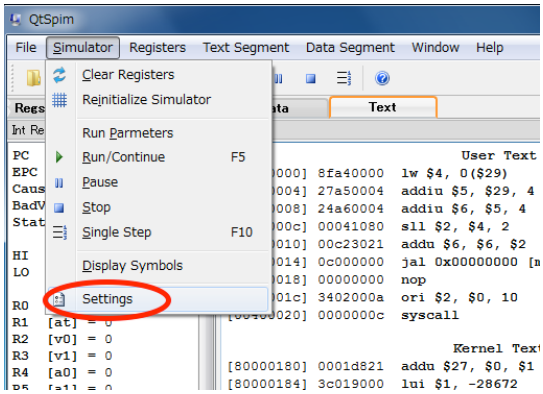

図 2 QtSpim Settings の選択

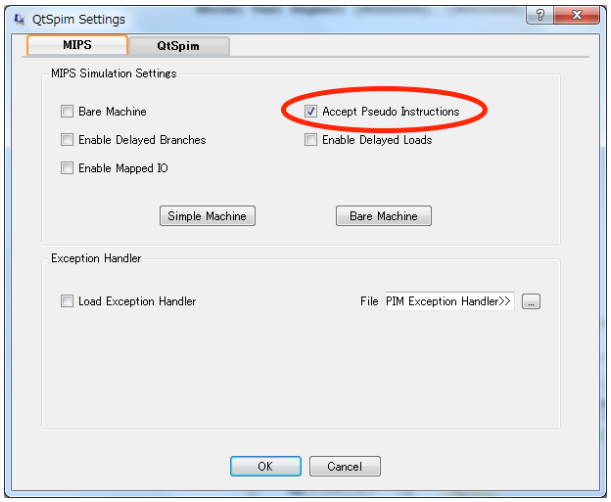

図 3 MIPS の Settings

[*1] 拡張子を表示したい場合，次の URL を参考にして下さい．`https://www.microsoft.com/ja-jp/atlife/tips/archive/windows/tips/252.aspx`

3. QtSpim の使い方とノウハウ

3.1 シングルステップ (Single Step)

課題では，fact 関数のトレースリストを手作業で完成させた．しかし，手作業でのトレースは手間が掛かったり，間違えたりするので，通常はシミュレータ QtSpim を使う．基本的な QtSpim の使い方は 1 命令ずつの実行（シングルステップ）である．1 命令ずつ実行しながら，レジスタの変化や分岐先アドレスなどを確認する．

QtSpim を起動し，`fact-exit.s` をロードし，1 命令ずつ実行してみる．まず，**図 4** のように，QtSpim を起動したら，図中の赤枠のアイコンを選択する．すると，**図 6** のようなポップアップウィンドウが開くので，`fact-exit.s` を選択する．

シングルステップは，**図 5** で赤枠のアイコンを選択するか，ファクンションキー F10 を押せばよい．ここでは，F10 を繰り返し押し，1 命令ずつ実行される様子を見ながら，手作業で作成したトレースリストが正しいかを確認せよ．

図 7 の状態からシングルステップを続けてゆくと，**図 8** に示すように $0040000c_{16}$ 番地の `syscall` 命令で停止する．

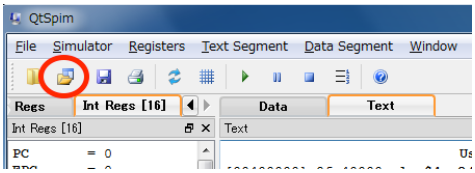

図 4 Reinitialize and load File

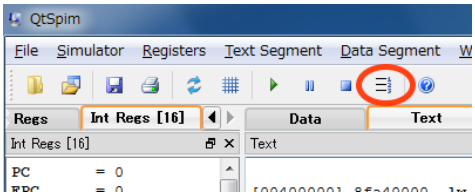

図 5 Single Step

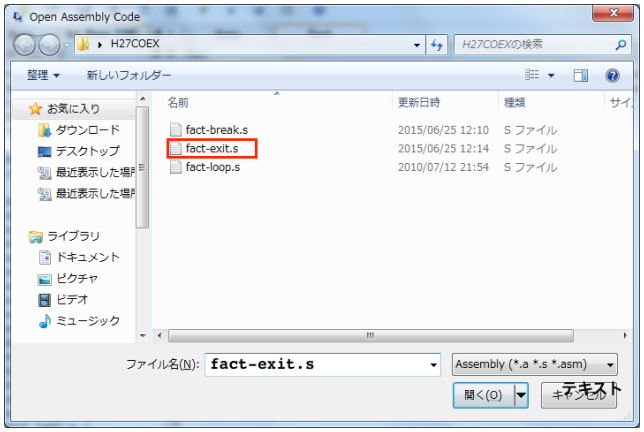

図 6 Open Assembly Code

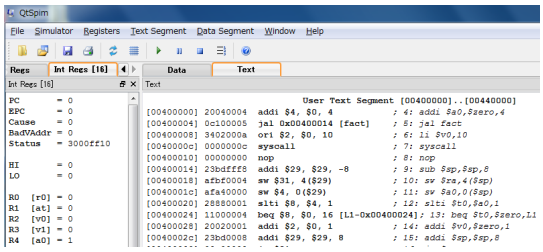

図 7 fact-exit.s のロード

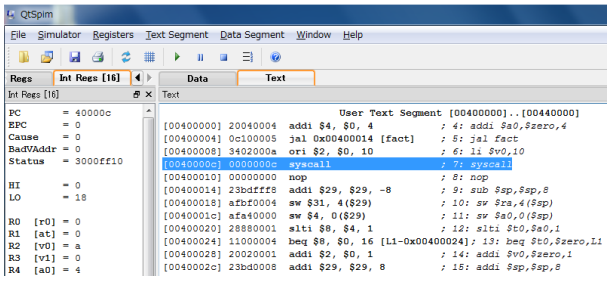

図 8 fact-exit.s の実行終了

3.2 実行と継続 (Run and Continue)

シングルステップではなく，一気に最後まで実行する方法もある．**図 9** の赤枠の緑三角のアイコンを選択すれば，止まることなく実行され，図 8 の状態になる．

しかし，この方法だと `fact` 関数の戻り値を確認できない．なぜならば，関数の戻り値は`$v0` レジスタに格納されるが，その直後の “`li $v0,10`” で`$v0` に 10_{10} を格納しているからである．図 8 の左側にあるレジスタ一覧から，`$v0` の値を見ると，a_{16}（10_{10}）となっている．

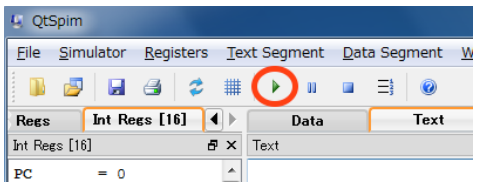

図 9 Run and Continue

3.3 break 命令によるプログラムの停止

プログラムは一気に実行したいが，関数の戻り値は確認したい．教科書には載っていない方法であるが，break 命令を使う方法がある．break 命令について解説している余裕はないので，こうすればよいというノウハウだけ伝える．

リスト 1 は fact-exit.s の始めの部分である．5–6 行目の 2 命令が exit 関数に相当する部分である．break 命令を使用したプログラムを**リスト 2** に示す．6 行目の nop 命令は，no operation 命令で，何もしない命令である． 2つのリストの行数を同じにするためだけに nop 命令を使用した．

リスト 1 fact-exit.s

```
1         .text
2         .globl __start
3 __start:addi $a0,$zero,4
4         jal fact
5         li $v0,10
6         syscall
```

リスト 2 fact-break.s

```
1         .text
2         .globl __start
3 __start:addi $a0,$zero,4
4         jal fact
5         break 2
6         nop
```

QtSpim に fact-break.s をロードし，実行してみる．ロード方法は図 4 にあり，実行方法は図 9 にある．QtSpim が break 命令を実行すると，**図 10** の画面になる．ここで OK を選択すると，**図 11** の画面になり，さらに OK を選択すると終了する．これらのエラーは例外処理に関するエラーであり，ここでは無視してよい．このため図 10 あるいは図 11 で，OK と Abort のどちらを選択してもよい．

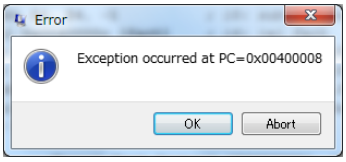

図 10 Error Exception ...

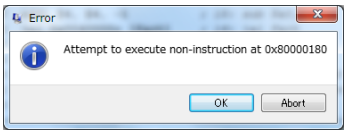

図 11 Error Attempt ...

さて，**図 12** に fact-break.s の実行終了後の画面を示す．プログラムは，00400008_{16} 番地 [*2] の break 命令で停止する．図 12 の左側のレジスタ一覧表で，$v0 レジスタの内容が 18_{16} になっている．プログラムは 4! を計算している．$4! = 24$ であり，24_{10} は 16 進数で 18_{16} である．

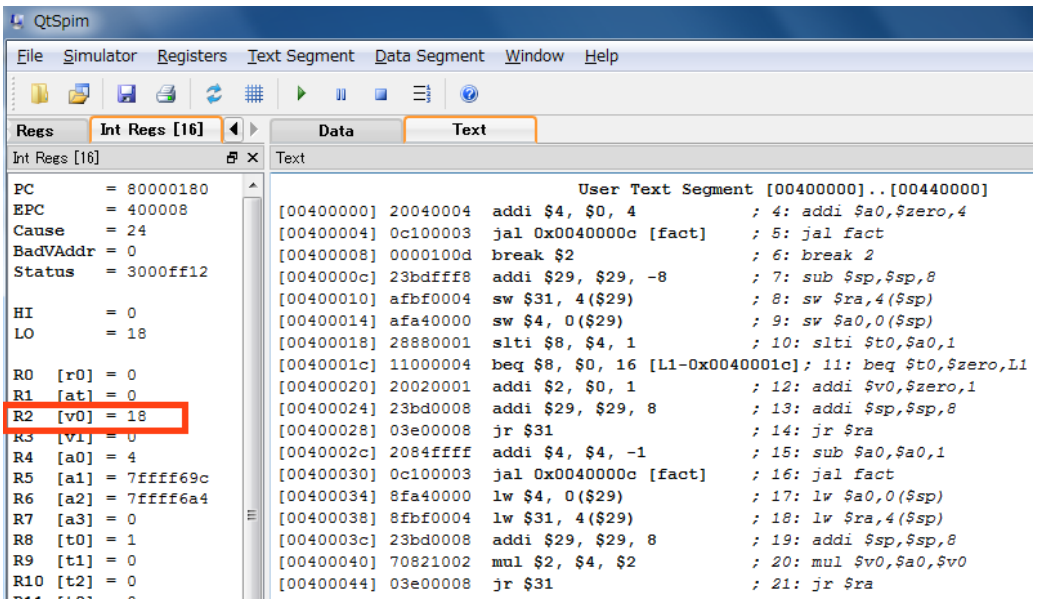

図 12 fact-break の実行終了画面

[*2] 例外処理の関係で，図 12 の左側のレジスタ一覧から，PC レジスタではなくて，EPC レジスタの値を見ること．

3.4 入出力を含めた階乗を計算するアセンブリプログラム

QtSpim に fact-loop.s をロードし，実行してみる．実行すると，コンソール画面には**図 13** のように，入力を促すような文字列 “x=” が出力される．コンソール画面で，数値を入力し return キーを押せば，その数値に応じて**図 14** のように階乗の計算結果が出力される．強制的に終了させない限り，繰り返し階乗が計算される．

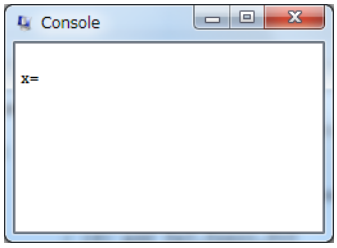

図 13 Input/Output 1

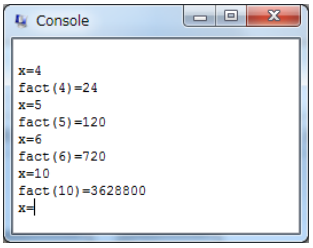

図 14 Input/Output 2

関数のデバッグが一通り終了したら，正しく動作していることを確認する．このようなときは，fact-loop.s のように，繰り返し，いろいろな値で計算できることは便利である．fact-loop.s のリストを**リスト 3** に示す．

リスト 3 fact-loop.s

```
1     .data
2  text1:.asciiz "\nx="
3  text2:.asciiz "fact("
4  text3:.asciiz ")="
5
6     .text
7     .globl __start
8  __start:
9
10    ori $v0,$zero,4      # print_string
11    la $a0,text1
12    syscall
13
14    ori $v0,$zero,5      # read_int
15    syscall
16    add $a0,$zero,$v0
17    add $s0,$zero,$v0
18
19    jal fact
20    add $s1,$zero,$v0
21
22    ori $v0,$zero,4      # print_string
23    la $a0,text2
24    syscall
25
26    ori $v0,$zero,1      # print_int
27    add $a0,$zero,$s0
28    syscall
29
30    ori $v0,$zero,4      # print_string
31    la $a0,text3
32    syscall
33
34    ori $v0,$zero,1      # print_int
35    add $a0,$zero,$s1
36    syscall
37
38    j __start
39
40
41 fact:sub $sp,$sp,8
42      sw $ra,4($sp)
43      sw $a0,0($sp)
44      slti $t0,$a0,1
45      beq $t0,$zero,fact1
46      addi $v0,$zero,1
47      addi $sp,$sp,8
48      jr $ra
49 fact1:sub $a0,$a0,1
50      jal fact
51      lw $a0,0($sp)
52      lw $ra,4($sp)
53      addi $sp,$sp,8
54      mul $v0,$a0,$v0
55      jr $ra
```

4. さいごに

教科書の “まえがき” に書いてあるようにインターネット提供のコンテンツに「SPIM のチュートリアル」が含まれている．より詳しい内容は，教科書の下巻 [1] の付録 B に『アセンブラ，リンカ，SPIM シミュレータ』にある．

参考文献

[1] ジョン・L. ヘネシー，デイビッド・A. パターソン：コンピュータの構成と設計 第 5 版下，pp. 742–811, 日経 BP 社 (2014).