CS1216 - Monsoon 2022 - Homework 2

Jivansh Sharma
UG 24 1020211193

30/9/2022

Collaborators: None

1. Annotate the following MIPS instructions to indicate source and destination registers. Also, describe in words (one sentence or less) what the instruction is trying to do. These are all individual instructions and not a part of a sequence

```
1. addi $t0, $t2, 10
```

$t0 = $t2 + 10. Source is register $t2 and the immediate 10. Destination is $t0.

```
2. sw $s2, 16($s5)
```

Memory[$s5 + 16] = $s2. Source is $s2. Destination is register $s5 with the offset of 16.

```
3. lw $s3, 4($gp)
```

$s3 = Memory[$gp + 4] . Source is global pointer $gp with the offset of 4. Destination is $s3.

```
4. jr $ra
```

jr returns control to the caller. It copies the contents of $ra into the system. Source would be $ra, and Destination would be the system itself.

```
5. bne $t3, $s0, Else
```

If $t3 $\neq$ $s0 go to Else.
bne is a conditional which checks if the values at $t3 and $s0 are not equal. If this boolean check passes, it redirects the program to the Else procedure. Sources can be considered as the boolean computation of $t3 and $s0. The destination would be Else procedure.

2. Consider the following fragment of C code. This code declares two global variables: an integer variable i, and an integer array array[], which has 40 elements. Then, the code loops over the entire array, doing a simple calculation as shown below. Convert this code to use MIPS assembly instructions that we have covered in class. Assume that you would have been provided a value in the gp register, which would specify the size of the text region of the program. Assume that i is laid out in memory, before array[]. Provide comments for each line of code so that it is clear what is happening, is actually intended.

```
    int i, array[40];
    main( ) {
        for (i=0; i<40; i++)
            array[i] = array[i] + i;
    }
```

```
# assume 0($gp) = i
# assume 4($gp) = array[0]

main:
    addi $s0, $zero, 40

for_loop:
    bge $gp, $s0, end       # if i >= 40 go to end

    mul $t1, $gp, 4         # (i * 4) = running offset calculation
    add $t2, $t1, 4($gp)    # array starts at offset 4 from gp + we add the running offset
    lw $t3, ($t2)           # load value at 4GP + running offset into t3

    add $t4, $t3, $gp       # t4 | new value of array[i] = t3 + i

    sw $t4, ($t2)           # at array[t2] store [t3 + i]

    addi $gp, $gp, 1     # i = i + 1
    j for_loop           # restart loop
end:
    jr $ra               # kill process
```

3. There are several different instructions that can be used to change the control flow in a MIPS program. Selecting from j, jal, jr, beq and bne which instruction has the greatest range? (In other words, which instruction can be used to go the furthest away relative to where the instruction is located?) Why is it true?

jr has the largest range because it has 32-Bits allocated only to the desitination address.
Whereas, j and jal each has 26-Bits.
And, beq and bne have 16-Bits.
Referred to the Glossary here: https://cgi.cse.unsw.edu.au/ cs1521/22T2/resources/mips-guide.html

4. Convert the following fragment of C code into MIPS assembly. Assume the only instructions available for you to use are add, addi, sub, lw, sw, sll, srl, j, beq and bne. Also assume that variables i and j are stored in $s2 and $s5, respectively.

```
    if ( i < j )
        j = j + (16 * i);
    else
        i = i - (64 * j);

main:
    bge $s2, $s5, else # if s2 >= s5 go to else

    # j = j + (16 * i)
    sll $t0, $s2, 4 # t0 = (16 * s2) and 4 << 2 == 16
    add $s5, $s5, $t0 # s5 = s5 + t0

    j exit
else:
    # i = i - (64 * j)
    sll $t0, $s5, 6 # t0 = (64 * s5) and 6 << 2 == 64
    sub $s2, $s2, $t0 # s5 = s5 + t0
exit:
    jr $ra
```

5. Assume that we would like to expand the MIPS register file to 128 registers and expand the instruction set to contain four times as many instructions than we have currently.

MIPS, by default has the register file of 32 registers. A 5-bit number is used to identify a register ($2^5 = 32$). For 128 registers, we would need $log_2 128 = 2^7 = 7$ Bits

(a) How would this affect the size of each of the bit fields in the R-type instructions?

We know the original 32 Bit entities are broken down into multiple fields.

```
OP CODE (Operation Code) -> 6 Bits
RS (Source Register) -> 5 Bits
RT (Source Code) -> 5 Bits
RD (Desitination Register) -> 5 Bits
SHAMT (Shift Amount) -> 5 Bits
FUNCT (Function) -> 6 Bits
```

As we increase the bit size, it is clear that each RS, RT and RD expand to 7 Bits.
Also, the number of instructions go from 50 to 4 times ie 200. Therefore the closest exponent of 2 would be 256, ie 2^8. Therefore OPCODE goes from 6 Bits to 8 Bits.

So, the final bit fields would look like this:

```
OP CODE (Operation Code) -> 8 Bits
RS (Source Register) -> 7 Bits
RT (Source Code) -> 7 Bits
RD (Desitination Register) -> 7 Bits
SHAMT (Shift Amount) -> 5 Bits
FUNCT (Function) -> 6 Bits
```

(b) How would this affect the size of each of the bit fields in the I-type instructions?

We know the original 32 Bit entities are broken down into multiple fields.

```
OP CODE (Operation Code) -> 6 Bits
RS (Source Register) -> 5 Bits
RT (Source Code) -> 5 Bits
CONSTANT -> 16 Bits
```

As we increase the bit size, it is clear that each RS and RT expand to 7 Bits.
Also, the number of instructions go from 50 to 4 times ie 200. Therefore the closest exponent of 2 would be 256, ie 2^8. Therefore OPCODE goes from 6 Bits to 8 Bits.

So, the final bit fields would look like this:

```
OP CODE (Operation Code) -> 8 Bits
RS (Source Register) -> 7 Bits
RT (Source Code) -> 7 Bits
CONSTANT -> 16 Bits
```

6. In MIPS assembly language, there exists an instruction called seq. It compares the contents of two source registers and sets the destination register to 1 if they are equal, else sets the destination register to 0. Write a short sequence of any valid MIPS assembly instructions that we have covered till now (except seq itself) to compare the contents of the source registers t1 and t2 and set the destination register v0 to 1 if they are equal, else 0.

```
Eg. seq $v0, $t1, $t2

main:
    addi $t1, $zero, 1          # t1 = 1
    addi $t2, $zero, 1          # t2 = 1

    bne $t1, $t2, else          # if t1 != t2, go to else
    addi $v0, $zero, 1          # Set v0 to 1 if t1 == t2
    j end                       # jump to end

else:
    addi $v0, $zero, 0          # Set v0 to 0 if t1 != t2

end:
    jr $ra                      # kill process
```

7. Observe the following piece of MIPS assembly code. Boil this down to one line of C. Additionally, put in the comments for each line to show what is being done in each line. You will have to look up the definition of the mult instruction. It would be helpful to think about the problem in terms of two C variables that are being used.

```
lw     $t0, 4($gp)          # t0 = gp[1]
add    $t1, $t0, $zero      # t1 = t0 + 0 = t0
addi   $t1, $t1, 3          # t1 += 3
mul    $t1, $t1, $t0        # t1 = t1*t0
sw     $t1, 0($gp)          # gp[0] = t1
```

Considering this as a C program and breaking it down,

```
int *gp[];
int main (void) {
    int t0 = gp[1];
    int t1 = t0;

    t1 += 3;
    t1 *= t0; // t1 = (t0 + 3) * t0

    gp[0] = t1;
}
```

So, we can truncate/prune this into one line and say

```
    gp[0] = (gp[1] + 3) * gp[1];
```

8. Use the register and memory values in the tables below for the next questions. Assume a 32-bit architecture. Also assume that each of the following questions (parts a, b and c) start from the table values.

(a) Provide the values of registers s1 and s3 after this instruction is executed:

```
lw $s3, 4($s1)
```

The value at $s1 is 16. In memory, the address 16 (0x000010) is also assigned to a value, namely 26. When an offset of 4 is applied, the memory address becomes 16 + 4 == 20. And, the value at memory address 20 is 28.

Therefore, $s3 = 28 and $s1 = 16

(b) Provide the values of registers s2 and s3 after this instruction is executed:

```
addi $s2, $s3, 16
```

The value at $s2 is 15, and that at $s3 is 20. However, the value at s2 will be replaced by the sum of 16 and 20.

Therefore, $s3 = 20 and $s2 = 36

9. Consider a program that declares global integer variables a, b, c[20]. These variables are allocated starting at a base address of decimal 1000. All these variables have been initialized to zero. The base address 1000 has been placed in gp. The program executes the following assembly instructions:

```
lw   $s1, 0($gp)
lw   $s2, 4($gp)
addi $s1, $s1, 5
addi $s2, $s2, 9
sw   $s1, 8($gp)
sw   $s2, 12($gp)
add  $s2, $s2, $s1
sw   $s2, 16($gp)
```

```
What we know,

gp = 1000
a = gp[0] = 0
b = gp[1] = 4($gp) = 0
c = gp[2] = 8($gp) = 0 ... continues till c[19]

Breaking down instructions,

s1 = gp[0] == 0
s2 = gp[1] == 0
s1 = s1 + 5 = gp[0] + 5 == 5
s2 = s2 + 9 = gp[1] + 9 == 9
c[1] = gp[3] = s2 = gp[1] + 9 == 9
c[0] = gp[2] = s1 = gp[0] + 5 == 5
s2 = s2 + s1 = gp[0] + 5 + gp[1] + 9 = gp[0] + gp[1] + 14 == 14
c[2] = gp[4] = gp[0] + gp[1] + 14 == 14
```

(a) What are the memory addresses of variables a, b, c[0], c[1], and c[2]?

```
a = 1000
b = 1004
c[0] = 1008
c[1] = 1012
c[2] = 1016
```

(b) What are the values of variables a, b, c[0], c[1], and c[2] at the end of the program?

```
a = 0
b = 0
```

```
c[0] = 5
c[1] = 9
c[2] = 14
```

10. Consider the following code fragment in MIPS assembly. In one sentence or less (per line) describe what is happening in each of the following instructions.

1. `addi $sp, $sp,  -12`

-12 is being added to the value at $sp and then stored again at $sp. This is actually the command in MIPS, to preserve space for `3 Words` in the stack pointer.

2. `sw   $t1, 8($sp)`

store $t1 at address $sp + 8. This is equivalent of saying RAM[sp + 8] = RAM[2] = $t1

3. `sw   $t0, 4($sp)`

store $t0 at address $sp + 4. This is equivalent of saying RAM[sp + 4] = RAM[1] = $t0

4. `sw   $s0, 0($sp)`

store $s0 at address $sp. This is equivalent of saying RAM[sp] = RAM[0] = $s0