Homework #1

Author: ZHU He (Ryker Zhu) **Student ID**: 202283930036 **Date**: April 21, 2024

Exercises (The answer to the questions are all in orange): .....

Exercise 2.4 For the following MIPS assembly instructions, what is the corresponding C statement? Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `$s0`, `$s1`, `$s2`, `$s3`, and `$s4`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `$s6` and `$s7`, respectively.

```
sll     $t0, $s0, 2     # $t0 = f * 4
add     $t0, $s6, $t0   # $t0 = &A[f]
sll     $t1, $s1, 2     # $t1 = g * 4
add     $t1, $s7, $t1   # $t1 = &B[g]
lw      $s0, 0($t0)     # f = A[f]
addi    $t2, $t0, 4
lw      $t0, 0($t2)
add     $t0, $t0, $s0
sw      $t0, 0($t1)
```

Solution The corresponding C statement for the MIPS assembly instructions is

```
B[g] = A[f + 1] + A[f];
```

Comments for the rest of the instructions are given below:

```
sll     $t0, $s0, 2     # $t0 = f * 4
add     $t0, $s6, $t0   # $t0 = &A[f]
sll     $t1, $s1, 2     # $t1 = g * 4
add     $t1, $s7, $t1   # $t1 = &B[g]
lw      $s0, 0($t0)     # f = A[f]
addi    $t2, $t0, 4     # $t2 = &A[f + 1]
lw      $t0, 0($t2)     # $t0 = A[f + 1]
add     $t0, $t0, $s0   # $t0 = $t0 + A[f]
sw      $t0, 0($t1)     # B[g] = $t0
```

.....

Exercise 2.8 Translate the following MIPS code to C. Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `$s0`, `$s1`, `$s2`, `$s3`, and `$s4`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `$s6` and `$s7`, respectively.

```
addi    $t0, $s6, 4
add     $t1, $s6, $0
sw      $t1, 0($t0)
lw      $t0, 0($t0)
add     $s0, $t1, $t0
```

Solution The corresponding C statement for the MIPS assembly instructions is

```
f = A + A;
```

Note that `A` is a integer pointer, so there is no need to use "address-of" operator. In this case, the variable `f` is also a pointer rather than a value. Comments for the instructions are given below:

```
addi    $t0, $s6, 4     # $t0 = &A[1]
add     $t1, $s6, $0    # $t1 = A + 0
sw      $t1, 0($t0)     # A[1] = A
```

```
lw      $t0, 0($t0)     # $t0 = A
add     $s0, $t1, $t0   # f = $t1 + $t2
```

Exercise 2.17 Assume the following register contents: `$t0 = 0xAAAAAAAA`, `$t1 = 0x12345678`

1. For the register values shown above, what is the value of `$t2` for the following sequence of instructions?

   ```
   sll     $t2, $t0, 44
   or      $t2, $t2, $t1
   ```

2. For the register values shown above, what is the value of `$t2` for the following sequence of instructions?

   ```
   sll     $t2, $t0, 4
   andi    $t2, $t2, -1
   ```

3. For the register values shown above, what is the value of `$t2` for the following sequence of instructions?

   ```
   srl     $t2, $t0, 3
   andi    $t2, $t2, 0xFFEF
   ```

Solution The values of `$t2` are:

1. `$t2 = 0xAAAAAAAA << 44 | 0x12345678`
   `= 0xAAAAAAAA << 12 | 0x12345678` (Since shifting left by a number that is greater than 32 will take its remainder of dividing by 32, considering that the MIPS registers are 32 bits wide. In this case, 44 modulo 32 equals to 12.)
   `= 0xAAAAA000 | 0x12345678 = 0xBABEF678`
2. `$t2 = 0xAAAAAAAA << 4 & 0xFFFFFFFF` (Since the two's complement of the integer -1 is `0xFFFFFFFF`)
   `= 0xAAAAAAA0` (Since ANDing with all 1s will produce itself)
3. `$t2 = 0xAAAAAAAA >> 3 & 0xFFEF = 0x00005545`

Exercise 2.18 Find the shortest sequence of MIPS instructions that extracts bits 16 down to 11 from register `$t0` and uses the value of this field to replace bits 31 down to 26 in register `$t1` without changing the other bits of registers `$t0` and `$t1` (Be sure to test your code using `$t0 = 0` and `$t1 = 0xffffffff`. Doing so may reveal a common oversight.).

Solution My version of MIPS subroutine (only 7 instructions):

```
srl     $t0, $t0, 11    # Shift right register $t0 logically by 11 bits
# Now the lower 11 bits of $t0 are removed
sll     $t0, $t0, 26    # Shift left register $t0 by 26 bits
# The reason why 26bits are shifted is that we should shift left back
# the 11 bits we have shiftted right in the first step (26 = 11 + 15)
# Now the upper 15 bits of $t0 are also removed
addi    $t2, $0, 0x03FF # Produce the higher bit mask
sll     $t2, $t2, 16    # Now the mask becomes 0x03FF0000
ori     $t2, $t2, 0xFFFF
# Now we have obtained the full mask $t2 = 0x03FFFFFF
# We shall obtain the 32bit integer by combining two 16bit ones together
# since the I-format instruction can only contain a 16bit immediate
and     $t1, $t1, $t2   # AND $t1 with the mask
or      $t1, $t1, $t0   # Replace the MSBs via ORing
```

Exercise 2.21 Assume `$t0` holds the value `0x00101000`. What is the value of `$t2` after the following instructions?

```
        slt     $t2, $0, $t0
        bne     $t2, $0, ELSE
        j       DONE
ELSE:   addi    $t2, $t2, 2
DONE:
```

Solution As for the instruction `slt $t2, $0, $t0`, register `$t2` would be set 1 if $0 <$ `$t0`. Since the value `$t0` holds is greater than zero, `$t2` would be set 1. The second instruction would branch into the `ELSE` label if `$t2` $\neq 0$ and in this case, the condition holds, so 2 would be added to `$t2`. Thus, the final result for `$t2` is `1 + 2 =` 3.

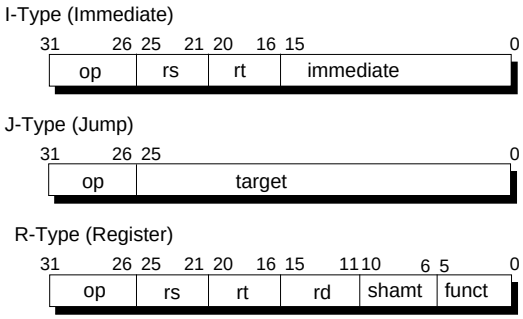
Exercise 2.23 Consider a proposed new instruction named `rpt`. This instruction combines a loop's condition check and counter decrement into a single instruction. For example, `rpt $s0, loop` would do the following:

```
if(x29 > 0) {
    x29 = x29 - 1;
    goto loop;
}
```

1. If this instruction were to be implemented in the MIPS instruction set, what is the most appropriate instruction format?
2. What is the shortest sequence of MIPS instructions that performs the same operation?

Solution

1. Most appropriate one is the I format, as is shown in the figure below (Image courtesy of IDT R30xx Family Software Reference Manual), since the target address of branching can be encoded into the 16 bit immediate field and the register can also be encoded into Rt field especially considering that register in the Rt can either be read or written.



op	6-bit operation code
rs	5-bit source register specifier
rt	5-bit target (source / destination) or branch condition
immediate	16-bit immediate, branch displacement or address displacement
target	26-bit jump target address
rd	5-bit destination register specifier
shamt	5-bit shift amount
funct	6-bit function field

Figure 1: Three Instruction Formats of MIPS Architecture

2. The `rpt` instruction can be implemented by using three existent ones:

```
blez    $s0, exit       # if ($s0 <= 0) goto exit;
addi    $s0, $s0, -1    # x29 -= 1;
j       loop            # goto loop;
exit:
```

Exercise 2.24 Consider the following MIPS loop:

```
LOOP:   slt     $t2, $0, $t1
        beq     $t2, $0, DONE
        subi    $t1, $t1, 1
        addi    $s2, $s2, 2
        j       LOOP
DONE:
```

1. Assume that the register `$t1` is initialized to the value 10. What is the value in register `$s0` assuming the `$s0` is initially zero?
2. For each of the loops above, write the equivalent C code routine. Assume that the registers `$s1`, `$s2`, `$t1`, and `$t2` are integers `A`, `B`, `i`, and `temp`, respectively.
3. For the loops written in MIPS assembly above, assume that the register `$t1` is initialized to the value N. How many MIPS instructions are executed?

Solution

1. The equivalent C code is shown as follows:

```
int t1 = 10, s2 = 0;
for (; t1 > 0; --t1) {
    s2 += 2;
}
```

After running the C code, `s2 = 20`. So the final value of `$s2` is 20.

2. By modifying the C code slightly, we can obtain the revised version:

```
int i = 10, B;
for (; i > 0; --i) {
    B += 2;
}
```

3. $5N+2$, since in each iteration of the loop, there are 5 MIPS instructions executed and when the condition does not hold, there still will be 2 instructions executed (i.e. `slt $t2, $0, $t1` and `beq $t2, $0, DONE`).

Exercise 2.25 Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of `a`, `b`, `i`, and `j` are in registers `$s0`, `$s1`, `$t0`, and `$t1`, respectively. Also, assume that register `$s2` holds the base address of the array `D`.

```
for(i = 0; i < a; i++)
    for(j = 0; j < b; j++)
        D[4 * j] = i + j;
```

Solution The equivalent MIPS instructions are given as follows:

```
        li      $t0, 0              # Loop for i, initialize i = 0
LOOP_I: bge     $t0, $s0, END_I     # Exit loop i if i >= a
        li      $t1, 0              # j = 0
LOOP_J: bge     $t1, $s1, END_J     # Exit loop j if j >= b
        sll     $t2, $t1, 2         # $t2 = j * 4
        add     $t2, $t2, $s2       # $t2 = D[$t2]
        add     $t3, $t0, $t1       # $t3 = i + j
        sw      $t3, 0($t2)         # D[$t2] = $t3
        addi    $t1, $t1, 1         # j++
        j       LOOP_J
END_J:  addi    $t0, $t0, 1         # i++
        j       LOOP_I
END_I:
```

Exercise 2.27 Translate the following loop into C. Assume that the C-level integer `i` is held in register `$t1`, `$s2` holds the C-level integer called `result`, and `$s0` holds the base address of the integer `MemArray`.

```
        addi    $t1, $0, 0
LOOP:   lw      $s1, 0($s0)
        add     $s2, $s2, $s1
        addi    $s0, $s0, 4
        addi    $t1, $t1, 1
        slti    $t2, $t1, 100
        bne     $t2, $s0, LOOP
```

Solution

```
for(i = 0; i < 100; ++i) {
    result += MemArray[i];
}
```

Exercise 2.28 Rewrite the loop from Exercise 2.27 reduce the number of MIPS instructions executed. Hint: Notice that variable `i` is used only for loop control.

Solution

```
        add     $t0, $s0, 400   # $t0 = &MemArray[100]
LOOP:   lw      $s1, 0($s0)     # $s1 = MemArray[i]
        add     $s2, $s2, $s1   # result += $s1
        addi    $s0, $s0, 4     # $s0 ++
        bne     $s0, $t0, LOOP  # if($s0 != $t0) goto LOOP
```

Exercise 2.39 Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

1. Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, and the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

2. Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

Solution 1. No, it is slower than before.

Former ISA: Takes 3800 million cycle time

$$\begin{aligned} n_{\text{cycles}} &= n_{\text{arithmetic}} \times 1 + n_{\text{load/store}} \times 10 + n_{\text{branch}} \times 3 \\ &= 500\text{m} \times 1 + 300\text{m} \times 10 + 100\text{m} \times 3 \\ &= 3800 \text{ million cycles} \\ t_{\text{run}} &= t_{\text{cycle}} \times n_{\text{cycles}} \\ &= 3800\text{m} \times t_{\text{cycle}} \end{aligned}$$

New ISA: Takes 4042.5 million cycle time

$$\begin{aligned} n_{\text{cycles}} &= 75\% \times n_{\text{arithmetic}} \times 1 + n_{\text{load/store}} \times 10 + n_{\text{branch}} \times 3 \\ &= 375\text{m} \times 1 + 300\text{m} \times 10 + 100\text{m} \times 3 \\ &= 3675 \text{ million cycles} \\ t_{\text{run}} &= t_{\text{cycle}} \times n_{\text{cycles}} \\ &= 110\% \times t_{\text{cycle}} \times 3675\text{m} \\ &= 4042.5\text{m} \times t_{\text{cycle}} \end{aligned}$$

2. If the performance of arithmetic instructions doubles, then their CPI would be $\frac{1}{2}$ and the program given before will run in $500\text{m} \times \frac{1}{2} + 300\text{m} \times 10 + 100\text{m} \times 3 = 3550$ million cycles, which means the speedup factor $= \frac{3800}{3550} \approx 1.07$.

   As for the 10 times of performance increase, the speedup factor can be calculated similarly: $500\text{m} \times \frac{1}{10} + 300\text{m} \times 10 + 100\text{m} \times 3 = 3350$ million cycles, which means the speedup factor $= \frac{3800}{3350} \approx 1.13$.

Exercise 2.40 Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

1. Given the instruction mix and the assumption that an arithmetic instruction requires 2 cycles, a load/store instruction takes 6 cycles, and a branch instruction takes 3 cycles, find the average CPI.
2. For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?
3. For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

Solution 1. $\text{CPI} = 70\% \times 2 + 10\% \times 6 + 20\% \times 3 = 2.6$.
2. Considering that the average CPI before improvement is 2.6, a 25% overall improvement will result in an average CPI of $75\% \times 2.6 = 1.95$. Let x denote the CPI of the arithmetic instruction, then by solving the equation $70\% \times x + 10\% \times 6 + 20\% \times 3 = 1.95$, we can obtain the improved CPI is $x = 1.07$.
3. Considering that the average CPI before improvement is 2.6, a 50% overall improvement will result in an average CPI of $50\% \times 2.6 = 1.3$. Let x denote the CPI of the arithmetic instruction, then by solving the equation $70\% \times x + 10\% \times 6 + 20\% \times 3 = 1.3$, we can obtain the improved CPI is $x = 0.14$.
