Name: . . . .

ID Number: . . . .

COMP203: Mid-Term Test

Model Solutions

13 April, 2005

Instructions

- Maximum time: **90 minutes**.
- Answer **all** the questions.
- There are 90 marks in total.
- Write your answers in the boxes in this test paper and hand in all sheets.
- Paper translation dictionaries are allowed.
- Non-programmable calculators are allowed.
- Every box with a heavy outline requires an answer.
- Page 11 shows some commonly used MIPS instructions and registers.

Questions	**Marks**
1. Basic Concepts	[10]
2. Registers, Memory, and Big Constants	[10]
3. Decision Making	[10]
4. Addressing Modes and Instruction Formats	[10]
5. Number Conversion	[10]
6. Boolean Expression and Logic Gates	[10]
7. Multiplication	[10]
8. Overflow Detection and Manipulation	[10]
9. Procedures/Functions	[10]
Total Marks	[90]

Question 1. Basic Concepts [10 marks]

(a) [3 marks] Briefly define the term *ALU* in the context of computer organisation.

ALU, or *alrithmetic and logic unit*, is a component of the processor that performs arithmetic and logic operations.

(b) [2 marks] Briefly define the term *control* in the context of computer organisation.

Control is a component of the processor that tells the datapath, memory, and I/O devices what to do according to the instructions of a program.

(c) [2 marks] Briefly define the term *instruction* in the context of computer organisation.

An instruction is an individual command to a computer.

(d) [3 marks] Briefly describe the major difference between combinational logic blocks and sequential logic blocks:

The major difference is that the combinational logic blocks do not have any memory while sequential logic blocks contain memory.

[The outputs of the combinational logic blocks purely depend on their inputs. The outputs of a sequential logic block usually depend on both the inputs and the current state (memory) of the block. In some cases, the outputs of a sequential logic block only change based on its memory.]

Question 2. Registers, Memory and Big Constants [10 marks]

(a) [6 marks] Consider the following C statement:

```
A[15] = A[20] + j;
```

Assume that register `$s1` holds integer variable `j` and that register `$s0` holds the base address of the integer array `A`. Write a sequence of MIPS instructions that directly corresponds to this statement. Use temporary registers if necessary.

```
lw $t0, 80($s0)
add $t0, $t0, $s1
sw $t0, 60($s0)
```

(b) [4 marks] Consider the following sequence of MIPS instructions:

```
lui $t1, 0x1234
ori $t2, $t1, 0x8201
addi $t3, $t1, 0x8201
```

What values will be stored in registers `$t1, $t2, $t3` after the above instructions are executed?

```
$t1 = 0x1234 0000

$t2 = 0x1234 8201

$t3 = 0x1233 8201
```

Question 3. Decision Making [10 marks]

Consider the following C code segment:

```
if (x >= 5)
   x = x - m;
else
   x = x + m;
x = x + 10;
```

Assume that the registers `$s0` and `$s1` hold the integer variables `x` and `m`, respectively.

Write a sequence of MIPS instructions that directly corresponds to this C code segment. Use temporary registers if necessary.

```
      slti $t0, $s0, 5
      bne $t0, $zero, else
      sub $s0, $s0, $s1
      j exit
else: add $s0, $s0, $s1
exit: addi $s0, $s0, 10
```

Question 4. Addressing Modes and Instruction Formats [10 marks]

Use the following sequence of MIPS instructions labelled as 1 to 8 to answer questions (a) and (b).

```
1           slt $t0, $s1, $s0
2           bne $t0, $zero, Else
3           sub $s1, $s1, $s0
4           addi $s1, $s1, 10
5           j Exit
6     Else: lw $t0, 4($s4)
7           add $s1, $s1, $t0
8     Exit: or $s1, $s1, $t0
```

(a) [8 marks] For each of the above 8 labelled instructions, state its addressing mode and instruction format.

label	addressing mode	instruction format
1	Register	R type
2	PC-relative	I type
3	Register	R type
4	Immediate/constant	I type
5	Pseudo-direct	J-type
6	Base	I type
7	Register	R type
8	Register	R type

(b) [2 marks]

Calculate the value of the branch relative address (*the offset in machine code*) of `Else` in Instruction 2 "`bne $t0, $zero, Else`". **Present the final result only** in the box below.

3

Question 5. Number Conversion [10 marks]

This question concerns different formats of numbers. **Write only the final answer into the boxes.**

(a) [2 marks] Convert the decimal number `-2049` into a 16-bit two's complement binary number.

1111 0111 1111 1111

(b) [2 marks] Convert the 16-bit two's complement binary number `1111 1111 0000 0000` into a decimal number.

-256

(c) [2 marks] Convert the IEEE 754 single precision binary number
`0000 0000 0000 0000 0000 0000 0000 0000` into a decimal number.

0

(d) [4 marks] Show the IEEE 754 binary representation of the the decimal floating point number `-5.125` in single precision format.

1100 0000 1010 0100 0000 0000 0000 0000

Question 6. Boolean Expression and Logic Gates [10 marks]

Given the following truth table for a PLA (Programmable Logic Array), answer questions (a) and (b):

Input			Output		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

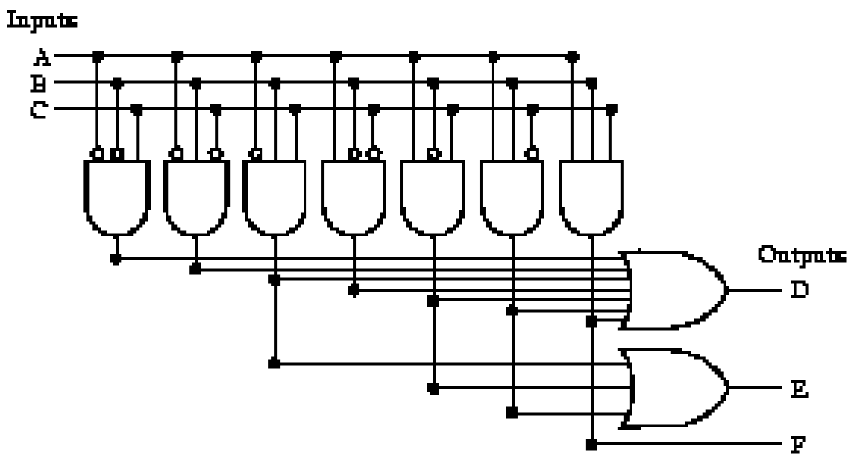
(a) [4 marks] Give a boolean expression for each of D, E and F based on the truth table.

$$D = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

$$E = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$

$$F = A \cdot B \cdot C$$

(b) [6 marks] Design a PLA (Programmable Logic Array) to implement the truth table based on the boolean expressions you gave in part (a).



Question 7. Multiplication

[10 marks]

Calculate the following multiplication using the Booth's algorithm:

`0110` × `1111`

Assume that the multiplicand and the multiplier are 4-bit 2's complement integers (consider the sign). Show your work in a table and identify your final result.

Loop	Action– steps	Mcand	Prod
0	Initialisation	0110	0000 1111 0
1	1c:Prod=Prod-Mcand	0110	**1010** 1111 0
	2: Prod>>1	0110	**1101 0111 1**
2	1d:no operation	0110	1101 0111 1
	2: Prod>>1	0110	**1110 101 1 1**
3	1d:no operation	0110	1110 1011 1
	2: Prod>>1	0110	**1111 0101 1**
4	1d: no operation	0110	1111 0101 1
	2: Prod>>1	0110	**1111 1010 1**

The final result is **1111 1010**.

Question 8. Overflow Detection and Manipulation [10 marks]

Assume that A and B are negative integers and that variables A, B, and C are placed in registers `$s1`, `$s2` and `$s3`, respectively. Write **at most 10** MIPS instructions in total to perform the following tasks:

- $C = A + B$;
- If there is no overflow, then add decimal constant 500 to C (`$s3`) and place the result in register `$s4`;
- Otherwise, set the least significant bit of C (`$s3`) to 0.

Use temporary registers if necessary.

```
            add $s3, $s1, $s2
            slt $t0, $s3, $zero
            bne $t0, $zero, noOverflow
            lui $t1, 0xffff
            ori $t1, $t1, 0xfffe
            and $s3, $s3, $t1
            j Exit
noOverflow: addi $s4, $s3, 500
Exit:
```

Question 9. Procedures/Functions [10 marks]

Given the following C procedure/function:

```
int test(int m, int n)
{
   int k;

   k = m + n - 3;

   return k;
}
```

Assume that register `$s0` holds the variable `k`. Write a sequence of MIPS instructions that directly corresponds to this function. Use temporary registers if necessary.

```
test:
  addi $sp, $sp, -4   # adjust stack
  sw $s0, 0($sp)      # push $s0
  add  $s0, $a0, $a1  # k = m + n
  addi $s0, $s0, -3   # k = m + n -3
  add $v0, $s0, $0    # $v0 for result return
  lw $s0, 0($sp)      # restore $s0
  addi $sp, $sp, 4    # adjust stack, pop
  jr $ra              # return
```

A Commonly Used MIPS Instructions

add	sub
lw	sw
addi	lui
and	or
andi	ori
sll	srl
jal	jr
j	
beq	bne
slt	slti
mult	div
mul	
lb	sb

B MIPS Registers — Numbers and Names

Name	Number	Usage
$zero	0	constant value 0
$at	1	reserved for assembler
$v0–$v1	2–3	values for results and expression evaluation
$a0–$a3	4-7	arguments, for functions/procedures
$t0–$t7	8-15	temporaries
$s0–$s7	16-23	saved. Fast locations for data
$t8–$t9	24-25	more temporaries
$k0–$k1	26-27	reserved for the OS
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address, for functions/procedures

****