VE370 RC2

HW2 SOLUTION

Q1

Convert the following assembly code fragment into machine code, assuming the memory location of the first instruction is 0x10000400: (15 points)

```
LOOP:   slt   $t2, $zero, $t0
        bne   $t2, $zero, ELSE
        j     DONE
ELSE:   addi  $s2, $s2, 2
        addiu $t0, $t0, 1
        j     LOOP
DONE:   …
```

- 000000 00000 01000 01010 00000 101010
- 000101 01010 00000 00000 00000 000001
- 000010 00000 00000 00000 00100 000110
- 001000 10010 10010 00000 00000 000010
- 001001 01000 01000 00000 00000 000001
- 000010 00000 00000 00000 00100 000000

Target Addressing Example

- Loop code from earlier example
  - Assume Loop at location 80000 (decimal)

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024						

- Used for near branch
  - Forward or backward
  - Target address = new PC + offset × 4
  - New PC = PC+4

Q2

2. Following memory location has address 0x10000000 and content 0x15478933.

	0	1	2	3
0x10000000	15	47	89	33

Write MIPS assembly instructions to load the byte 47 to register $s1, then show the content of $s1 after the operations. (5 points)

```
lui  $t0, 4096
lb   $s1, 1($t0)
```

0x00000047

Load Byte	lb	I	`lw $s1,100($s2)` $s1 = Mem[100+$s2]	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}
Load Byte Unsigned	lbu	I		R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}
Load Halfword	lh	I		R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}
Load Halfword Unsigned	lhu	I	`lui $s1,100` $s1 = 100*2^16	R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}
Load Upper Imm.	lui	I		R[rt]={imm,16'b0}
Load Word	lw	I		R[rt]=M[R[rs]+SignExtImm]
Load Immediate	li	P		R[rd]=immediate
Load Address	la	P		R[rd]=immediate

2.19.1(b)

b.
```
int fib_iter(int a, int b, int n){
    if(n == 0)
        return b;
    else
        return fib_iter(a+b, a, n-1);
}
```

2.19.1 [15] <2.8> Implement the C code in the
the total number of MIPS instructions needed to

b.
```
fib_iter:
        addi $sp, $sp, -16
        sw   $ra, 12($sp)
        sw   $s0, 8($sp)
        sw   $s1, 4($sp)
        sw   $s2, 0($sp)

        add  $s0, $a0, $0
        add  $s1, $a1, $0
        add  $s2, $a2, $0

        add  $v0, $s1, $0,
        bne  $s2, $0, exit

        add  $a0, $s0, $s1
        add  $a1, $s0, $0
        add  $a2, $s2, -1
        jal  fib_iter
exit:
        lw   $s2, 0($sp)
        lw   $s1, 4($sp)
        lw   $s0, 8($sp)
        lw   $ra, 12($sp)
        addi $sp, $sp, 16
        jr   $ra
```

2.19.2(b)

2.19.2 [5] <2.8> 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. Implement an "in-line" version of the C code in the table in MIPS assembly. What is the reduction in the total number of MIPS assembly instructions needed to complete the function? Assume that the C variable n is initialized to 5.

Due to the recursive nature of the code, not possible for the compiler to in-line the function call..

2.19.3(b)

2.19.3 [5] <2.8> For each function call, show the contents of the stack after the function call is made. Assume the stack pointer is originally at addresss 0x7ffffffc, and follow the register conventions as specified in Figure 2.11.

b.

```
after calling function fib_iter:
old $sp =>      0x7ffffffc      ???
                -4              contents of register $ra
                -8              contents of register $s0
                -12             contents of register $s1
$sp =>          -16             contents of register $s2
```

Procedure Calling Convention

- Before procedure starts executing
  1. Allocate memory of frame's size
     - by moving $sp downwords for frame's size
  2. Save registers that should be saved by the procedure in the frame, before they are overwritten
     - $s0-$s7 (if to be used), $fp (if used), $ra (non-leaf procedure),

2.23.1 (a)

2.22.3 [5] <2.5, 2.9> Translate the hexadecimal ASCII values to text.

Exercise 2.23

In this exercise, you will be asked to write a MIPS assembly program that converts strings into the number format as specified in the table.

a.	positive integer decimal strings
b.	two's complement hexadecimal integers

2.23.1 [10] <2.9> Write a program in MIPS assembly language to convert an ASCII number string with the conditions listed in the table above, to an integer. Your program should expect register `$a0` to hold the address of a null-terminated string containing some combination of the digits 0 through 9. Your program should compute the integer value equivalent to this string of digits, then place the number in register `$v0`. If a nondigit character appears anywhere in the string, your program should stop with the value –1 in register `$v0`. For example, if register `$a0` points to a sequence of three bytes 50_{ten}, 52_{ten}, 0_{ten} (the null-terminated string "24"), then when the program stops, register `$v0` should contain the value 24_{ten}.

2.23.1(a)

a.

```
MAIN:   addi $sp, $sp, -4
        sw   $ra, ($sp)
        add  $t6, $0, 0x30  # '0'
        add  $t7, $0, 0x39  # '9'
        add  $s0, $0, $0
        add  $t0, $a0, $0

LOOP:   lb   $t1, ($t0)
        slt  $t2, $t1, $t6
        bne  $t2, $0, DONE
        slt  $t2, $t7, $t1
        bne  $t2, $0, DONE
        sub  $t1, $t1, $t6

        beq  $s0, $0, FIRST
        mul  $s0, $s0, 10
FIRST:  add  $s0, $s0, $t1
        addi $t0, $t0, 1
        j LOOP

DONE:   add  $v0, $s0, $0
        lw   $ra, ($sp)
        addi $sp, $sp, 4
        jr   $ra
```

2.24.3 (a)

Assume that the register $t1 contains the address 0x1000 0000 and the register $t2 contains the address 0x1000 0010.

a.	lb $t0, 0($t1) sw $t0, 0($t2)
b.	lb $t0, 0($t1) sb $t0, 0($t2)

2.24.3 [5] <2.9> Assume that the data (in hexadecimal) at address 0x1000 0000 is:

1000 0000	11	00	00	FF

What value is stored at the address pointed to by register $t2? Assume that the memory location pointed to $t2 is initialized to 0x5555 5555.

0x00000011

2.26.1 (a)

Exercise 2.26

For this exercise, you will explore the range of branch and jump instructions in MIPS. For the following problems, use the hexadecimal data in the table below.

a.	0x00001000
b.	0xFFFC0000

2.26.1 [10] <2.6, 2.10> If the PC is at address 0x00000000, how many branch (no jump instructions) do you need to get to the address in the table above?

$0x00001000 < 2^{15} * 4$

1 branch is enough!

I-format Example 3

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

```
beq $s0, $t0, LOOP
```

op	$s0	$t0	Relative Address
4	16	8	Relative Address
000100	10000	01000	Relative Address

LOOP = PC+4+Relative Address*4

2.26.2(b) & 2.26.3(b)

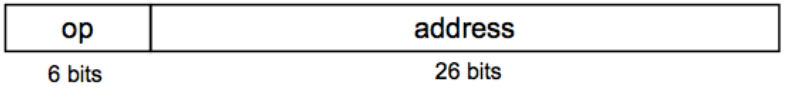
2.26.2 [10] <2.6, 2.10> If the PC is at address 0x00000000, how many jump instructions (no jump register instructions or branch instructions) are required to get to the target address in the table above?

2.26.3 [10] <2.6, 2.10> In order to reduce the size of MIPS programs, MIPS designers have decided to cut the immediate field of I-type instructions from 16 bits to 8 bits. If the PC is at address 0x0000000, how many branch instructions are needed to set the PC to the address in the table above?

26.2(b): can't be done with PC starting with 0x0...

26.3(b): 0xFFFC0000
(1111 1111 1111 1100 0000 0000 0000 0000)
(0000 0000 0000 0011 1111 1111 1111 1111 +1)
$= 2^{18}$
$2^{18} / 2^{7} / 4 = 512$ branches

J-format



- Encode full address in instruction
- (Pseudo) Direct jump
  - Target address = PC[31:28] : (address × 4)

2.26.4

For the following problems, you will be using making modifications to the MIPS instruction set architecture.

a.	8 registers
b.	10 bit immediate/address field

2.26.4 [10] <2.6, 2.10> If the instruction set of the MIPS processor is modified, the instruction format must also be changed. For each of the suggested changes above, what is the impact on the range of addresses in a beq instruction? Assume that all instructions remain 32 bits long and any changes made to the instruction format of I-type instructions only increase/decrease the immediate field of the beq instruction.

A. The Imm filed increases from 16-bit to 20-bit --> 16 times larger in branch.
B. 64 times smaller in branch.

2.27.1 & 2

In the following problems, you will be using exploring different addressing modes in the MIPS instruction set architecture. These different addressing modes are listed in the table below.

a.	Register Addressing
b.	PC-relative Addressing

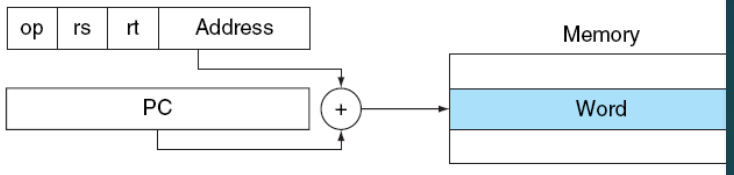
2.27.1 [5] <2.10> In the table above are different addressing modes of the MIPS instruction set. Give an example MIPS instructions that shows the MIPS addressing mode.

2.27.2 [5] <2.10> For the instructions in 2.27.1, what is the instruction format type used for the given instruction?

Register Addressing

- All or some operands provided by registe IDs directly
- Used in R-type and I-type instructions
- Example:
  - add $t0, $s0, $s1
  - beq $s0, $s1, FUNCTION

PC-relative Addressing



- Operand relative to PC
- Used for near branch
  - Forward or backward
  - Target address = new PC + offset × 4
  - New PC = PC+4
- Example:
  - beq $s0, $s1, LESS (I-type)