QtSpim によるアセンブリ・プログラミング演習

1. はじめに

科目 “計算機構成” では，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 に慣れておくこと．QtSpim を使えば，第 10 回に配布した階乗を計算する `fact` 関数のトレースが正しいかどうも自分で検証することができる．

本資料では，習うより慣れろとうことで，３つのプログラムを例に，QtSpim の使い方を解説する．自分でアセンブリ・プログラムを書くときは，適当なテキストエディタ[*1]（例えば，サクラエディタ）を使う．

2. 準備

2.1 ３つのアセンブリ・プログラムの入手

こ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．アセンブリ・プログラムを入手するには，Github の画面右上の緑色の `Clone or download` を選択し，さらに `Download Zip` を選択すればよい．

`https://github.com/tomisawa/ComputerOrganization`

フォルダ “R01COEX” に，次の３つのファイルを含めること．拡張子 “.s” は表示されない場合もある[*2]．

`fact-exit.s` `fact-break.s` `fact-loop.s`

以下，スクリーンショットでは，ディストップに作成したフォルダ “R01COEX” にアセンブプログラムがあるとする．デスクトップ以外にフォルダ “R01COEX” を作成した場合は適宜読み換えること．

2.2 QtSpim の設定

PC ルームでは，ログインするごとに QtSpim の設定を確認すること[*3]．

ディスクトップにある QtSpim のショートカットアイコンを**図 1** に示す．このアイコンをクリックして QtSpim を起動する．**図 2** のように，Simulator のプルダウンから，Settings を選択する．**図 3** のように，MIPS タブの “Accept Pseudo Instructions” だけをチェックし，それ以外はチェックしない．“Load Exception Handler” はチェックしてはいけない．

図 1 QtSpim のアイコン

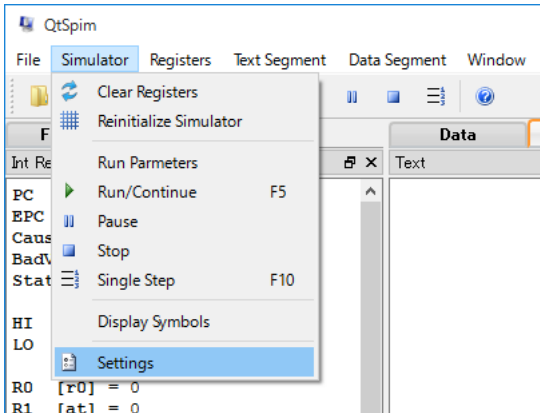

図 2 Settings の選択

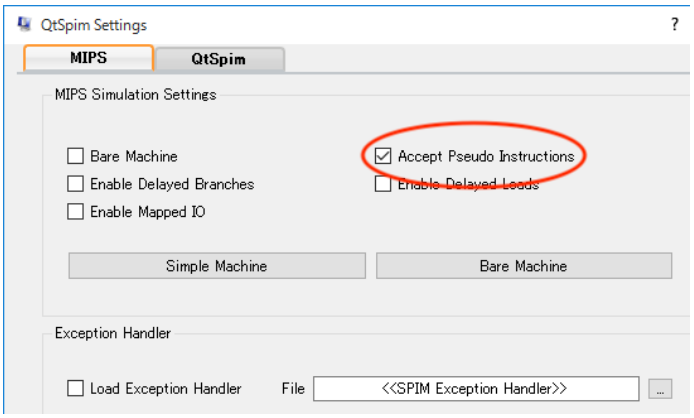

図 3 MIPS タブの内容

[*1] Visual Studio でも作成できますが，動作が重いのでオススメしない．
[*2] 拡張子を表示する方法．`https://support.microsoft.com/ja-jp/help/4479981/windows-10-common-file-name-extensions`
[*3] PC ルームの Windows は固定プロファイルで運用されているので，ログアウトすると設定が初期化されてしまう．

3. QtSpim の使い方

3.1 アセンブリ・プログラムのロード

アセンブリ・プログラムを QtSpim にロードする方法を解説する．QtSpim を起動し，**図 4** の赤枠のアイコンを選択する．すると，**図 5** のようなポップアップウィンドウが開くので，`fact-exit.s` を選択し，OK すれば，**図 6** の画面になる．

この画面の左側にレジスタの内容が表示されており，プログラムカウンタであるレジスタ PC は 0 になっていることに注意せよ．なお，QtSpim では，レジスタを意味する$記号は使われず，レジスタ$t0 は R8[t0] と表示する．

図 6 の右側の Text タブを見ると，004000000 番地から `fact-exit.s` の機械語が格納されていることがわかる．実行するには，PC に実行開始番地である 004000000 を設定すればよい．

メニューバーの一番右にある HELP をクリックすれば，QtSpim の使い方が表示される．

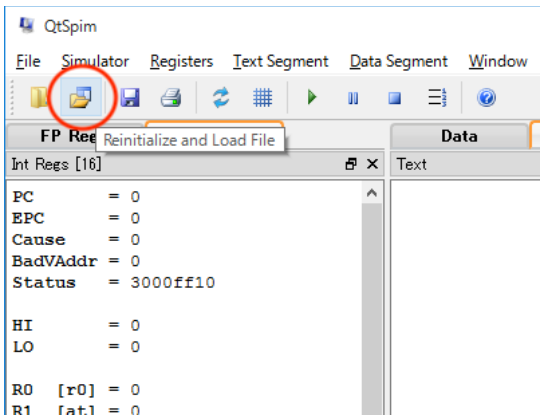

図 4 Reinitialize and load File

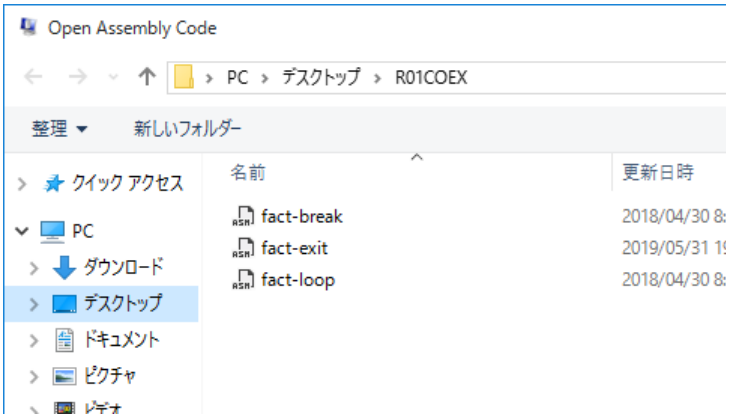

図 5 Open Assembly Code

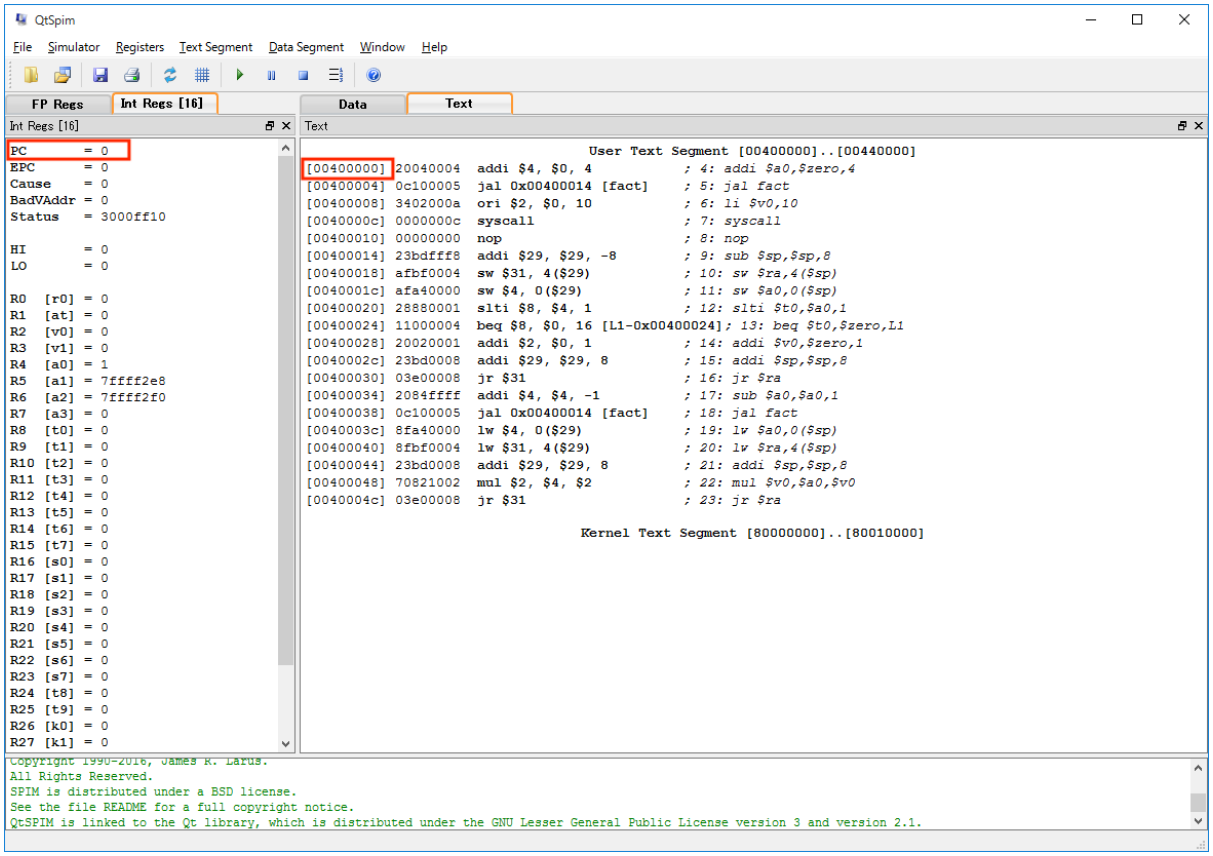

図 6 fact-exit.s のロード

3.2 レジスタの設定

ロードしたプログラムを実行するために，レジスタ PC の値を 00400000 に設定する．レジスタに値を設定するには，図 6 の画面で，値を設定したいレジスタ（ここでは PC）にカーソルを合わせて，右クリックメニューを表示させる．**図 7** のメニューから Change Register Contents を選択する．**図 8** に示すポップアップで，プログラムの実行開始番地である 00400000 を設定し OK を選択すると，**図 9** の画面になる．値を設定したレジスタが赤文字になっている．

メニューバーの一番右にある HELP をクリックすれば，QtSpim の使い方が表示される．

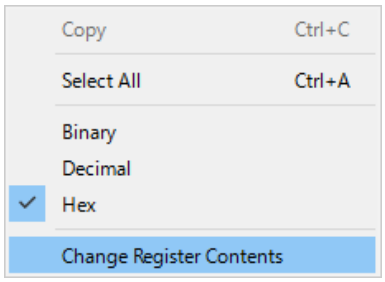

図 7 右クリックメニュー Change Register Contents

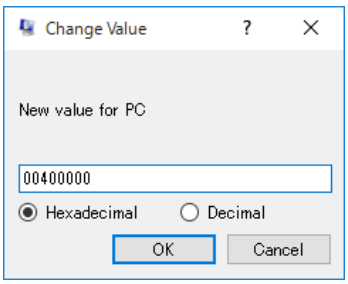

図 8 New value for PC

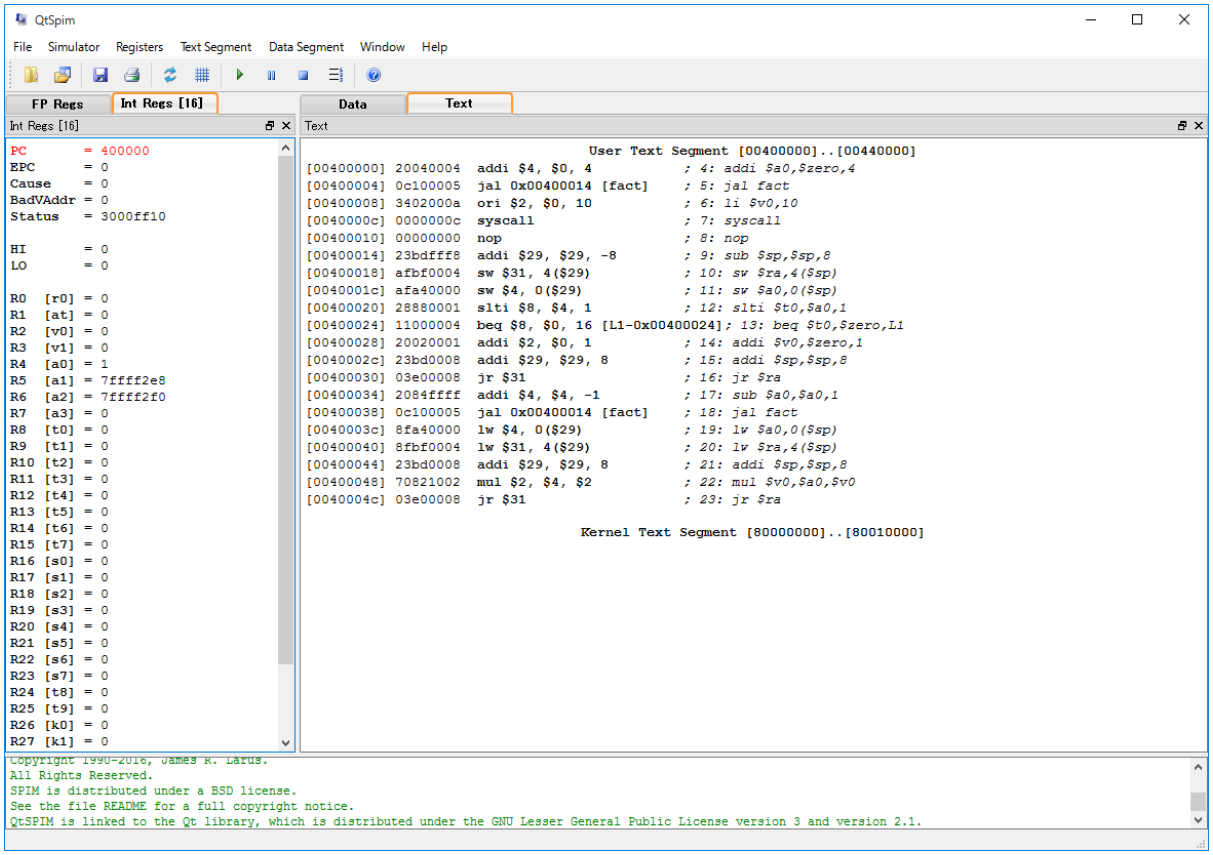

図 9 レジスタ PC に 00400000 を設定した後の画面

3.3 シングルステップ (Single Step)

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．シングルステップを使うと，レジスタの変化を 1 命令ごとに確認できる．

前節までの操作で，QtSpim を起動し，`fact-exit.s` をロードし，レジスタ PC に開始番地 00400000 を設定した．シングルステップは，**図 10** で赤枠のアイコンをクリックするか，ファクンションキー F10 を押せばよい．

PC が 00400008_{16} になったとき，レジスタ R2[v0] の値は 18_{16} になっている．さらに，シングルステップを続けてゆくと，**図 11** に示すように，プログラムは $0040000c_{16}$ 番地の `syscall` 命令を実行して終了する．レジスタ`$v0` に 10 を設定して `syscall` 命令を実行すると，C 言語の `exit` のようにプログラムを終了する [1].

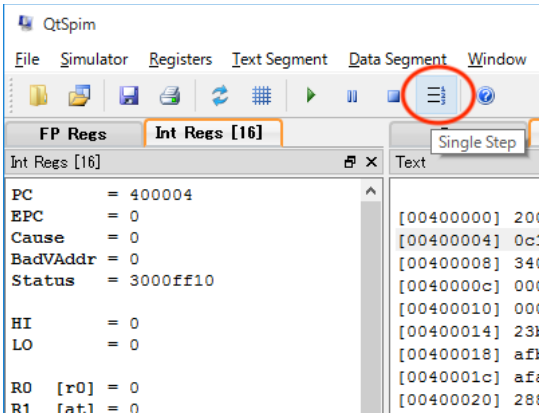

図 10 Single Step

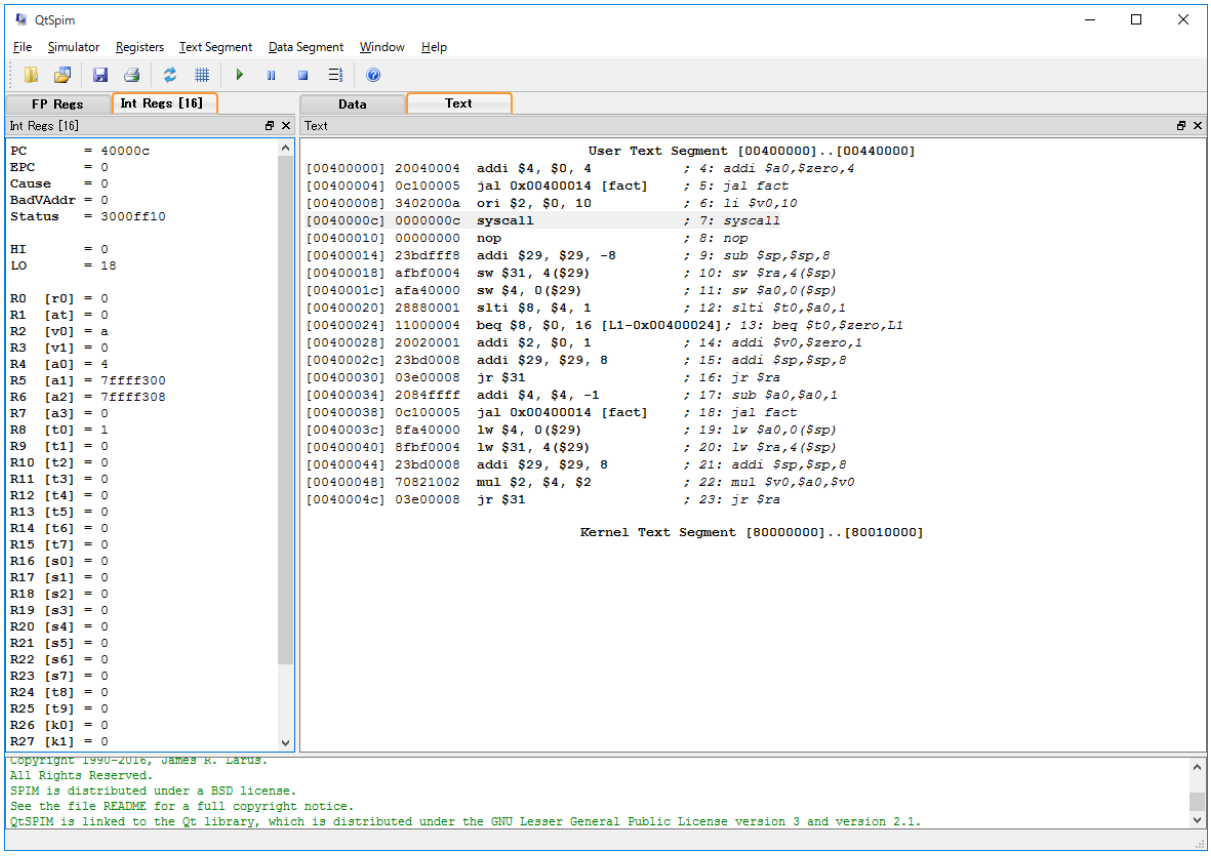

図 11 fact-exit.s の実行終了

3.4 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 12** の赤枠の緑三角のアイコンを選択すれば，例外が発生する命令（この場合は syscall 命令）まで一気に実行され，図 11 の状態になる．

しかし，この実行例だと fact 関数の戻り値が確認できない．なぜならば，関数の戻り値は$v0 レジスタに格納されるが，00400000a 番地の “li $v0,10” で$v0 に 10_{10} を格納しているからである．図 11 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．実際に確認せよ．

fact 関数の戻り値を確認するには，0040000004 番地の jal 命令の実行直後に，プログラムを停めればよい．

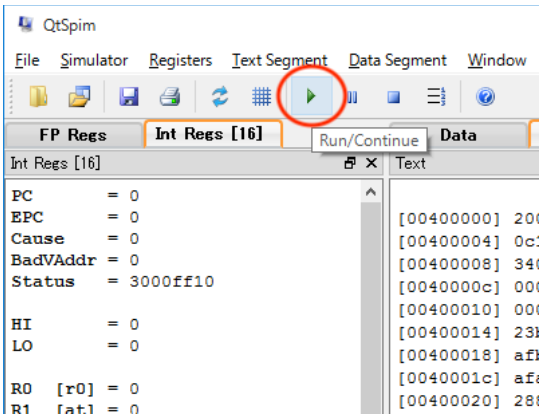

図 12 Run and Continue

3.5 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．ここで解説する方法は，一般的な方法ではなくて，通常は 3.6 で解説するブレークポイントを使う．break 命令は，オペランドで指定した番号の例外を発生させる命令である．QtSpim は break 命令を実行すると例外が発生し，例外ハンドラーに制御が移る．しかし，2.2 で例外ハンドラーをロードしていないので，エラーが発生する．すなわち，break 命令によりエラーを発生して，プログラムを停止させている．エラーが発生すればよいので，break 命令のオペランドはなんでもよい．

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．QtSpim が break 命令を実行すると**図 13** の画面になる．ここで OK を選択すると**図 14** 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[*4]．

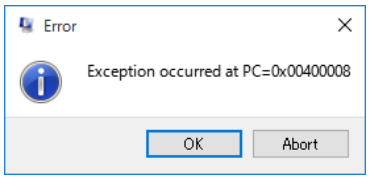

図 13 Error Exception ...

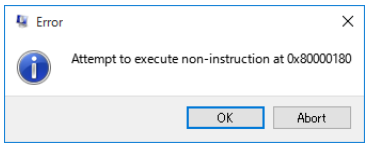

図 14 Error Attempt ...

さて，**図 15** に fact-break.s のプログラムが停止したときの画面を示す．プログラムは，00400008_{16} 番地の break 命令で停止している．図 15 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．PC の 80001080 番地は例外ハンドラーの開始番地であり，EPC の 40000008 は例外を発生した break 命令の番地である．

プログラムを break 命令で停止する方法は一般的ではないので，無理に使う必要はない．プログラムの最後に break 命令を書いておくと，そこで停止するので，ちょっと便利かな程度であろう．

[*4] エラーのメッセージに関心なければ，Abort を選択して終了してもよい．知りたい人は，教科書で例外処理を調べること．

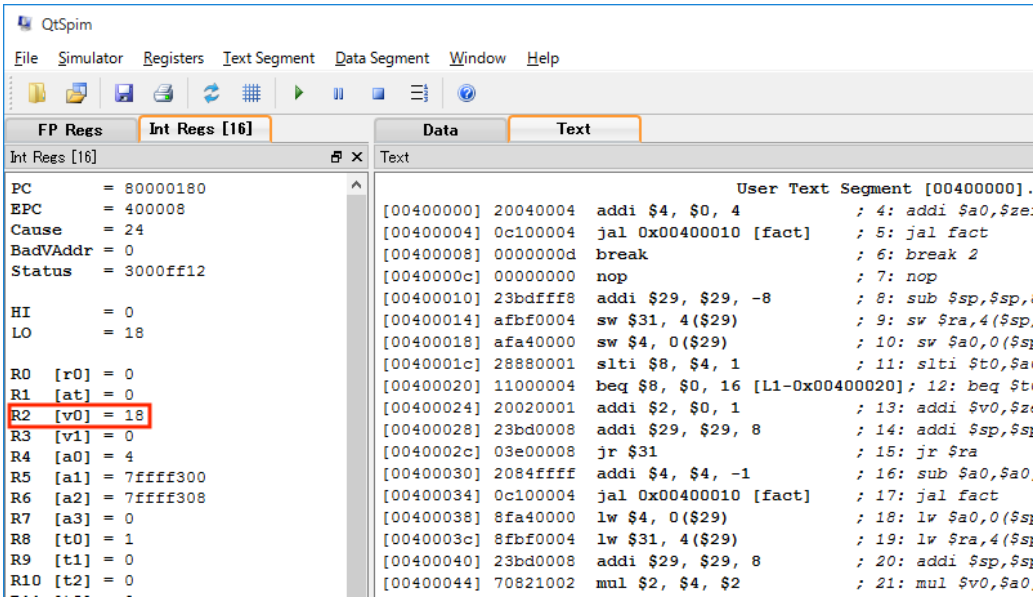

図 15 fact-break の実行終了画面

3.6 ブレークポイント

ブレークポイントを使うと，指定した番地にある命令を，実行前に中断することができる．長いプログラムを，シングルステップだけでデバッグするのは手間がかかる．ブレークポイントを上手に使うと，デバッグの効率も良くなる．

まずは，`fact-exit.s` をロードして，図 6 の状態にする．0040004 番地の `jal` 命令の戻り値である`$v0` の値を確認するには，`jal` 命令を実行した後にプログラムを中断すれば良い．このため，次の命令のある 0040008 番地にブレークポイントを設定する．カーソルで 0040008 を指し，右クリックすれば**図 16** に示すようなメニューが表示される．ここで，Set Breakpoint を選択すれば，ブレークポイントが設定される．

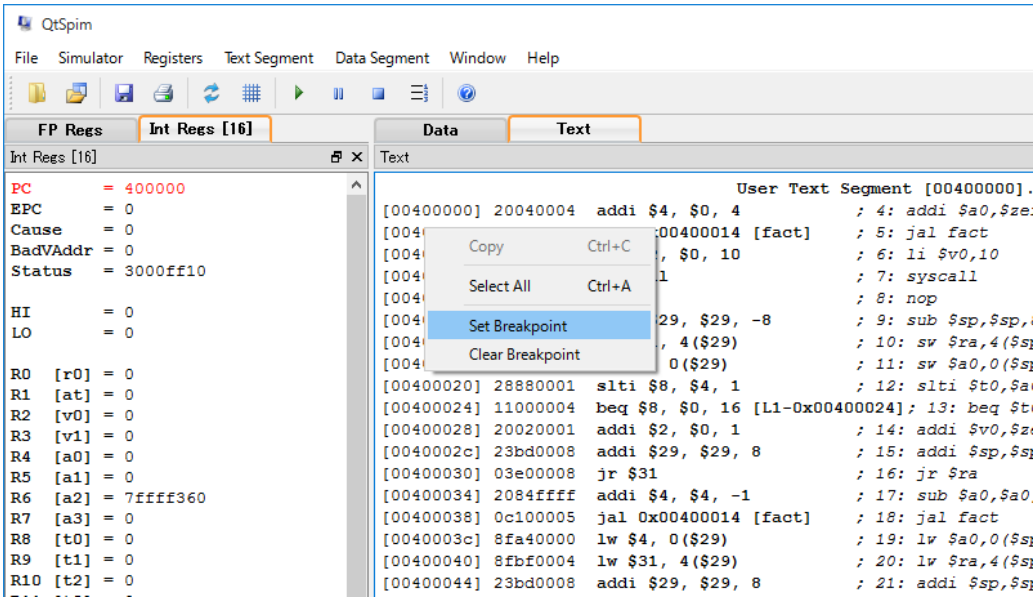

図 16 ブレークポイントの設定

ブレークポイントが設定された番地には，**図 17** に示すように，赤い手が表示される．これで，0040008 番地の ori 命令の実行前にプログラムを中断することができる．図 12 で示したように，緑色の三角をクリックして，一気に実行する．

プログラムは，0040008 番地で中断し，**図 18** の画面になる．0040008 番地の ori 命令は実行されていないので，関数の戻り値であるレジスタ$v0 を見ると 18_{16} になっている．

図 18 の画面で，実行を継続するならば Continue，以降シングルステップで実行するならば Single Step，停止するならば Abort を選択する．この場合では，Abort を選択すればよいだろう．

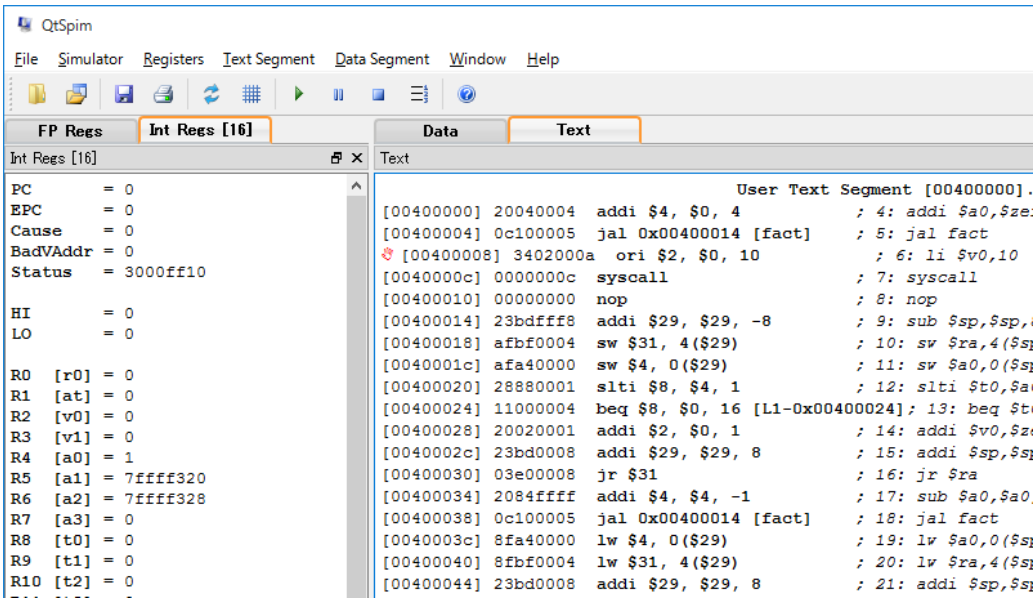

図 17　ブレークポイントの印

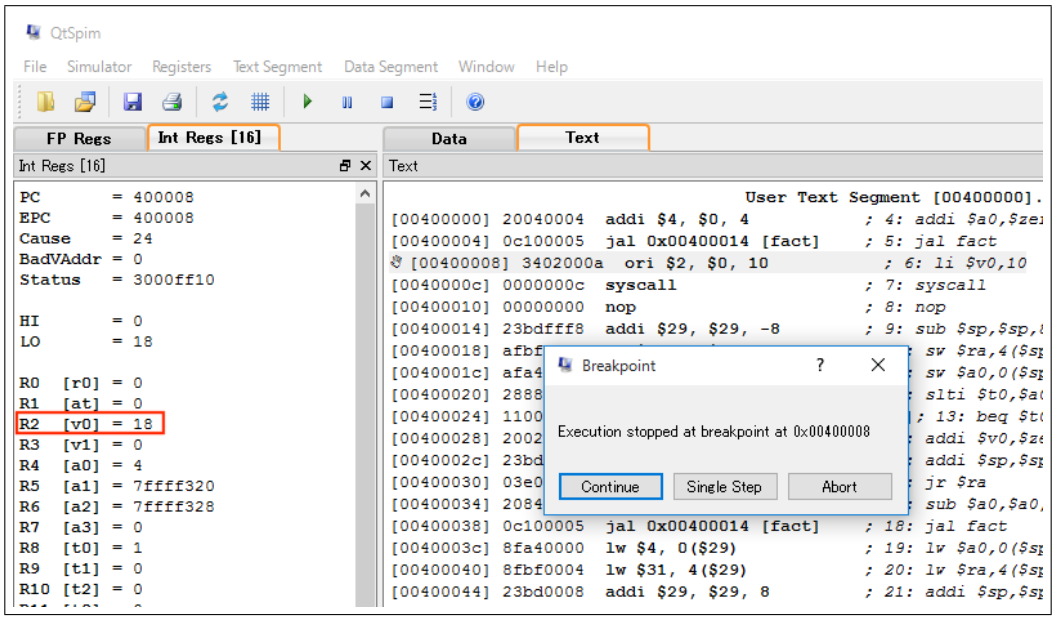

図 18　ブレークポイントの中断

3.7 入出力を含めた階乗を計算するアセンブリプログラム

fact-loop.s をロードし，実行すると，コンソール画面には**図 19** 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 20** のように階乗の計算結果が出力される．

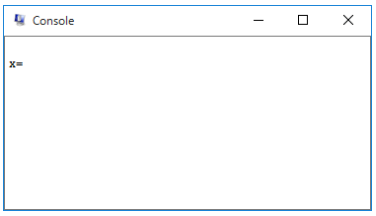

図 19 Input/Output 1

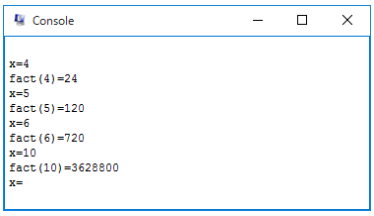

図 20 Input/Output 2

この無限ループのプログラムを停めるには，**図 21** の四角の停止ボタンをクリックすればよい．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．

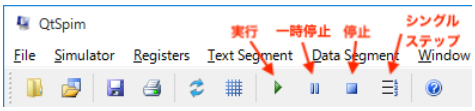

図 21 実行/一時停止/停止/シングルステップ

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. おわりに

本資料では，再帰関数である fact 関数を例に，QtSpim の使い方を解説した．**リスト 3** には．知らない命令が含まれているだろう．ちゃんと理解するには，アセンブラについてもっと勉強する必要がある．アセンブラに関する詳しい解説は，教科書の付録の “Assemblers, Linkers,and the SPIM Simulator” の PDF ファイル [1]（その日本語訳 [2]）にある．

参考文献

[1] Larus, J. R.: Appendix A: Assemblers, Linkers, and the SPIM Simulator, University of Wisconsin-Madison (online), available from ⟨http://pages.cs.wisc.edu/ larus/HP_AppA.pdf⟩ (accessed 2019-06-29).
[2] 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. 566–635, 日経 BP 社 (2014).

