LAB #04 Flow Control

MIPS Jump and Branch Instructions

For unconditional jump, the instruction j label is used where label is the address of the target instruction.
There are two MIPS conditional branch instructions that branch based on the condition whether two registers are equal or not: `beq, bne`
Four additional MIPS instructions are provided based on comparing the content of a register with 0: `bltz, bgtz, blez, bgez`
Note that MIPS does not provide the instructions `beqz` and `bnez` as these can be implemented using the `beq` and `bne` instructions with register `$0` used as the second operand.

Operation	Meaning
`j    label`	`jump to label (unconditional jump)`
`beq  Rs, Rt, label`	`branch to label if (Rs == Rt)`
`bne  Rs, Rt, label`	`branch to label if (Rs != Rt)`
`bltz Rs, label`	`branch to label if (Rs < 0)`
`bgtz Rs, label`	`branch to label if (Rs > 0)`
`blez Rs, label`	`branch to label if (Rs <= 0)`
`bgez Rs, label`	`branch to label if (Rs >= 0)`

Fig. 1 MIPS Jump and Branch Instructions

Set on less than

MIPS also provides four set on less than instructions as follows:

Operation	Meaning
`slt   rd, rs, rt`	`rd = (rs < rt)? 1 : 0`
`sltu  rd, rs, rt`	`unsigned <`
`slti  rt, rs, im16`	`rt = (rs < im16)? 1 : 0`
`sltiu rt, rs, im16`	`unsigned <`

Fig. 2 set on less than

In Fig. 2, `im16` means 16 bits immediate value. **Note** that the instructions `slt` / `slti` are used for signed comparison while instructions `sltu` / `sltiu` are used for unsigned comparison.

For example, assume that `$s0 = 1` and `$s1 = -1` (= `0xffffffff`), then the following two instructions produce different results as shown below:

```
slt $t0, $s0, $s1   # results in $t0 = 0
sltu $t0, $s0, $s1  # results in $t0 = 1
```

Pseudo Instructions

Pseudo instructions are instructions introduced by an assembler as if they were real instructions. We have seen an example of a pseudo instruction before, which is the `li` instruction. Pseudo instructions are useful as they facilitate programming in assembly language.

For example, the MIPS processor does not have the following conditional branch comparison instructions:

Operation	Meaning
`blt, bltu`	`branch if less than (signed/unsigned)`
`ble, bleu`	`branch if less or equal (signed/unsigned)`
`bgt, bgtu`	`branch if greater than (signed/unsigned)`
`bge, bgeu`	`branch if greater or equal (signed/unsigned)`

Fig. 3 Conditional branch comparison instructions

The reason for not implementing these instructions as part of the MIPS instruction set is to reduce the complexity of instructions (i.e., more instructions → more decoding units). Also, they can be easily implemented based on a set of two instructions.

For example, the instruction `blt $s0, $s1, label` can be implemented using the following sequence of two instructions:

```
slt $at, $s0, $s1
bne $at, $zero, label
```

Similarly, the instruction `ble $s2, $s3, label` can be implemented using the following sequence of two instructions:

```
slt $at, $s3, $s2
beq $at, $zero, label
```

The following figure shows more examples of pseudo instructions. **Note** that the assembler temporary register `$at(=$1)` is reserved for its own use.

Operation	Real instructions
`move $s1, $s2`	`addu $s1, $zero, $s2`
`not  $s1, $s2`	`nor  $s1, $s2, $zero`
`li   $s1, 0xabcd`	`ori  $s1, $zero, 0xabcd`
`li   $s1, 0xabcd1234`	`lui  $at, 0xabcd` `ori  $s1, $at, 0x1234`
`sgt  $s1, $s2, $s3`	`slt  $s1, $s3, $s2`
`blt  $s1, $s2, label`	`slt  $at, $s1, $s2` `bne  $at, $zero, label`

Fig. 4 Example of pseudo instructions.

NOTE: If you want to check other pseudo instructions, check the **help** of MARS.

Flow Control

We can translate any high-level flow construct into assembly language using the jump, branch and set-less-than instructions.

if statement

if statement	Assembly

`if (a == b)`
`      c = d + e;`
`else`
`      c = d - e;`
	`      bne $s0, $s1, else`
	`      addu $s2, $s3, $s4`
	`      j exit`
	`else: subu $s2, $s3, $s4`
	`exit: …`

Fig. 5 if statement

In Fig. 5, we assume that the variables *a, b, c, d,* and *e* are stored in registers `$s0` through `$s4` respectively.

if statement with condition	Assembly

`if (($s1 > 0) && ($s2 < 0))`
`{`
`    $s3++;`
`}`
	`      blez $s1, next        # skip if false`
	`      bgez $s2, next        # skip if false`
	`      addiu $s3, $s3, 1     # both are true`
	`next: …`

Fig. 6 if statement with compound condition with AND

if statement with condition	Assembly

`if (($sl > $s2) \|\| ($s2 > $s3))`
`{`
`    $s4 = 1;`
`}`
	`       bgt $s1, $s2, body # execute body part`
	`       ble $s2, $s3, next # skip body part`
	`body:  li $s4, 1          # set $s4 to 1`
	`next: …`

Fig. 7 if statement with compound condition with OR

Loop

Assumption of the following examples is that variable *i* is stored in register `$s0` and *n* is stored in register `$s1`.

for loop	Assembly

`for (i=0; i<n; i++)`
`{`
`    loop body`
`}`
	`        li $s0, 0          # i = 0`
	`ForLoop:`
	`        bge $s0, $s1, EndFor`
	`        loop body`
	`        addi $s0, $s0, 1   # i++`
	`        j ForLoop`
	`EndFor:`

Fig. 8 for loop

while loop	Assembly

`i = 0;`
`while (i < n) {`
`    loop body`
`    i++;`
`}`
	`# Same as for loop example`

Fig. 9 while loop

do-while loop	Assembly

`i = 0;`
`do {`
`    loop body`
`    i++;`
`} while (i < n)`
	`        li $s0, 0        #i = 0`
	`WhileLoop:`
	`        loop body`
	`        addi $s0, $s0, 1 # i++`
	`        blt $s0, $s1, WhileLoop`

Fig. 10 do-while loop

Task to do

1. Write a program that asks the user to enter an integer and then displays the number of 1's in the binary representation of that integer. For example, if the user enters 9, then the program should display 2.
2. Write a program that asks the user to enter two integers: n1 and n2 and prints the sum of all numbers from n1 to n2. For example, if the user enters n1=3 and n2=7, then the program should display the sum as 25.
3. Write a program that asks the user to enter an integer and then display the hexadecimal representation of that integer.
4. The Fibonacci sequence are the numbers in the following integer sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

   Write a program that asks the user to enter a positive integer number *n* then prints the *n*-th Fibonacci number. The following algorithm can be useful:

```
Input: n positive integer
Output: n-th Fibonacci number
```

Pseudocode

`Fib0 = 0, Fib1 = 1`
`for (i=2; i<=n; i++) do`
`    temp = fib0`
`    fib0 = fib1`
`    fib1 = temp + fib1`
`if (n > 0)`
`    fib = fib1`
`else fib = 0`

NOTE:

1. Write a report according to the LAB questions and submit a word (or pdf) file.
2. For each question, **write** assembly code (**not** screen capture… hard to read).
3. For each problem save an asm file and **zip** them with your report.
   (file name of your asm file should be lab.4.1.asm, lab.4.2.asm, etc.)