1. “Short or Quick Answer” Questions:

   a. (4%) In MIPS, what are the least 2 significant bits of a word address? Answer: $(00)_2$

   b. (4%) Name one MIPS assembly instruction that has destination last?

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`	Three operands; data in registers
	add immediate	`addi $s1, $s2, 100`	`$s1 = $s2 + 100`	Used to add constants
Data transfer	load word	`lw $s1, 100($s2)`	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	`sw $s1, 100($s2)`	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte	`lb $s1, 100($s2)`	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	`sb $s1, 100($s2)`	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	`lui $s1, 100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	`beq $s1, $s2, 25`	if (`$s1 == $s2`) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1, $s2, 25`	if (`$s1 != $s2`) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1, $s2, $s3`	if (`$s2 < $s3`) `$s1` = 1; else `$s1` = 0	Compare less than; for beq, bne
	set less than immediate	`slti $s1, $s2, 100`	if (`$s2 < 100`) `$s1` = 1; else `$s1` = 0	Compare less than constant
Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to `$ra`	For switch, procedure return
	jump and link	`jal 2500`	`$ra` = PC + 4; go to 10000	For procedure call

   Answer: store word.

   c. (4%) Do we need `subi` in MIPS as `addi` supports negative constant? Justify your answer. Answer: We do not need `subi` because we can use `addi` to add a negative number to complete the target function of `subi`.

   d. (4%) What makes a computer different from a calculator? Answer: The ability to make decisions.

   e. (4%) What is the corresponding real number for a 4-byte floating point number $(01000001\ 01000000\ 00000000\ 00000000)_{binary}$? Answer: $(1.1)_{binary}\times2^3 = (1100)_{binary} = (12)_{ten}$.

2. (16%) Give an example for each Design Principle in terms of MIPS. (i) Simplicity favors regularity. (ii) Smaller is faster. (iii) Make the common case fast. (iv) Good design demands compromise.

Answers: (i) Simplicity favors regularity. In MIPS, all instructions are 4-byte long. Another example is that in MIPS, all (R-type) instructions have 3 operands. (ii) Smaller is faster. We need to keep the balance between the register number and speed. A very large number of registers would increase the clock cycle time simply because it takes electrical signals longer when they must travel further. (iii) Make the common case fast. Operations on small constants occur very often in programs. Hence, adding specific instructions for arithmetic operations about small constants, such as `addi`, will benefit the system performance. (iv) Good design demands compromise. To have fixed-length instructions but different formats (R-type, I-type, J-type) is a compromise against Design Principle I.

3. (Modified from Example on page 98) The while loop, `while(save[i]==k)i+=1;`, was

compiled into MIPS assembler code below.

```
    Loop:  sll $t1,$s3,2        # Temp reg $t1=4*i
           add $t1,$t1,$s6      # $t1=address of save[i]
           lw $t0,0($t1)        # Temp reg $t0=save[i]
           bne $t0,$s5,Exit     # go to Exit if save[i]≠k
           addi $s3,$s3,1       # i=i+1
           j Loop               # go to Loop
    Exit:
```

(a) (8%) If we assume we place the loop starting at location z_{ten}, represent x_{ten} and y_{ten} as a function of z.

Address	6-bit op	5-bit	5-bit	5-bit	5-bit	6-bit
$(z)_{ten}$	0_{ten}	0_{ten}	19_{ten}	9_{ten}	4_{ten}	0_{ten}
$(z+4)_{ten}$	0_{ten}	9_{ten}	22_{ten}	9_{ten}	0_{ten}	32_{ten}
$(z+8)_{ten}$	35_{ten}	9_{ten}	8_{ten}	0		
$(z+12)_{ten}$	5_{ten}	8_{ten}	21_{ten}	x_{ten}		
$(z+16)_{ten}$	8_{ten}	19_{ten}	19_{ten}	1_{ten}		
$(z+20)_{ten}$	2_{ten}	y_{ten}				
$(z+24)_{ten}$	…					

(b) (6%) Determine all the possible values of y according to your formula in (a) and the 26-bit field limitation of y. (Hint: Can y be negative? Can y be zero? You may let $z = i \times 16^7 + 4 \times j - 24$, where i and j are both non-negative integers and $j < 16^7/4$, and check all the feasible i and j.)

Answers: (a) $(z+16) + x \times 4 = (z+24)$ and $\lfloor (z+24)/16^7 \rfloor \times 16^7 + y \times 4 = z$. Hence, $x = 2$ and $y = (z - \lfloor (z+24)/16^7 \rfloor \times 16^7)/4$.

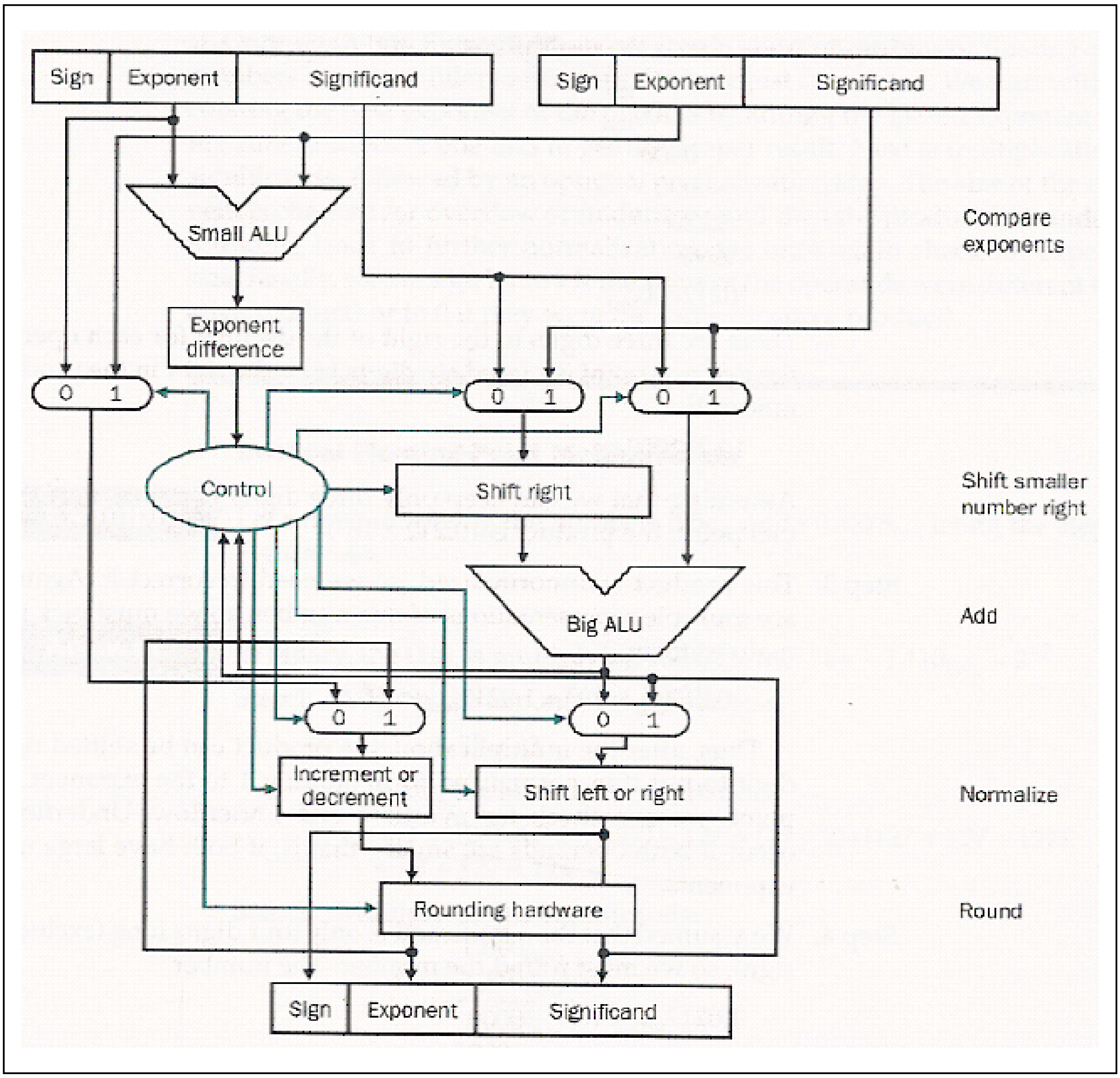
(b) Let $z = i \times 16^7 + 4 \times j - 24$, where i and j are both non-negative integers and $j < 16^7/4$. Then according to the formula in (a), $y = (z - \lfloor (z+24)/16^7 \rfloor \times 16^7)/4 = ((i \times 16^7 + 4 \times j - 24) - (i \times 16^7))/4 = j - 6$. Hence, $(j - 6)$ can be any integers greater than and equal to -6, and strictly less than $16^7/4 - 6 = 67108858$. (This part weights 4%.)

On the other hand, the 2's complement range for a 26-bit field like y is from $-2^{25}(= -33554432)$ to $2^{25}-1$ (= 33554431). Consequently, $-6 \le y \le \min(67108857, 33554431) = 33554431$. (This part weights 2%.)

4. (a) (10%) Describe the floating-point addition algorithm in block diagram. (Hint: Before, addition, we have to compare the exponent of the two numbers. After this, we shift one of the numbers.)

(b) (6%) Why we choose to shift the smaller number to the right, not to shift the larger number to the left?

Answers: (a)



(b) to make the exponent the same; shift the smaller number to the right will not change the format of the fraction and make it easier to add together. This way will save hardware. Shift the larger to the left, will generate extra requirement to handle the bits that are on the left side of the floating point.

5. (16%) Do it step by step like a computer doing the division, where Divisor = $(0010)_2$ and dividend = $(0000\ 0111)_2$.

Answers:

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⇒ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⇒ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

6. (18%) Describe the difference among the three instructions at location `0x98765432`.

(i) `0x98765432` `j 12345678`

(ii) `0x98765432` `jr $ra`

(iii) `0x98765432` `jal 12345678`

Specially specify what is in the program counter and what is in the return register after the execution of each instruction.

Answers: `j 0x01234567`, `PC=12345678`, `$ra` not affected.

`jr $ra`, `PC=$ra`, `$ra` not affected

`jal 0x01234567`, `PC=12345678`, `$ra=0x98765436`