Computer Organization
Midterm Exam

Instructor: Dr. Ing-Jer Huang

1. Consider a computer running programs with CPU times shown in the following table.

FP Instr.	INT Instr.	L/S Instr.	Branch Instr.	Total Time
35 S	85 S	50 S	30 S	200 S

(1) (5%) By how much is the **total time reduced** if the time for FP operations is reduced by 20%?
(2) (5%) By how much is the time for **INT operations reduced** it the total time is reduced by 20%?

2. 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 array A and B are in registers $s6 and $s7, respectively.

(1) (5%) For the C statement below, what is the corresponding MIPS assembly code?
(**Please refer to the final page** for the MIPS assembly language revealed in chapter 2)

```
C statement: f = g + h + B[4]
```

(2) (5%) For the C statement above, how many different registers are needed to carry out the C statement?

3. For the following problems, the table holds various binary values for register $t0 and $t1.

$t0 = 1010 1101 0001 0000 0000 0000 0000 0000$_{two}$
$t1 = 0011 1111 1111 1000 0000 0000 0000 0000$_{two}$

(1) (5%) What is the value of **$t2** after the following instructions?

```
        slt   $t2, $t0, $t1
        beq   $t2, $zero, ELSE
        j     DONE
ELSE:   addi  $t2, $zero,2
DONE:
```

(2) (5%) What is the value of **$t2** after the following instructions?

```
        sll   $t0, $t0, 2
        slt   $t2, $t0, $zero
```

4. (5%) Show a truth table for a multiplexor (inputs *A*, *B*, and *S*; output *C*), using don't cares to simplify the table where possible.

5. (5%) Prove that the NAND gate is universal gate by showing how to build the AND, OR, and NOT functions using a two-input NAND gate.

6. The following table shows results for SPEC2006 benchmark programs running on an AMD Bracelona.

	Name	Intr. Count x10^9	Execution time (seconds)	Reference time (seconds)
a	perl	2118	500	9770
b	mcf	336	1200	9120

(1) (5%) Find the CPI if the clock cycle time is 0.333 ns.

(2) (5%) Find the SPEC ration.

(3) (5%) For these two benchmarks, find the geometric mean.

7. Compile the following C statement. (**Please refer to the final page** for the MIPS assembly language revealed in chapter 2)

(1) (5%) Assume variable f, g, h, i, and j are assigned to the registers $s0, $s1, $s2, $s3, and $4 respectively. For the C statement below, what is the corresponding MIPS assembly code?

```
C statement: f = (g + h) – (i + j)
```

(2) (5%) Assume variable h is assigned to the registers $s2 and the base address of the array A is in $s3. For the C statement below, what is the corresponding MIPS assembly code?

```
C statement: A[12] = h + A[8]
```

8. (5%) Show how the data in the table would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0

```
0x12345678
```

9. (10%) Consider an integer array a [5], which contains five 32-bits integer. What is the value of X, Y, Z, and W.

	Value	Mem Address
a[0]:	300	100 ~ 103
a[1]:	400	104 ~ 107
a[2]:	500	108 ~ 111
a[3]:	600	112 ~ 115
a[4]:	700	116 ~ 119

```
(1) X = a;
(2) Y = *a;
(3) Z = *(a+2);
(4) W = *a+2;
```

10. (10%) Let's look in more detail at multiplication. We will use the numbers in the following table.

A	B
101000_{two}	010011_{two}

Using the hardware described in Figure 1 to calculate the product of two unsigned 6-bits binary numbers A and B. Complete the contents of each register on each step list on the following table.

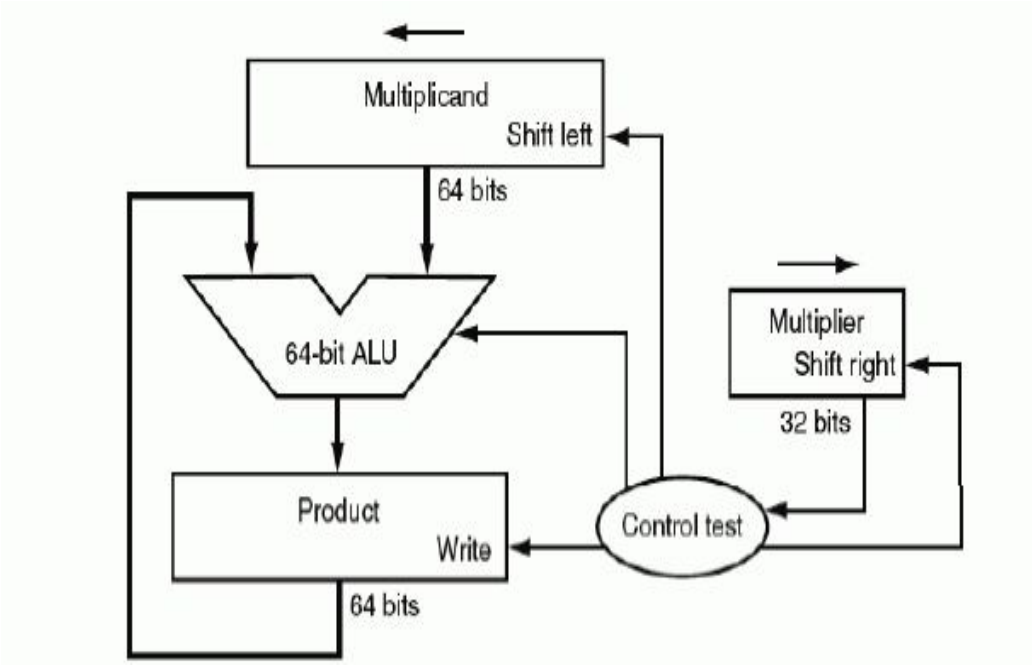

Figure 1 First version of the multiplication hardware

Step	Action	Multiplier	Multiplicand	Product
0	Initial values	010 011	000 000 101 000	000 000 000 000
1	Prod = Prod + Mcand	010 011	000 000 101 000	000 000 101 000
	Lshift Mcand	010 011	000 001 010 000	000 000 101 000
	Rshift Mplier	001 001	000 001 010 000	000 000 101 000
2	Prod = Prod + Mcand	001 001	000 001 010 000	000 001 111 000
	Lshift Mcand	001 001	000 010 100 000	000 001 111 000
	Rshift Mplier	000 100	000 010 100 000	000 001 111 000
3	LSB = 0, no operation	000 100	000 010 100 000	**(1)** _____
	Lshift Mcand	000 100	000 101 000 000	**(2)** _____
	Rshift Mplier	000 010	000 101 000 000	**(3)** _____
4	LSB = 0, no operation	000 010	000 101 000 000	**(4)** _____
	Lshift Mcand	000 010	001 010 000 000	000 001 111 000
	Rshift Mplier	000 001	001 010 000 000	000 001 111 000
5	Prod = Prod + Mcand	000 001	001 010 000 000	**(5)** _____
	Lshift Mcand	000 001	010 100 000 000	**(6)** _____
	Rshift Mplier	000 000	010 100 000 000	**(7)** _____
6	LSB = 0, no operation	000 000	010 100 000 000	**(8)** _____
	Lshift Mcand	000 000	101 000 000 000	**(9)** _____
	Rshift Mplier	000 000	101 000 000 000	**(10)** _____

11. The following table shows bit patterns expressed in hexadecimal notation.

Bit pattern: $0xAFBF0000_{sixteen}$

What decimal number does the bit pattern represent.

(1) (5%) If it is a two's-complement integer?

(2) (5%) If it is a floating-point number? Use the IEEE 754 standard.

12. (10%) **Please refer to the 4-bit ALU as shown in Figure 2.** The ALU supported set on less than (`slt`) using just the sign bit of the adder. Let's try a set on less than operation using the value -7_{ten} and 6_{ten}. To make it simpler to follow the example, let's limit the binary representations to 4 bits: 1001_{two} and 0110_{two}

$$1001_{two} - 0110_{two} = 1001_{two} + 1010_{two} = 0011_{two}$$

This result would suggest that -7 > 6, which is clearly wrong. Hence we must factor in overflow in the decision. Modify the Most Significant Bit (MSB) ALU list on the right side to handle `slt` correctly. Make your changes on this paper directly to save time. (Hint: if overflow not occurs, when sign bit equals to '1', it implies a < b; if overflow occurs, when sign bit equal to '0', it implies a < b. Therefore, you will add a **logic gate** on the signal **Set and Overflow**, and then generate a **new Set** signal.)

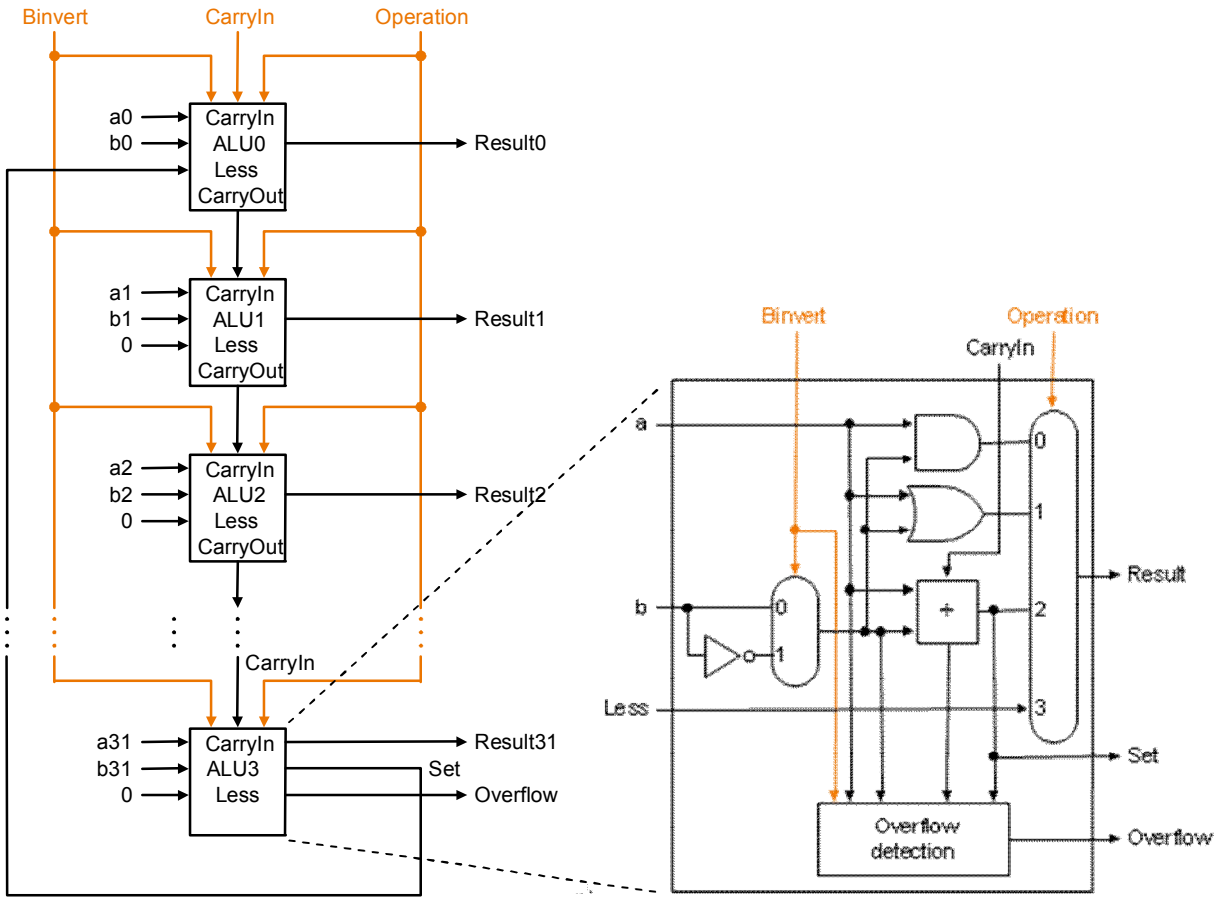

Figure 2 4-bit ALU

Appendix: MIPS assembly language revealed in chapter 2

Category	Instruction	Example	Meaning
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1
	load half	lh $s1,100($s2)	$s1 = Memory[$s2 + 100]
	store half	sh $s1,100($s2)	Memory[$s2 + 100] = $s1
	load byte	lb $s1,100($s2)	$s1 = Memory[$s2 + 100]
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1
	load upper immed.	lui $s1,100	$s1 = 100 * 2^{16}
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 |$s3)
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0
Uncondi-tional jump	jump	j 2500	go to 10000
	jump register	jr $ra	go to $ra
	jump and link	jal 2500	$ra = PC + 4; go to 10000