Lab 3. Jump and Branch Instructions

Goals

After this laboratory exercise, you should know how to use jump and branch instructions. You should also be able to implement high level programming language structures such as conditional statement (if-then-else), loop and selection statement (switch-case)

References

- RISC-V documents, textbook.
- The RISC-V Instruction Set Manual: riscv-spec-20191213.pdf

Preparation

Before start this laboratory, you should review textbook and read the entire laboratory exercise in detail. You also need to review Laboratory Exercise 2

Assignments at Home and at Lab

- Conditional branch instructions (if the condition occurs then jump to target address (label)): RISC-V supports 6 conditional branch instructions.

`beq rs1, rs2, label`	branch if =	`if (rs1 == rs2) PC = BTA`
`bne rs1, rs2, label`	branch if ≠	`if (rs1 ≠ rs2) PC = BTA`
`blt rs1, rs2, label`	branch if <	`if (rs1 < rs2) PC = BTA`
`bge rs1, rs2, label`	branch if ≥	`if (rs1 ≥ rs2) PC = BTA`
`bltu rs1, rs2, label`	branch if < unsigned	`if (rs1 < rs2) PC = BTA`
`bgeu rs1, rs2, label`	branch if ≥ unsigned	`if (rs1 ≥ rs2) PC = BTA`

- Unconditional branch instructions: RISC-V supports 2 unconditional branch instructions:

`jalr rd, rs1, imm`	jump and link register	`PC = rs1 + SignExt(imm), rd = PC + 4`
`jal rd, label`	jump and link	`PC = JTA, rd = PC + 4`

  Pseudo-instruction jump (**`j label`**), jump to the label (target address) without condition.

- The instruction **`slt t1, t2, t3`**, set t1 = 1 if t2 < t3, else clear t1 = 0.

- The follow assignments show the implementation of programming structures as in high-level languages
  - `if/else`
  - `for/while`
  - `switch/case`

Home Assignment 1

This home assignment implements "if-then-else" statement using some fundamental instructions, such as slt, addi, jump and branch.

```
if (i <= j)
     x = x + 1
     z = 1
else
     y = y - 1
     z = 2 * z
```

At first, you should draw the algorithm chart for this statement. After that, you read this example carefully, try to clarify the function of each intructions.

```
# Laboratory Exercise 3, Home Assignment 1
.text
start:
   # TODO:
   # Initialize i to s1
   # Initialize j to s2

   # Cách 1:
   # blt s2, s1, else   # if j < i then jump else

   # Cách 2:
   slt      t0, s2, s1     # set t0 = 1 if j < i else clear t0 = 0
   bne      t0, zero, else # t0 != 0 means t0 = 1, jump else

then:
   addi  t1, t1, 1         # then part: x=x+1
   addi  t3, zero, 1       # z=1
   j     endif             # skip "else" part
else:
   addi  t2, t2, -1        # begin else part: y=y-1
   add   t3, t3, t3        # z=2*z
endif:
```

Home Assignment 2

The following example demonstrates how to implement loop statement. This program computes the sum of elements of array A.

Description of algorithm by C language:

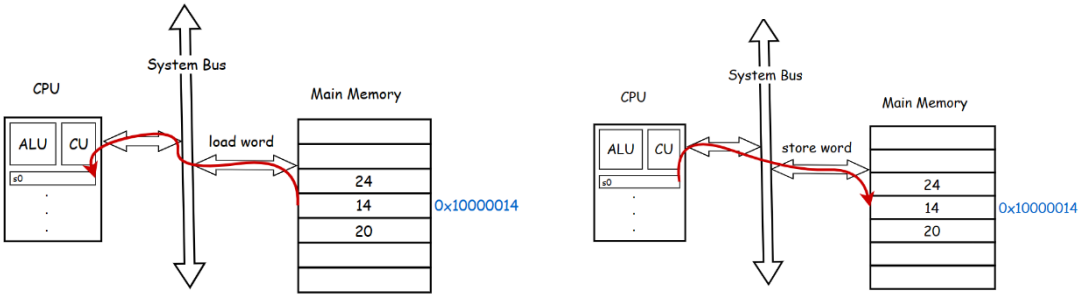
```
sum = 0;
for (int i = 0; i < n; i += step)
     sum += A[i];
```

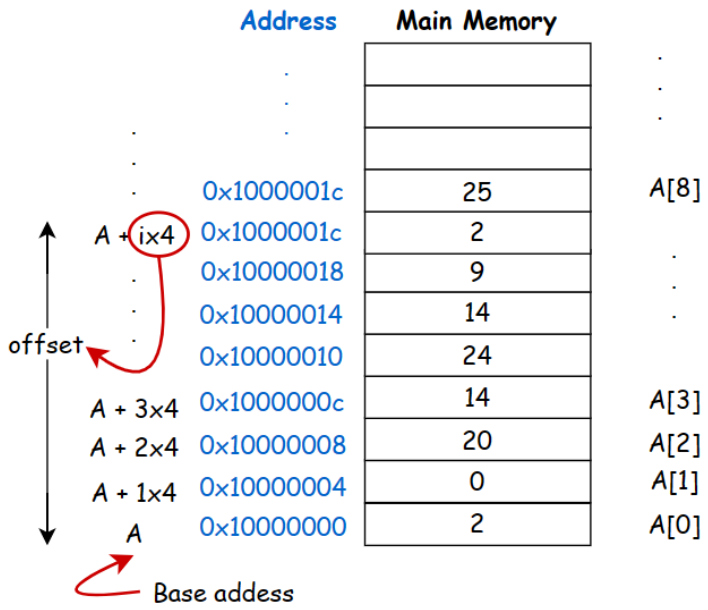
Description of algorithm by pseudo-code:

```
     sum = 0
     i = 0
```

```
loop:     if (i >= n) goto endloop
          sum = sum + A[i]
          i = i + step
          goto loop
endloop:
```

Accessing to an array: **`A`** store the starting address of the array A – the address of the element **`A[0]`**. Assume that the size of each element is a word (or 4 byte), the address of the element **`A[i]`** is computed by **`A + 4×i.`** Using the instruction **lw** for reading data from memory to a register, and the instruction **sw** for writing the data from a register to memory.





Assuming that the index i, the starting address of A, the comparison constant n, step and sum are found in registers s1, s2, s3, s4 and s5, respectively. You should try to understand each line in this code

```
# Laboratory 3, Home Assigment 2
.data
   A: .word 1, 3, 2, 5, 4, 7, 8, 9, 6
.text
   # TODO: Initialize s2, s3, s4 registers
   li s1, 0    # i = 0
   li s5, 0    # sum = 0
loop:
```

```
   slt t2, s1, s3    # check loop condition i < n
   beq   t2, zero, endloop # if i >= n then end loop
   add t1, s1, s1    # t1 = 2 * s1
   add t1, t1, t1    # t1 = 4 * s1 => t1 = 4*i
   add t1, t1, s2    # t1 store the address of A[i]
   lw    t0, 0(t1)   # load value of A[i] in t0
   add s5, s5, t0    # sum = sum + A[i]
   add s1, s1, s4 # i = i + step
   j  loop           # go to loop
endloop:
```

Home Assignment 3

A switch/case statement allows multiway branching based on the value of an integer variable. In the following example, the switch variable test can assume one of the three values 0, 1, 2 and a different action is specified for each case.

```
switch(test) {
      case 0:
            a=a+1; break;
      case 1:
            a=a-1; break;
      case 2:
            b=2*b; break;
      }
```

Assuming that **a** and **b** are stored in registers $s2 and $s3. You should read this code section carefully, understand how to implement switch/case statement.

```
# Laboratory Exercise 3, Home Assignment 3
.data
   test: .word 0
.text
   la s0, test    # Nạp địa chỉ của biến test vào s0
   lw s1, 0(s0)   # Nạp giá trị của biến test vào s1
   li t0, 0       # Nạp giá trị cần kiểm tra
   li t1, 1       # Nạp giá trị cần kiểm tra
   li t2, 2       # Nạp giá trị cần kiểm tra
   beq s1, t0, case_0
   beq s1, t1, case_1
   beq s1, t2, case_2
   j default
case_0:
   addi s2, s2, 1 # a = a + 1
   j continue
case_1:
   sub s2, s2, t1 # a = a - 1
   j continue
case_2:
   add s3, s3, s3 # b = 2 * b
   j continue
default:
continue:
```

Assignment 1

Create a new project to implement the code in Home Assignment 1. Initialize for i and j variable. Compile and upload to the simulator. Run this program step by step, observe the changing of memory and the content of registers at each step.

Assignment 2

Create a new project implementing the code in Home Assignment 2. Initialize for i, n, step, sum variables and array A. Compile and upload to the simulator. Run this program step by step, observe the changing of memory and the content of registers by each step. Try to test with some more cases (change the value of variables).

Assignment 3

Create a new project implementing the code in Home Assignment 3. Compile and upload to the simulator. Run this program step by step; observe the changing of memory and the content of registers by each step. Change the value of test variable and run this program some times to check all cases.

Assignment 4

Modify the Assignment 1, so that the condition tested is:

a. `i < j`
b. `i >= j`
c. `i + j <= 0`
d. `i + j > m + n` (với m và n được lưu trong các thanh ghi khác)

Assignment 5

Modify the Assignment 2, so that the condition tested at the end of the loop is:

a. `i <= n`
b. `sum >= 0`
c. `A[i] != 0`

Assignment 6

Using all of above instructions and statements, create a new project to implement this function: find the element with the largest absolute value in a list of integers. Assuming that this list is store in an integer array and we know the number of elements in

Conclusions

Before you pass the laboratory exercise, think about the questions below:

- Which registers are affected by a branch instruction?
- What is diferrence between jump and branch instructions ?
- Why are they implemented by two different formats ?