Computer Architecture (../index.html) /

Exercise 1 : Fundamentals of assembly language programming

Purpose of this exercise

- Getting used to SPIM
- Understanding how to use MIPS assembler instructions
- To prepare programs to be used in future tasks

In this exercise, we perform simple programming using the assembler of MIPS processor. To operate this program, use the SPIM simulator to check if it is actually working properly. The program created here will be a test program that you run on a small MIPS machine you design at a later time.

How to start SPIM

Use the SPIM simulator located in the following location.

> /home/course/comparch/bin/xspim (for solsv* machine)

Other directories such as /home/course/comparch/bin also contain xspim of different versions, but please do not use them. Since this simulator loads the assembler source file from the current directory, when you do the exercise you create your own directory, and "~nisim/pub/comparch/xspim" or "~ nisim/pub/comparch/ Please start SPIM as xspim.i386.

Documentation of SPIM

The documentation for SPIM can be found in HERE (http://webfs-int.u-aizu.ac.jp/%7Ebenab/classes/ca/2008/doc/spim_documentation.pdf)

Exercises of this week

There are three "examples" and four "tasks" below. For "example" please simulate using SPIM for the published program and confirm that you get the correct results. Please think about the program by yourself and verify the result by simulation.

In this exercise, please pay attention to the following points.

- Do not use instructions other than the table (assemble.html)

  If you use other instructions, it will not work on MIPS processors designed using Cadence. You can also use pseudo-instructions such as la, li, and move which are replaced with instructions in the table as a result.

- Simulation is performed by step execution.

  For the same reason as above, the exit system call can not be used. Therefore, when you use the run command, xspim hangs.

- Insert a blank line at the end of the assembler program.

  Otherwise it may be a syntax error.

Example 1. C = A + B

This example adds the values of two variables in memory and writes the result in memory.

Assume that variables A, B and C are arranged in memory as follows. Each is a 32-bit word variable, occupying 4 bytes of address space per variable.

Address	Initial value(dec)	Result(dec)
A	19	19
B	75	75
C	0	94

MIPS processor operation can only be done between registers. Therefore, in order to calculate the data in the memory, you must load the data into the register once, store the operation result in the memory after performing the operation. Such a program is shown below.

```
        .data
A:      .word 19
B:      .word 75
C:      .word 0
        .text
main:   lw   $8, A
        lw   $9, B
        add  $10, $8, $9
        sw   $10, C
exit: j  exit
```

First is the explanation of the assembler syntax.

Syntax	Description
.data	Indicate the beginning of the data segment
A: .word 19	Define a variable with contents of "19" with a 32-bit numeric value so that its address can be referred to by "A".
B: .word 75	Define a variable with content of "75" with a 32-bit numerical value so that its address can be referred to by "B".
C: .word 0	Define a variable with "32" bit value and content "0" so that its address can be referred to by "C".
.text	Beginning of text segment

Next is the actual calculation.

Instruction	Behavior	Contents of $8 after operation	Contents of $9 after operation	Contents of $10 after operation
lw $8, A	Load data at address A to register $8 (lw $8, A is short for lw $8, A ($0))	19	Indeterminate	Indeterminate

Instruction	Behavior	Contents of $8 after operation	Contents of $9 after operation	Contents of $10 after operation
lw $9, B	Load data at address B into register $9	19	75	Indeterminate
add $10, $8, $9	Store registers $8 and $9 and store the result in $10	19	75	94
sw $10, C	Store register $10 at address C	19	75	94
j exit	Jump to label exit (termination processing)	19	75	04

In the MIPS processor, the registers freely available as temporary variables are stipulated from $8 to $15. In particular, please be aware that $1 is reserved for assembler and can not be used unexpectedly because it may be rewritten unexpectedly. SPIM also defines $t0 to $t7 as aliases for $8 to $15.

Then, please save the above program to a file such as "sum1.s" by cutting & pasting and simulate on SPIM. Click the "load" button to load the source file and "step" button to execute the program.

Exercise 2-1. S = (A + B – C) | 3

Create a program to calculate the expression S = (A + B – C) | 3 containing three variables and constants and simulate on SPIM. "|" Is a bitwise OR operation.

Please set the initial value of each variable as follows.

Address	Data (dec)
A	19
B	75
C	10
S	Result

Example 2. S = A[0] + A[1] + A[2] + A[3]

(Reference of memory by register)

This example shows how to use register relative address to indirectly refer to memory. If the addresses of consecutive data are arranged regularly like an array, you can easily write the program by accessing the memory while adding or subtracting the address using the register. For example, suppose that data is lined up as follows.

Address	Data (dec)
A	19
A + 4	75
A + 8	10

Address	Data (dec)
A + 12	15
S	Result

In such a case, if we know the first address A, the next address can be obtained by adding 4 to A. Therefore, in order to calculate the sum of the four elements, you need to write the following program.

ex_2_sample_0.s (ex_2_sample_0.s)

```
.data
A:      .word 19
.word 75
.word 10
.word 15
S:      .word 0
.text
main:
        or   $8, $0, $0   # Initialize to 0
        la   $9, A        # Put the address of A in $ 9. In fact it replaces the ori cc

        lw   $10, 0($9)
        add  $8, $8, $10

        lw   $10, 4($9)
        add  $8, $8, $10

        lw   $10, 8($9)
        add  $8, $8, $10

        lw   $10, 12($9)
        add  $8, $8, $10


        sw   $8, S
exit:   j exit
```

In addition, the above program can be rewritten as shown below. In this form, the processing for each data becomes exactly the same instruction sequence, so it is suitable for obtaining the sum using the loop described later.

ex_2_sample_1.s (ex_2_sample_1.s)

```
        .data
A:      .word 19
        .word 75
        .word 10
        .word 15
S:      .word 0
        .text
main:   or   $8, $0, $0   # Initialize to 0
        la   $9, A        # Put the address of A in $ 9. In fact it replaces the ori cc


        lw   $10, 0($9)
        add  $8, $8, $10
        addi $9, $9, 4    # Move to next address

        lw   $10, 0($9)
        add  $8, $8, $10
        addi $9, $9, 4

        lw   $10, 0($9)
        add  $8, $8, $10
        addi $9, $9, 4

        lw   $10, 0($9)
        add  $8, $8, $10

        sw   $8, S
exit:   j exit
```

Please cut and paste the above second assembler source file into the editor and check the operation on SPIM.

If you understand how to access the memory while incrementing the register, it is acceptable.

Example 3. $\Sigma_{i=1..n}n$

In this example, we will deal with a program that simply adds 1 to n. When written in C language, it becomes as follows.

ex_2_sample_2.c (ex_2_sample_2.c)

```
int main(void)
{
    int i = 0;
    int n = 15;
    int s = 0;
    while(i != n)
    {
        i++;
        s += i;
    }
    return 0;
}
```

Processing like $\Sigma_{i=1..n}n$ can be described by using loop as above. The assembler implements a loop using branch instructions and comparison instructions.

Let's assume the initial state of memory as follows.

Address	Data (dec)
n	15
s	Result

Well, I will convert this code to assembler. Use the beq (branch on equal) command to write a program with only the instructions listed in the table of instructions to be used. Let's say that the loop variable i is $8 and the register containing the value of n is $9, the loop part can be written as follows.

```
         or  $8, $0, $0        # i = 0
loop:    beq  $8, $9, loopend  # if i == n, jump to loopend
         addi $8, $8, 1        # i++

        ... content of the loop ...

         j loop
loopend:
```

In addition, if there is a bne (branch on not equal) instruction,

```
        or   $8, $0, $0       # i = 0
loop:
        addi $8, $8, 1        # i++

        ... content of the loop ...

        bne  $8, $9, loop     # if i != n, jump to loop
```

Or reversing the order of the loop makes it even easier,

```
loop:
    ... content of the loop ...

    addi $9, $9 -1  # n = n - 1
    bne $9, $0, loop # if n != 0, jump to loop
```

Although I can write it, unfortunately it can not be used on a small MIPS machine created this time, please use the first format.

The example source file as a whole is as follows.

```
    .data
n:
    .word 15
s:
    .word 0
    .text
main:
    or   $8, $0, $0       # i = 0
    lw   $9, n            # load n
    or   $10, $0, $0      # s = 0

loop:
    beq  $8, $9, loopend  # if i == n, jump to loopend
    addi $8, $8, 1        # i++
    add  $10, $10, $8     # s += i
    j    loop

loopend:
    sw   $10, s           # store s

exit:
    j    exit
```

Exercise 2-2. $\Sigma_{i=1..n}A[i]$

Create a program to calculate the sum of N elements arrayed in memory and check the operation with the SPIM simulator.

Please use the code below for the beginning part of the program.

```
    .data
N:
    .word 10    # The length of Array
A:
    .word 8     # A[0] = 8
        .word 4     # A[1] = 4
    .word 7
    .word 12
    .word 13
    .word 19
    .word 23
    .word 43
    .word 56    # A[8] = 56
    .word 32    # A[9] = 32
S:
    .word 0
    .text
main:
    #  ... write programm ...
```

Exercise 2-3. Copy of Array Contents

Create a program to copy the contents of array A consisting of N elements arranged in memory to another array B and check the operation with the simulator. Please use the code below for the beginning of the program.

```
    .data
N:
    .word 10    # The length of Array
A:
    .word 8     # A[0] = 8
    .word 4     # A[1] = 4
    .word 7
    .word 12
    .word 13
    .word 19
    .word 23
    .word 43
    .word 56    # A[8] = 56
    .word 32    # A[9] = 32
B:
    .space 40   # The storage size of array B is 40 byte.
    .text
main:
    #   ... write programm here ...
```

Exercise 2-4. Bubble Sort

Using the bubble sort algorithm, create a program that sorts N integers stored in array A in ascending order, and use the assembler to check the operation with the simulator. Please use the value of N and the initial value of array A that are same as task 2-3. For reference, a part of C language program is shown below.

```
for( i = 0; i < N-1; i++ ) {
    for( j = N-2; j >= i; j-- ) {
        if( A[j] > A[j+1] ) {
            tmp = A[j];
            A[j] = A[j+1];
            A[j+1] = tmp;
        }
    }
}
```

About issue report

Please describe the program for tasks 2-1 to 2-4, attach the assembler source file and submit the report. You do not need to mention the example in the report.

Suppliment

- A detailed explanation of the program may be embedded as a comment in the source file.
- For exercise 2-1, 2-2, please write the execution result (calculation result) of the program in the report.
- In exercise 2-3 and 2-4, please obtain the memory value after execution displayed on xspim by cut & paste, window image dump, etc. and include it in the report.