15.11.2022

CENG311 Midterm Exam Answer Key- Fall 2022

Q.1 (20 Points) Assume we have an integer (4-byte) array A with base address stored in $s1. Consider the MIPS code given below and answer the following questions.

```
1:              add $s2, $zero, $zero
2:              addi $s0, $zero, 5
3:    L1:       beq $s0, $zero, exit
4:              lw $t0, 8($s1)
5:              add $s2, $s2, $t0
6:              addi $s0,$s0, -1
7:              j L1
8:    exit:
```

a) What does the code perform (you can write a sentence)? How many instructions will be executed for the given code? How many of them are memory operations?

Adds A[2] five times (multiplies by 5) and stores the result in $s2.

28 (+ exit) instructions
5 memory operations

b) Modify the code to reduce the number of memory operations by keeping the same functionality. How many memory operations are performed in the modified code?

Exchange 3 and 4 (lw out of the loop)
(Change with multiplication is also OK)

1 memory operation

Q.2 (40 Points)

```
1:              sll $t0, $s1, 2
2:              sll $t1, $s2, 2
3:              add $t0, $t0, $a0
4:              add $t1, $t1, $a1
5:              bne $s1, $s2, else
6:      if:     lw $t2, 0($t1)
7:              j end
8:      else:   lw $t2, 4($t1)
9:      end:    sw $t2, 0($t0)
```

Assume that we have the following memory and register file content at the beginning of the MIPS program given above.

MEMORY

Address	Value
0	5
4	10
8	12
12	13
16	15
20	17
24	9
28	10
32	12

REGISTER FILE

Register	Value
$s1	3
$s2	2
$t0	4
$t1	8
$t2	12
$a0	4
$a1	20

a) Show all executed instructions in order. After each instruction completion, show the modified register and memory location (if any) by specifying the new value. Finally, show the content of **all** given memory locations and registers.

Example:

```
sll $t0, $s1, 2: $t0=new value
sll $t1, $s2, 2: $t1=new value
…
```

Execution order: 1-2-3-4-5-8-9

```
1:              sll $t0, $s1, 2          $t0=3x4=12
2:              sll $t1, $s2, 2          $t1=2x4=8
3:              add $t0, $t0, $a0        $t0=12+4=16
4:              add $t1, $t1, $a1        $t1=8+20=28
5:              bne $s1, $s2, else       no register/memory update
6:      if:     lw $t2, 0($t1)           not executed
7:              j end                    not executed
8:      else:   lw $t2, 4($t1)           $t2=memory [4+28] => $t2=12
9:      end:    sw $t2, 0($t0)           memory [0+16]=12 => memory [16]=12
```

NEW CONTENT:

MEMORY

Address	Value
0	5
4	10
8	12
12	13
16	**12**
20	17
24	9
28	10
32	12

REGISTER FILE

Register	Value
$s1	3
$s2	2
$t0	**16**
$t1	**28**
$t2	12
$a0	4
$a1	20

b) Assume we have the following CPI values for each instruction type and run the program in a computer with 2 GHz frequency. What is the execution time of the program?

Instruction class	CPI
ALU	6
Memory operation	8
Branch/Jump	5

4 ALU (sll, sll, add, add) + 1 Branch (bne) + 2 Memory instructions (lw, sw)

Execution time = CPU clock cycles / Clock rate
= 6 x 4 + 8 x 2 + 5 x 1 / 2x10-9 = 45/ 2x10-9

c) Convert the last two executed instructions into binary representation.

lw $t2, 4($t1) : 1000 1101 0010 1010 0000 0000 0000 0100

sw $t2, 0($t0) : 1010 1101 0000 1010 0000 0000 0000 0000

Q.3 (40 Points) Consider the C code below and its MIPS translation given partially:

C code:

```
int evens(int A[], int size){
        int count = 0;
        for(int i = 0; i < size; i++){
                if(isEven(A[i])) count++;
        }
        return count;
}
```

MIPS code:

```
1:              jal evens

2:     evens:   addi $sp, $sp, __-16____     # stack operations
3:              sw __$ra_____, 12($sp)
4:              sw __$s0_____, 8($sp)
5:              sw __$s1_____, 4($sp)
6:              sw __$s2_____, 0($sp)
7:              add $s0, $zero, $zero         # i
8:              add $s1, $zero, $zero         # count
9:              add $s2, $a0, $zero   # argument
10:     loop:   slt $t0, $s0, ___$a1_____
11:             beq $t0, $zero, exit
12:             sll $t0, $s0, ___2_____
13:             add $t0, $s2, $t0
14:             lw ___$a0_____, 0($t0)
15:             jal ___isEven_____
16:             addi __$s0_____,___$s0_____, ____1_____ #increment i
17:             beq ___$v0_____, ____$zero____, loop
18:             addi __$s1_____, ___$s1_____, ____1_____          #increment count
19:             j loop
20:     exit:   add $v0, $zero, $s1
21:             lw ___$s2_____, 0($sp)        #stack operations
22:             lw ___$s1_____, 4($sp)
23:             lw ___$s0_____, 8($sp)
24:             lw ___$ra_____, 12($sp)
25:             addi $sp, $sp, ___16_____
26:             jr $ra

27:  isEven:    add $v0, $zero, $zero
28:             andi $t0, ___$a0_____, ___1_____
29:             beq $t0, $zero, even
30:             __jr $ra_____
31:   even:     addi $v0, $zero, 1      # return 1 if the number is even
32:             jr $ra
```

a) By considering the corresponding C code and MIPS language conventions, fill in the blanks in the given MIPS code.

Hint: In the binary representation of a number, any number with its least significant bit set to 0 is even.

b) Functions can often be implemented by compilers “in-line.” An in-line function is when the body of the function is copied into the program space, allowing the overhead of the function call to be eliminated. Rewrite the above code by performing *function inlining* for *isEven* function. You need to eliminate all the instructions necessary for function invocation and to move instructions inside the function.