

Interprétation et compilation

TP 2 : Premiers pas en MIPS avec SPIM

Exercice 0:

Prise en main de SPIM.

1.

→ Lancez **spim** et exécutez la commande **help**:

```
vi@vietlaptop: ~/Documents/ITLMAU/Semestre_5_L3/Interprétation_et_Compilation/TP/TP2/Lc_04-tp-premiers-pas-mips_files$ spim
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
(spim) help

SPIM is a MIPS R2000 simulator.
Its top-level commands are:
exit -- Exit the simulator
quit -- Exit the simulator
read "FILE" -- Read FILE of assembly code into memory
load "FILE" -- Same as read
run <ADDR> -- Start the program at optional ADDRESS
step <N> -- Step the program for N instructions
continue -- Continue program execution without stepping
print $N -- Print register N
print $fN -- Print floating point register N
print ADDR -- Print contents of memory at ADDRESS
print symbols -- Print all global symbols
print all_regs -- Print all MIPS registers
print all_regs_hex -- Print all MIPS registers in hex
reinitialize -- Clear the memory and registers
breakpoint <ADDR> -- Set a breakpoint at address
delete <ADDR> -- Delete all breakpoints at address
list -- List all breakpoints
dump [ "FILE" ] -- Dump binary code to spim.dump or FILE in network byte order
dumpnative [ "FILE" ] -- Dump binary code to spim.dump or FILE in host byte order
-- Host of time is assembly instruction to put in memory
vcr -- Newline reexecutes previous command
? -- Print this message

Host commands can be abbreviated to their unique prefix
e.g., ex(tt), re(ad), l(oad), ru(n), s(step), p(rint)
(spim)
```

2.

→ Lancez la commande permettant d'afficher l'état de tous les registres.

```
vi@vietlaptop: ~/Documents/ITLMAU/Semestre_5_L3/Interprétation_et_Compilation/TP/TP2/Lc_04-tp-premiers-pas-mips_files$ spim
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
(spim) print_all_regs
PC      = 00000000    EPC    = 00000000    Cause   = 00000000    BadVAddr= 00000000
Status  = 3000ff10    HI     = 00000000    LO      = 00000000

      General Registers
R0 (r0) = 0          R8 (t0) = 0          R16 (s0) = 0          R24 (t8) = 0
R1 (a0) = 0          R9 (t1) = 0          R17 (s1) = 0          R25 (t9) = 0
R2 (v0) = 0          R10 (t2) = 0         R18 (s2) = 0          R26 (k0) = 0
R3 (v1) = 0          R11 (t3) = 0         R19 (s3) = 0          R27 (k1) = 0
R4 (a0) = 0          R12 (t4) = 0         R20 (s4) = 0          R28 (gp) = 268468224
R5 (a1) = 0          R13 (t5) = 0         R21 (s5) = 0          R29 (sp) = 2147479388
R6 (a2) = 2147479396 R14 (t6) = 0         R22 (s6) = 0          R30 (s8) = 0
R7 (a3) = 0          R15 (t7) = 0         R23 (s7) = 0          R31 (ra) = 0

FIR      = 00009800    FCSR   = 00000000    FCCR    = 00000000    FEXR    = 00000000
FENR     = 00000000

      Double Floating Point Registers
FP0 = 0.000000    FP8 = 0.000000    FP16 = 0.000000    FP24 = 0.000000
FP2 = 0.000000    FP10 = 0.000000    FP18 = 0.000000    FP26 = 0.000000
FP4 = 0.000000    FP12 = 0.000000    FP20 = 0.000000    FP28 = 0.000000
FP6 = 0.000000    FP14 = 0.000000    FP22 = 0.000000    FP30 = 0.000000

      Single Floating Point Registers
FP0 = 0.000000    FP8 = 0.000000    FP16 = 0.000000    FP24 = 0.000000
FP1 = 0.000000    FP9 = 0.000000    FP17 = 0.000000    FP25 = 0.000000
FP2 = 0.000000    FP10 = 0.000000    FP18 = 0.000000    FP26 = 0.000000
FP3 = 0.000000    FP11 = 0.000000    FP19 = 0.000000    FP27 = 0.000000
FP4 = 0.000000    FP12 = 0.000000    FP20 = 0.000000    FP28 = 0.000000
FP5 = 0.000000    FP13 = 0.000000    FP21 = 0.000000    FP29 = 0.000000
FP6 = 0.000000    FP14 = 0.000000    FP22 = 0.000000    FP30 = 0.000000
FP7 = 0.000000    FP15 = 0.000000    FP23 = 0.000000    FP31 = 0.000000
(spim)
```

3.

→ Lancez la commande **run**. Que se passe-t-il ?

```
(spim) run
The following symbols are undefined:
main

Instruction references undefined symbol at 0x00400014
[0x00400014]    0x0c000000 jal 0x00000000 [main]          ; 188: jal main
(spim) █
```

Cette erreur signifie que SPIM ne peut pas trouver la fonction “main” dans ma programme MIPS assembly.

4.

→ Charger en mémoire le fichier “main.s” qui vous est fourni.

```
(spim) load main.s
Must supply a filename to read
(spim) load "main.s"
(spim) █
```

5.

→ Lancez à nouveau la commande run. Que se passe-t-il ?

```
(spim) load "main.s"
(spim) run
(spim) █
```

Rien ne s’est passé ...

6.

→ Que s’est-il passé lorsqu’on a lancé la commande run pour la seconde fois ?

```
(spim) load "main.s"
(spim) run
(spim) run
(spim) █
```

Rien ne s’est passé ...

7.

(a) → Essayez de charger en mémoire le fichier “basic.s”. Que se passe-t-il ?

```
(spin) load "basic.s"
spin: (parser) Label is defined for the second time on line 4 of file basic.s
      main:
        ^
(spin) █
```

L'erreur indique que l'étiquette "main" est définie deux fois dans votre fichier "basic.s".

(b) → Réinitialisez la mémoire et les registres puis chargez en mémoire le fichier "basic.s".

\$ nano basic.s

Et supprimer la ligne
'global main' dans la
fichier.

```
GNU nano 6.2
.text
main:
  nop
  li $t0, 42
  li $t1, 9
  add $t2, $t0, $t1
  move $a0, $t2
  li $v0, 1
  syscall
  jr $ra
```

```
viet@vietLaptop:~/Documents/ITLMAU/Semestre_5_L3/Interprétation_et_Compilation/TP/TP2/lc_04-tp-prenters-pas-mips_files$ nano basic.s
viet@vietLaptop:~/Documents/ITLMAU/Semestre_5_L3/Interprétation_et_Compilation/TP/TP2/lc_04-tp-prenters-pas-mips_files$ spin
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
(spin) load "basic.s"
(spin) █
```

=> Chargez en mémoire réusite!

8.

→ Une fois le fichier "basic.s" chargé, positionnez un breakpoint (commande breakpoint) sur le label main, et lancez l'exécution (run).

```
(spin) load "basic.s"
(spin) breakpoint main
(spin) run
Breakpoint encountered at 0x00400024
(spin) █
```


9.

→ Expliquez chacune des instructions du main de "basic.s".

```
(spim) load "basic.s"
(spim) breakpoint main
(spim) run
Breakpoint encountered at 0x00400024
(spim) step main
[0x00400024] 0x00000000 nop ; 4: nop
(spim) print_all_regs
PC = 00400028 EPC = 00400024 Cause = 00000024 BadVAddr= 00000000
Status = 3000ff10 HI = 00000000 LO = 00000000
General Registers
R0 (r0) = 0 R8 (t0) = 0 R16 (s0) = 0 R24 (t8) = 0
R1 (at) = 0 R9 (t1) = 0 R17 (s1) = 0 R25 (t9) = 0
R2 (v0) = 0 R10 (t2) = 0 R18 (s2) = 0 R26 (k0) = 0
R3 (v1) = 0 R11 (t3) = 0 R19 (s3) = 0 R27 (k1) = 0
R4 (a0) = 0 R12 (t4) = 0 R20 (s4) = 0 R28 (gp) = 268468224
R5 (a1) = 2147479392 R13 (t5) = 0 R21 (s5) = 0 R29 (sp) = 2147479388
R6 (a2) = 2147479396 R14 (t6) = 0 R22 (s6) = 0 R30 (s8) = 0
R7 (a3) = 0 R15 (t7) = 0 R23 (s7) = 0 R31 (ra) = 4194328

FIR = 00009800 FCSR = 00000000 FCCR = 00000000 FEXR = 00000000
FENR = 00000000
Double Floating Point Registers
FP0 = 0.00000 FP8 = 0.00000 FP16 = 0.00000 FP24 = 0.00000
FP2 = 0.00000 FP10 = 0.00000 FP18 = 0.00000 FP26 = 0.00000
FP4 = 0.00000 FP12 = 0.00000 FP20 = 0.00000 FP28 = 0.00000
FP6 = 0.00000 FP14 = 0.00000 FP22 = 0.00000 FP30 = 0.00000
Single Floating Point Registers
FP0 = 0.00000 FP8 = 0.00000 FP16 = 0.00000 FP24 = 0.00000
FP1 = 0.00000 FP9 = 0.00000 FP17 = 0.00000 FP25 = 0.00000
FP2 = 0.00000 FP10 = 0.00000 FP18 = 0.00000 FP26 = 0.00000
FP3 = 0.00000 FP11 = 0.00000 FP19 = 0.00000 FP27 = 0.00000
FP4 = 0.00000 FP12 = 0.00000 FP20 = 0.00000 FP28 = 0.00000
FP5 = 0.00000 FP13 = 0.00000 FP21 = 0.00000 FP29 = 0.00000
FP6 = 0.00000 FP14 = 0.00000 FP22 = 0.00000 FP30 = 0.00000
FP7 = 0.00000 FP15 = 0.00000 FP23 = 0.00000 FP31 = 0.00000
```

=> La fonction main de le programme effectue une addition simple de 42 et 9, stocke le résultat dans \$a0, puis affiche ce résultat à l'aide d'une interruption système avant de quitter le programme. Le résultat attendu serait l'affichage de "51" à la sortie.

Exercice 1:

Premiers programmes (fichiers à récupérer sur le page du cours).

1.

→ Chargez et exécutez ce programme.

```
(spim) read "hello.s"
(spim) run
Hello, world!
(spim)
```

2.

→ Ouvrez le fichier "add.s" et décrivez ce qui se passe dedans. Comment récupère-t-on la valeur retournée par l'appel à **read_int** ?

```
(spim) load "add.s"
(spim) run
Please enter a first number: 7
Please enter a second number: 9
The sum of these numbers is: 16
(spim) █
```

- La section **.text** commence par déclarer **main** comme une étiquette globale.
- Le programme commence par afficher le message "**Please enter a first number:** " en utilisant l'appel système **4** (**li \$v0, 4** et **syscall**). Cette instruction utilise **\$v0** pour indiquer que nous voulons afficher une chaîne de caractères et **\$a0** pour spécifier l'adresse de la chaîne à afficher.
- Ensuite, le programme utilise l'appel système **5** (**li \$v0, 5** et **syscall**) pour lire un entier saisi par l'utilisateur. La valeur entière lue est stockée dans le registre **\$v0**.
- Le programme déplace ensuite la valeur lue de **\$v0** vers le registre **\$t0** en utilisant l'instruction **move \$t0, \$v0**. Cela permet de conserver le premier nombre saisi par l'utilisateur.
- Le programme répète les étapes 2 à 4 pour le deuxième nombre saisi par l'utilisateur.
- Après avoir obtenu les deux nombres, le programme effectue une addition en utilisant l'instruction **add \$a0, \$t0, \$t1** et stocke le résultat dans le registre **\$a0**.
- Il utilise ensuite l'appel système **1** (**li \$v0, 1** et **syscall**) pour afficher la somme calculée.
- Le programme affiche ensuite un message "**The sum of these numbers is:** " pour donner un contexte à la sortie.
- Enfin, il affiche un saut de ligne en utilisant un appel système pour la nouvelle ligne.
- Le programme se termine en effectuant un saut (**jr \$ra**), ce qui permet de quitter proprement la fonction **main** et de terminer le programme.

- La section **.data** du programme contient des chaînes de caractères utilisées pour les messages d'invite et de sortie, comme "Please enter a first number: ", "Please enter a second number: ", "The sum of these numbers is: " et "\n". Ces chaînes sont définies à l'aide de l'instruction **.ascii**.

==> Ce programme demande à l'utilisateur de saisir deux nombres, effectue leur addition, affiche le résultat et se termine. Le résultat de l'appel à **read_int** (la valeur entrée par l'utilisateur) est stocké dans les registres **\$v0** et **\$t0**, puis utilisé pour l'opération d'addition.

3.

Fichier "funcall.s"

→ Chargez et exécutez ce programme. Que constatez-vous ?

```
(spim) read "funcall.s"
(spim) run
Please enter a first number: 8
Please enter a second number: 9
The sum of these numbers is: 17
^C
Execution interrupted
(spim)
```

=> L'exécution du programme ne s'est pas arrêtée après l'impression des résultats jusqu'à ce que j'appuie sur **Ctrl + C**

4.

→ Expliquez ce comportement (aidez-vous des fonctionnalités de debug si nécessaire).

- Dans le fichier "funcall.s", la fonction **main** utilise l'instruction **jal add_user_num** pour appeler la fonction **add_user_num**. L'instruction **jal** effectue un saut vers l'étiquette spécifiée (**add_user_num** dans ce cas) tout en sauvegardant l'adresse de retour dans le registre **\$ra** (Return Address). Cela signifie que lorsque la fonction **add_user_num** se termine, le programme essaie de retourner à l'instruction suivant **jal add_user_num**, qui se trouve être **jr \$ra** dans la fonction **main**.

- Le programme ne se termine pas tant que **add_user_num** n'a pas été exécuté, et il est réentrant à chaque appel de cette fonction. C'est pourquoi il ne s'arrête pas après avoir imprimé les résultats. Il reste dans une boucle infinie, car à chaque itération de la boucle, il appelle à nouveau **add_user_num**.

5.

→ Proposez une correction pour le programme "funcall.s".

```
GNU nano 6.2
.text
.globl main

add_user_num:
    li $v0, 4
    la $a0, num1q
    syscall

    li $v0, 5
    syscall
    move $t0, $v0

    li $v0, 4
    la $a0, num2q
    syscall

    li $v0, 5
    syscall
    move $t1, $v0

    li $v0, 4
    la $a0, sum
    syscall

    add $a0, $t0, $t1
    li $v0, 1
    syscall

    li $v0, 4
    la $a0, nl
    syscall

    jr $ra

main:
    jal add_user_num
    li $v0, 10
    syscall

.data
num1q: .asciiz "Please enter a first number: "
num2q: .asciiz "Please enter a second number: "
sum:   .asciiz "The sum of these numbers is: "
nl:    .asciiz "\n"
```

- Ouvrir par la commande: **\$ nano funcall.s**

- Puis ajouter une instruction **syscall** avec le code de service **10** à la fin de la fonction **main**. Cela terminera proprement l'exécution du programme.

```
(spim) read "funcall.s"
(spim) run
Please enter a first number: 9
Please enter a second number: 7
The sum of these numbers is: 16
(spim)
```

6.

→ Quelles sont les limites de la solution que vous avez proposée ?

Les limites de la solution que j'ai proposé:

- Terminaison immédiate : L'ajout de **li \$v0, 10** suivi de **syscall** à la fin de la fonction **main** provoque la terminaison immédiate du programme.

=> le programme se termine immédiatement après avoir exécuté la fonction **add_user_num**.

- Perte de la possibilité de réutilisation : En ajoutant la terminaison immédiate à la fin de la fonction **main**, on ne peut exécuter la fonction **add_user_num** qu'une seule fois.

- Synchronisation manuelle : Pour que le programme se termine, nous devons ajouter manuellement l'instruction **li \$v0, 10** suivi de **syscall** à chaque point d'achèvement du programme où on veut qu'il se termine.

=> Cela nécessite une gestion manuelle de la terminaison, ce qui peut être complexe si notre programme comporte plusieurs fonctions ou interactions utilisateur.

7.

→ Avez-vous une idée de comment remédier à ces limitations ?

Oui, je peux utiliser des boucles pour permettre à notre programme de rester actif et de continuer à interagir avec l'utilisateur

Par exemple: Je peux demander à l'utilisateur s'il souhaite effectuer une autre opération après avoir affiché le résultat de l'addition. Si l'utilisateur choisit continuer, le programme retourne au début de la boucle pour effectuer une autre opération.

8.

Le fichier "loop.s"

→ Expliquez le fonctionnement de la boucle.

```
(spim) load "loop.s"
(spim) run
Please enter a first number: 7
Please enter a second number: 9
The sum of these numbers is: 16
Please enter a first number: 9
Please enter a second number: 8
The sum of these numbers is: 17
Please enter a first number: 6
Please enter a second number: 8
The sum of these numbers is: 14
(spim)
```

==> Ce programme MIPS assembly utilise une boucle pour effectuer trois itérations de la fonction **add_user_num**.

Pendant chaque itération, l'utilisateur est invité à saisir deux nombres, dont la somme est ensuite affichée. Après trois itérations, le programme se termine.

Exercice 2:

Votre premier programme

1.

- Créer un fichier 'sum.s' par la commande: **\$ nano sum.s**

```

GNU nano 6.2
.data
prompt:      .ascii "Enter a positive integer (n): "
result_msg:  .ascii "The sum of the first n integers is: "
nl:          .ascii "\n"

.text
.globl main

main:
    li $v0, 4
    la $a0, prompt
    syscall

    li $v0, 5
    syscall
    move $t0, $v0

    bgtz $t0, calculate_sum
    li $v0, 4
    la $a0, nl
    syscall
    j end_program

calculate_sum:
    li $t1, 0
    li $t2, 1

sum_loop:
    beq $t2, $t0, print_result

    add $t1, $t1, $t2

    addi $t2, $t2, 1

    j sum_loop

print_result:
    li $v0, 4
    la $a0, result_msg
    syscall

    move $a0, $t1
    li $v0, 1
    syscall

end_program:
    li $v0, 10
    syscall

```

- Puis chargez en mémoire avec spim et **run** :

```
vlet@vietLaptop:~/Documents/ITLMAU/Semestre_5_L3/Interprétation_et_Compilation/TP/TP2/lc_04-tp-premiers-pas-mips_files$ nano sum.s
vlet@vietLaptop:~/Documents/ITLMAU/Semestre_5_L3/Interprétation_et_Compilation/TP/TP2/lc_04-tp-premiers-pas-mips_files$ spim
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
(spim) load "sum.s"
(spim) run
Enter a positive integer (n): 99
The sum of the first n integers is: 4851(spim) quit
vlet@vietLaptop:~/Documents/ITLMAU/Semestre_5_L3/Interprétation_et_Compilation/TP/TP2/lc_04-tp-premiers-pas-mips_files$
```