THE UNIVERSITY OF WAIKATO
Department of Computer Science

COMP201Y Computer Systems 2003
Exercise 5 Test - 21st July 2003

Worth 11% — Marked out of: 30

Time allowed: 45 Min

---

1. In Exercise 4 you used *WRAMPsim* to simulate the execution of WRAMP instructions on a WRAMP data-path. Figure 1 shows the architecture of the WRAMP CPU that you used in the Exercise. Although not shown on the diagram there are control lines between the control unit and each of the components, which are used to control the flow of data on the data-path. The control signals for each of the components and their functionality is defined in Tables 1 and 2 of Appendix A.

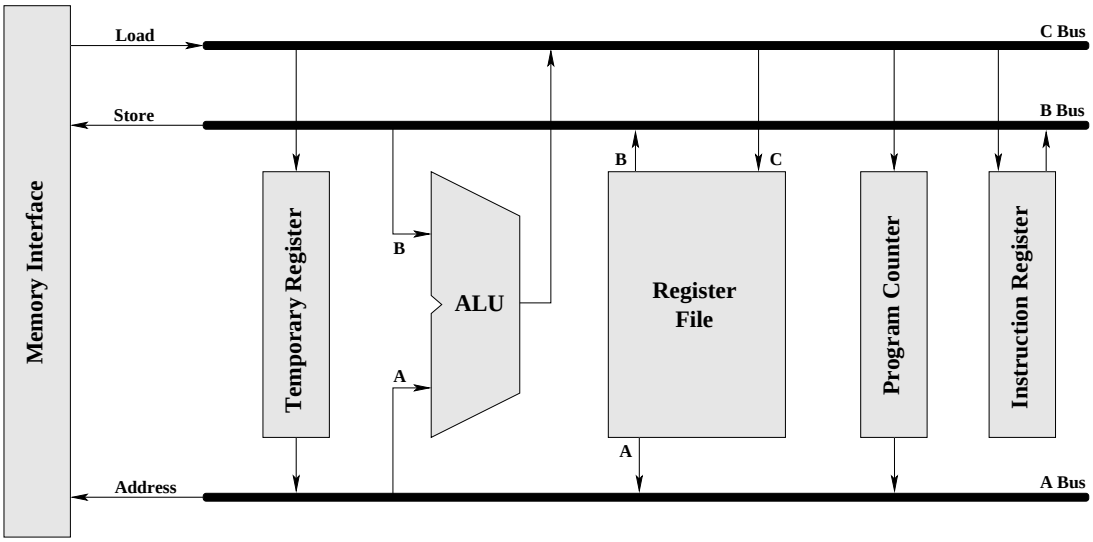

Figure 1: Data-path Architecture for WRAMP CPU

(a) Define the control steps necessary to fetch and execute the instruction `sw $3, 3($3)`. Remember to include the control steps to fetch the instruction from memory, and increment the program counter. Make sure you clearly indicate in your answer which control signals are defined in each of the control steps.

(4 marks)

(b) Define the control steps necessary to fetch and execute the instruction `jalr $7`. Remember to include the control steps to fetch the instruction from memory, and increment the program counter. Make sure you clearly indicate in your answer which control signals are defined in each of the control steps.

(4 marks)

2. Figure 2 contains a correct solution for Question 4 of Exercise 5 that asked you to write a program to output 25 'X's to the serial port connected to the *terminal*. Appendix B defines the operation of the serial ports for this question.

   (a) If lines 8 - 10 of this program were removed, would it still output 25 'X's to the *terminal* when run? If not, what would the likely output be? Why?

   (4 marks)

   (b) What change(s) would have to be made to the program so that it would output 10 'X's (instead of 25) when run? On your answer you should clearly indicate the line numbers of the line(s) you have changed.

   (2 marks)

   (c) What change(s) would have to be made to the program so that it would output 'W's instead of 'X's when it was run? On your answer you should clearly indicate the line numbers of the line(s) you have changed.

   (2 marks)

```
1:  .global main
2:  .text
3:
4:  main:
5:          code removed
```

Figure 2: Code for Question 2

3. In Question 5 of Exercise 5 you were asked to write a program that continually reads a character from the serial port connected to the *terminal*, converts all *lowercase* characters to *uppercase* and outputs the character to the serial port connected to the *Linux machine*. For example, if an 'a' is the input then 'A' is the output. If an 'A' is the input then 'A' is the output. Figure 3 contains a *correct* solution for this question. Appendix B shows the format of the status register used in the assignment and this solution.

```
1: .global main
2: .text
3: main:
4:        code removed
```

Figure 3: Code for question 3

(a) What changes would be necessary to the program so that it reads characters from the *Linux machine* and outputs characters to the *terminal*?

(4 marks)

(b) What changes would need to be made to this program so it swaps the case of the characters typed on the *terminal* and outputs them to the *Linux Machine*? For example, if an 'a' is input on the *terminal* then an 'A' is output to the *Linux machine*. If an 'A' is the input then 'a' is the output.

(10 marks)

A Definition of the WRAMP Control Signals for Question 1

Component	Signal Name	Description
Register File	a_out	Causes the contents of the register selected by sel_a to be output onto the A bus.
	sel_a	Select which register will be output onto the A bus if a_out is asserted.
	b_out	Causes the contents of the register selected by sel_b to be output onto the B bus.
	sel_b	Select which register will be output onto the B bus if b_out is asserted.
	c_in	Causes the value from the C bus to be written into the register selected by sel_c.
	sel_c	Select which register to write the value from the C bus into when the c_in signal is asserted.
ALU	alu_out	Causes the result of the current ALU function selected by alu_func to be output to the C bus.
	alu_func	Defines the current operation that the ALU should perform. ALU functions are defined in table 2.
Memory Interface	mem_read	Causes the contents of the memory address specified on the A bus to be read and output onto the C bus.
	mem_write	Causes the value on the B bus to be written into the memory address specified on the A bus.
Program Counter	pc_out	Causes the contents of the PC register to be output onto the A bus.
	pc_in	Causes the value on the C bus to be written into the PC.
Instruction Register	imm_16_out	Causes the least significant 16 bits of the IR to be output onto the B bus.
	imm_20_out	Causes the least significant 20 bits of the IR to be output onto the B bus.
	sign_extend	Causes the output from the IR to be sign extended to 32bits.
	ir_in	Causes the value on the C bus to be written into the IR.
Temporary Register	temp_out	Causes the contents of the temporary register to be output onto the A bus.
	temp_in	Causes the value on the C bus to be written into the temporary register.

Table 1: Descriptions of each of the control signals

All arithmetic and test/set operations have both signed and unsigned variants. The unsigned variant is indicated by an operation with a 'u' suffix. A signed variant treats all inputs as signed integers while the unsigned variant treats inputs as unsigned integers.

Type	**Name**	**Function**	**Description**
Arithmetic	`add, addu`	A + B	Perform an integer addition between A and B.
	`sub, subu`	A - B	Perform an integer subtraction between A and B.
	`mult, multu`	A * B	Perform an integer multiplication between A and B.
	`div, divu`	A / B	Perform an integer division between A and B.
	`rem, remu`	A mod B	Obtain the remainder from an integer division between A and B.
Bitwise	`sll`	A << B	Shift the value on A left by the number of places specified by B. Fill with zeros.
	`and`	A AND B	Perform a bitwise AND between A and B.
	`srl`	A >> B	Shift the value on A right by the number of places specified by B. Fill with zeros.
	`or`	A OR B	Perform a bitwise OR between A and B.
	`sra`	A >> B	Shift the value on A right by the number of places specified by B. Fill with MSB.
	`xor`	A XOR B	Perform a bitwise XOR between A and B.
Test/set	`slt, sltu`	out = 1 if (A < B) else out = 0	Set out if A is less than B
	`sgt, sgtu`	out = 1 if (A > B) else out = 0	Set out if A is greater than B
	`sle, sleu`	out = 1 if (A $\leq$ B) else out = 0	Set out if A is less than or equal to B
	`sge, sgeu`	out = 1 if (A $\geq$ B) else out = 0	Set out if A is greater than or equal to B
	`seq, sequ`	out = 1 if (A = B) else out = 0	Set out if A is equal to B
	`sne, sneu`	out = 1 if (A $\neq$ B) else out = 0	Set out if A is not equal to B
Misc	`lhi`	$out_{[31...16]} = B_{[15...0]}$ $out_{[15...0]} = 0$	Set the upper 16 bits of out to be the lower 16 bits of B. Lower 16 bits of out set to zero.
	`inc`	out = A + 1	Increment A

Table 2: ALU Operations

B Details of the serial ports for Questions 2 and 3

The REX board provides two serial interfaces, one of which is connected to the Linux machine and the other is connected to the terminal that should be sitting on the shelf above the board.

For each of the serial interfaces there are 4 registers accessible to the CPU. These registers are the transmit data register, the receive data register, the status register and the control register.

The transmit data register holds the value that is to be sent out of the serial port. The receive data register holds the value that has been received in the serial port. The status register indicates if there is a value in the receive data register and also if the value in the transmit data register has been sent. The control register allows the configuration of the serial port. For this Exercise the control register will have already been configured for you by the monitor so it will not need to be altered. Each serial port has a base address and the 4 registers for each serial port are expressed as an offset from this address. The base addresses for each port are defined in table 3 and the offsets are defined in table 4. The format of the status register for each of the serial devices is shown in table 5.

Port	**Base Address**
Linux machine	0x70000
Terminal	0x71000

Table 3: Base addresses for each serial port

Register	**Address**
Transmit Data	Base + 0
Receive Data	Base + 1
Control	Base + 2
Status	Base + 3

Table 4: Offsets for each register

Bit	**Function**
0	Receive Data Ready 1 if data in receive data register, 0 otherwise
1	Transmit Data Sent 1 if ready for next character, 0 if data is still being transmitted

Table 5: Status register format

C WRAMP Instruction Set Summary

- $\int$ denotes a signed operation. For immediate values this implies sign extension. For bitwise shift right this implies an arithmetic shift.
- MEM[R_s + offset] denotes the contents of the memory location addressed by the sum of register R_s and the 20 bit offset.
- On instruction fetch the Program Counter is incremented. This means that branch instructions operate relative to the address of the following instruction, and `jal` and `jalr` instructions save the address of the following instruction.

Assembler	Machine code	Function	Description
`add` R_d, R_s, R_t	0000 dddd ssss 0000 0000 0000 0000 tttt	$R_d \leftarrow R_s + R_t$	Add
`addi` R_d, R_s, immed	0001 dddd ssss 0000 iiii iiii iiii iiii	$R_d \leftarrow R_s + \int(immed)$	Add Immediate
`addu` R_d, R_s, R_t	0000 dddd ssss 0001 0000 0000 0000 tttt	$R_d \leftarrow R_s + R_t$	Add Unsigned
`addui` R_d, R_s, immed	0001 dddd ssss 0001 iiii iiii iiii iiii	$R_d \leftarrow R_s + immed$	Add Unsigned Immediate
`sub` R_d, R_s, R_t	0000 dddd ssss 0010 0000 0000 0000 tttt	$R_d \leftarrow R_s - R_t$	Subtract
`subi` R_d, R_s, immed	0001 dddd ssss 0010 iiii iiii iiii iiii	$R_d \leftarrow R_s - \int(immed)$	Subtract Immediate
`subu` R_d, R_s, R_t	0000 dddd ssss 0011 0000 0000 0000 tttt	$R_d \leftarrow R_s - R_t$	Subtract Unsigned
`subui` R_d, R_s, immed	0001 dddd ssss 0011 iiii iiii iiii iiii	$R_d \leftarrow R_s - immed$	Subtract Unsigned Immediate
`mult` R_d, R_s, R_t	0000 dddd ssss 0100 0000 0000 0000 tttt	$R_d \leftarrow R_s \times R_t$	Multiply
`multi` R_d, R_s, immed	0001 dddd ssss 0100 iiii iiii iiii iiii	$R_d \leftarrow R_s \times \int(immed)$	Multiply Immediate
`multu` R_d, R_s, R_t	0000 dddd ssss 0101 0000 0000 0000 tttt	$R_d \leftarrow R_s \times R_t$	Multiply Unsigned
`multui` R_d, R_s, immed	0001 dddd ssss 0101 iiii iiii iiii iiii	$R_d \leftarrow R_s \times immed$	Multiply Unsigned Immediate
`div` R_d, R_s, R_t	0000 dddd ssss 0110 0000 0000 0000 tttt	$R_d \leftarrow R_s \div R_t$	Divide
`divi` R_d, R_s, immed	0001 dddd ssss 0110 iiii iiii iiii iiii	$R_d \leftarrow R_s \div \int(immed)$	Divide Immediate
`divu` R_d, R_s, R_t	0000 dddd ssss 0111 0000 0000 0000 tttt	$R_d \leftarrow R_s \div R_t$	Divide Unsigned
`divui` R_d, R_s, immed	0001 dddd ssss 0111 iiii iiii iiii iiii	$R_d \leftarrow R_s \div immed$	Divide Unsigned Immediate
`rem` R_d, R_s, R_t	0000 dddd ssss 1000 0000 0000 0000 tttt	$R_d \leftarrow R_s \,\%\, R_t$	Remainder
`remi` R_d, R_s, immed	0001 dddd ssss 1000 iiii iiii iiii iiii	$R_d \leftarrow R_s \,\%\, \int(immed)$	Remainder Immediate
`remu` R_d, R_s, R_t	0000 dddd ssss 1001 0000 0000 0000 tttt	$R_d \leftarrow R_s \,\%\, R_t$	Remainder Unsigned
`remui` R_d, R_s, immed	0001 dddd ssss 1001 iiii iiii iiii iiii	$R_d \leftarrow R_s \,\%\, immed$	Remainder Unsigned Immediate
`lhi` R_d, immed	0011 dddd ssss 1110 iiii iiii iiii iiii	$R_d \leftarrow immed \ll 16$	Load High Immediate
`la` R_d, address	1100 dddd 0000 aaaa aaaa aaaa aaaa aaaa	$R_d \leftarrow address$	Load Address

Table 6: Arithmetic Instructions

`and` R_d, R_s, R_t	0000 dddd ssss 1011 0000 0000 0000 tttt	$R_d \leftarrow R_s \; AND \; R_t$	Bitwise AND
`andi` R_d, R_s, immed	0001 dddd ssss 1011 iiii iiii iiii iiii	$R_d \leftarrow R_s \; AND \; immed$	Bitwise AND Immediate
`or` R_d, R_s, R_t	0000 dddd ssss 1101 0000 0000 0000 tttt	$R_d \leftarrow R_s \; OR \; R_t$	Bitwise OR
`ori` R_d, R_s, immed	0001 dddd ssss 1101 iiii iiii iiii iiii	$R_d \leftarrow R_s \; OR \; immed$	Bitwise OR Immediate
`xor` R_d, R_s, R_t	0000 dddd ssss 1111 0000 0000 0000 tttt	$R_d \leftarrow R_s \; XOR \; R_t$	Bitwise XOR
`xori` R_d, R_s, immed	0001 dddd ssss 1111 iiii iiii iiii iiii	$R_d \leftarrow R_s \; XOR \; immed$	Bitwise XOR Immediate
`sll` R_d, R_s, R_t	0000 dddd ssss 1010 0000 0000 0000 tttt	$R_d \leftarrow R_s \ll R_t$	Shift Left Logical
`slli` R_d, R_s, immed	0001 dddd ssss 1010 iiii iiii iiii iiii	$R_d \leftarrow R_s \ll immed$	Shift Left Logical Immediate
`srl` R_d, R_s, R_t	0000 dddd ssss 1100 0000 0000 0000 tttt	$R_d \leftarrow R_s \gg R_t$	Shift Right Logical
`srli` R_d, R_s, immed	0001 dddd ssss 1100 iiii iiii iiii iiii	$R_d \leftarrow R_s \gg immed$	Shift Right Logical Immediate
`sra` R_d, R_s, R_t	0000 dddd ssss 1110 0000 0000 0000 tttt	$R_d \leftarrow \int(R_s \gg R_t)$	Shift Right Arithmetic
`srai` R_d, R_s, immed	0001 dddd ssss 1110 iiii iiii iiii iiii	$R_d \leftarrow \int(R_s \gg immed)$	Shift Right Arithmetic Immediate

Table 7: Bitwise Instructions

slt R_d, R_s, R_t	0010 dddd ssss 0000 0000 0000 0000 tttt	$R_d \leftarrow R_s < R_t$	Set on Less than
slti R_d, R_s, immed	0011 dddd ssss 0000 iiii iiii iiii iiii	$R_d \leftarrow R_s < \int(immed)$	Set on Less than Immediate
sltu R_d, R_s, R_t	0010 dddd ssss 0001 0000 0000 0000 tttt	$R_d \leftarrow R_s < R_t$	Set on Less than Unsigned
sltui R_d, R_s, immed	0011 dddd ssss 0001 iiii iiii iiii iiii	$R_d \leftarrow R_s < immed$	Set on Less than Unsigned Immediate
sgt R_d, R_s, R_t	0010 dddd ssss 0010 0000 0000 0000 tttt	$R_d \leftarrow R_s > R_t$	Set on Greater than
sgti R_d, R_s, immed	0011 dddd ssss 0010 iiii iiii iiii iiii	$R_d \leftarrow R_s > \int(immed)$	Set on Greater than Immediate
sgtu R_d, R_s, R_t	0010 dddd ssss 0011 0000 0000 0000 tttt	$R_d \leftarrow R_s > R_t$	Set on Greater than Unsigned
sgtui R_d, R_s, immed	0011 dddd ssss 0011 iiii iiii iiii iiii	$R_d \leftarrow R_s > immed$	Set on Greater than Unsigned Immediate
sle R_d, R_s, R_t	0010 dddd ssss 0100 0000 0000 0000 tttt	$R_d \leftarrow R_s \leq R_t$	Set on Less than or Equal
slei R_d, R_s, immed	0011 dddd ssss 0100 iiii iiii iiii iiii	$R_d \leftarrow R_s \leq \int(immed)$	Set on Less or Equal Immediate
sleu R_d, R_s, R_t	0010 dddd ssss 0101 0000 0000 0000 tttt	$R_d \leftarrow R_s \leq R_t$	Set on Less or Equal Unsigned
sleui R_d, R_s, immed	0011 dddd ssss 0101 iiii iiii iiii iiii	$R_d \leftarrow R_s \leq immed$	Set on Less or Equal Unsigned Imm
sge R_d, R_s, R_t	0010 dddd ssss 0110 0000 0000 0000 tttt	$R_d \leftarrow R_s \geq R_t$	Set on Greater than or Equal
sgei R_d, R_s, immed	0011 dddd ssss 0110 iiii iiii iiii iiii	$R_d \leftarrow R_s \geq \int(immed)$	Set on Greater or Equal Immediate
sgeu R_d, R_s, R_t	0010 dddd ssss 0111 0000 0000 0000 tttt	$R_d \leftarrow R_s \geq R_t$	Set on Greater or Equal Unsigned
sgeui R_d, R_s, immed	0011 dddd ssss 0111 iiii iiii iiii iiii	$R_d \leftarrow R_s \geq immed$	Set on Greater or Equal Unsigned Imm
seq R_d, R_s, R_t	0010 dddd ssss 1000 0000 0000 0000 tttt	$R_d \leftarrow R_s = R_t$	Set on Equal
seqi R_d, R_s, immed	0011 dddd ssss 1000 iiii iiii iiii iiii	$R_d \leftarrow R_s = \int(immed)$	Set on Equal Immediate
sequ R_d, R_s, R_t	0010 dddd ssss 1001 0000 0000 0000 tttt	$R_d \leftarrow R_s = R_t$	Set on Equal Unsigned
sequi R_d, R_s, immed	0011 dddd ssss 1001 iiii iiii iiii iiii	$R_d \leftarrow R_s = immed$	Set on Equal Unsigned Immediate
sne R_d, R_s, R_t	0010 dddd ssss 1010 0000 0000 0000 tttt	$R_d \leftarrow R_s \neq R_t$	Set on Not Equal
snei R_d, R_s, immed	0011 dddd ssss 1010 iiii iiii iiii iiii	$R_d \leftarrow R_s \neq \int(immed)$	Set on Not Equal Immediate
sneu R_d, R_s, R_t	0010 dddd ssss 1011 0000 0000 0000 tttt	$R_d \leftarrow R_s \neq R_t$	Set on Not Equal Unsigned
sneui R_d, R_s, immed	0011 dddd ssss 1011 iiii iiii iiii iiii	$R_d \leftarrow R_s \neq immed$	Set on Not Equal Unsigned Immediate

Table 8: Test Instructions

Branch Instructions			
j address	0100 0000 0000 aaaa aaaa aaaa aaaa aaaa	$PC \leftarrow Address$	Jump
jr R_s	0101 0000 ssss 0000 0000 0000 0000 0000	$PC \leftarrow R_s$	Jump to Register
jal address	0110 0000 0000 aaaa aaaa aaaa aaaa aaaa	\$ra $\leftarrow PC$, $PC \leftarrow Address$	Jump and Link
jalr R_s	0111 0000 ssss 0000 0000 0000 0000 0000	\$ra $\leftarrow PC$, $PC \leftarrow R_s$	Jump and Link Register
beqz R_s, offset	1010 0000 ssss oooo oooo oooo oooo oooo	$if(R_s = 0)\ PC \leftarrow PC + offset$	Branch on equal to 0
bnez R_s, offset	1011 0000 ssss oooo oooo oooo oooo oooo	$if(R_s \neq 0)\ PC \leftarrow PC + offset$	Branch on not equal to 0
Memory Instructions			
lw R_d, offset(R_s)	1000 dddd ssss oooo oooo oooo oooo oooo	$R_d \leftarrow MEM[R_s + offset]$	Load word
sw R_d, offset(R_s)	1001 dddd ssss oooo oooo oooo oooo oooo	$MEM[R_s + offset] \leftarrow R_d$	Store word
Special Instructions			
movgs R_d, R_s	0011 0000 0000 1100 0000 0000 0000 0000	$R_d \leftarrow R_s$	Move General to Special Register
movsg R_d, R_s	0011 0000 0000 1101 0000 0000 0000 0000	$R_d \leftarrow R_s$	Move Special to General Register
break	0010 0000 0000 1100 0000 0000 0000 0000		Generate Break Point Exception
syscall	0010 0000 0000 1101 0000 0000 0000 0000		Generate Syscall Exception
rfe	0010 0000 0000 1110 0000 0000 0000 0000	$PC \leftarrow$ \$ear	Return from Exception

Table 9: Other Instructions