THE UNIVERSITY OF WAIKATO
Department of Computer Science

COMP201Y Computer Systems 2004
Mid Year Test - Monday May 31, 2004, 5pm PWC

- Please answer ALL questions.
- Time allowed: 90 minutes
- Marked out of: 90
- Contribution toward final grade: 25%
- Please answer questions on the answer sheet provided.
- Write your name and student ID on each page
- This is a CLOSED book test, although calculators are permitted

---

1 Multichoice (1 mark each)

1. Which of the following is not a component of the central processing unit?
   (a) buses
   (b) arithmetic logic unit
   (c) temporary registers
   (d) i/o control registers
2. Which of the following formats is used for bitmap graphics?
   (a) TIFF
   (b) AVI
   (c) Postscript
   (d) PDF
3. What is the correct sequence for translating a file from a high level language to executable machine code?
   (a) compiling - assembling - debugging
   (b) assembling - linking -debugging
   (c) compiling - assembling - linking - loading
   (d) assembling - compiling -linking
4. Assembler directives are used in an assembly language program to...
   (a) generate instructions
   (b) replace instructions
   (c) give commands to the assembler
   (d) comment the code

5. The gif file format is used to store...

   (a) bitmap graphics data
   (b) vectorised graphics data
   (c) character data
   (d) audio data

(a)

```
        addi $2, $0, 1
label:  sgei $4, $2, 4
        bnez $4, label2
        ...
        addi $2, $2, 1
        j    label
label2:
```

(b)

```
        addi $2, $0, 1
label:  sgti $4, $2, 4
        bnez $4, label2
        ...
        addi $2, $2, 1
        j    label
label2:
```

(c) ...

```
        slt   $13, $3, $4
        beqz  $13, label
        ...
        j     label2
label:
        ...
label2:
```

(d) ...

```
        sge   $13, $3, $4
        beqz  $13, label
        ...
label:
        ...
```

Figure 1: Code for multichoice questions 6 – 8

6. Which of the WRAMP code segments in Figure 1 would implement an if statement without an else clause?

   (a) a
   (b) b
   (c) c
   (d) d

7. Which of the WRAMP code segments in Figure 1 would implement an if statement with an else clause?

   (a) a
   (b) b
   (c) c
   (d) d

8. Which of the WRAMP code segments in Figure 1 would implement a loop which iterates four (4) times?

   (a) a
   (b) b
   (c) c
   (d) d

9. Which of the following statements is true?

   (a) EBCDIC is the international standard version of ASCII.

   (b) ASCII is an eight bit code for alphanumeric characters.

   (c) Unicode is an eight bit code for international character sets.

   (d) In either ASCII or EBCDIC changing the case of characters can be done by changing only one bit.

10. Which one of the following statements is true?

    (a) WRAMP registers are composed of 32 JK Flip-flops

    (b) The register file uses tri-state outputs.

    (c) Registers implement the fetch-decode-execute cycle as a state machine.

    (d) In modern CPUs' data transfers between registers within the CPU are significantly slower than those between registers and main memory.

2 Short Answer

1. Convert the following numbers (show any working):

   (a) $5B_{16}$ into decimal

   (b) 55_8 into Hexadecimal

   (c) 76_{10} into BCD

   (d) 11011010_2 into octal

(6 marks)

2. What is the EBCDIC coding scheme used to represent? How many bits are used?

(2 marks)

3. Which WRAMP intructions are used to pass data to and from I/O devices?

(2 marks)

4. Explain the difference between the operation of the following two WRAMP instructions in the case where `$4` = `0xfffffffe` and `$5` = `0x00000003`. You must also indicate the result that each instruction will produce.

   (a) `slt $3, $4, $5`

   (b) `sltu $3, $4, $5`

(4 marks)

5. Convert -47 decimal to 2's complement 32 bit format. Show your working.

(2 marks)

6. Name the four types of operation that all programming languages must support. Give an example of a WRAMP command for each type.

(4 marks)

7. What problem is polled I/O used to solve?

(2 marks)

8. What value will `$2` contain after the following two instructions have been executed? Show your work.

```
subi    $2, $0, 71
srai    $2, $2, 2
```

(2 marks)

9. Register `$ra` serves a special function in WRAMP. Explain in detail which instruction(s) use it and how it is important.

(4 marks)

10. Convert the following 32 bit IEEE-754 format number to decimal. The IEEE-754 format is shown in Figure 2. Show your working.

```
01000000 11010100 00000000 00000000 (0x40d40000)
```

(4 marks)

$$(-1)^{S} * (1.M)*2^{(E - Bias)}$$

where:

S = Sign
M = Mantissa
E = Exponent
Bias = 127

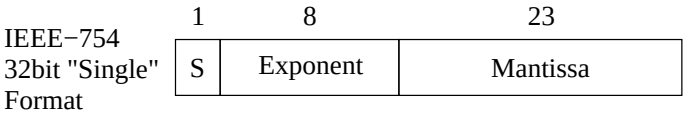

Figure 2: IEEE-754 Format

11. If the instruction `beqz $3, 0x17` is located at the address `0xbaa`, if this instruction is executed and the branch is taken (i.e. `$3` = 0), then which memory location will the next instruction be fetched from?

(4 marks)

12. Using the supplied WRAMP Instruction Set Architecture document, assemble the WRAMP instruction: `sequ $11, $7, $13` to machine code.

(4 marks)

13. Using the supplied WRAMP Instruction Set Architecture document, disassemble the WRAMP machine code instruction:

```
0010 0111 0110 1011 0000 0000 0000 1010
```

(4 marks)

14. The header for a .WAV sound file is shown in Figure 14. Determine (in decimal) The number of channels, the sample rate, the size of each sample and the number of samples of this audio clip. You should show any working. The header fields and format for WAV files are shown in Figure 4.

(4 marks)

```
00000000: 5249 4646 E20F 0000 5741 5645 666D 7420  RIFF....WAVEfmt
00000010: 1000 0000 0100 0100 44AC 0000 8858 0100  ........D....X..
00000020: 0200 1000 6461 7461 B80F 0000 0001 0001  ....data........
00000030: 0001 0001 0001 0001 0001 0001 0001 0001  ................
00000040: 0001 0001 0003 0003 0003 0003 0003 0004  ................
```

Figure 3: HEX dump of WAV file header

Field Name	File Offset	Field Size	Endian	Value
Chunk ID	0	4	big	"RIFF"
ChunkSize	4	4	little	Filesize - 8
Format	8	4	big	"WAVE"
Subchunk1ID	12	4	big	"fmt "
Subchunk1Size	16	4	little	Size of SubChunk1 -8
AudioFormat	20	2	little	1 for PCM
NumChannels	22	2	little	No. of audio channels
SampleRate	24	4	little	Samples per second
ByteRate	28	4	little	Bytes/sec (all channels)
BlockAlign	32	2	little	Bytes/sample (all channels)
BitsPerSample	34	2	little	Bits/sample (one channel)
SubChunk2ID	36	4	big	"data"
SubChunk2Size	40	4	little	Bytes of data
data	44	variable	little	audio sample data

Figure 4: WAV file header fields

15. Figure 5 contains a WRAMP program. You are to trace the execution in order to determine what it does. A break point has been set in the middle of the loop (at line 10) to achieve this.

    (a) What would the contents of registers `$2` to `$5` be each time this break point was encountered when the program was run?

    (6 marks)

    (b) What does this program do?

    (2 marks)

```
1:   .global main
2:   main:
3:         lhi  $3, 0x1234
4:         addi $3, $3, 0x5678
5:         add  $5, $0, $0
6:         addi $4, $0, 4
7:   loop:
8:         addi $2, $0, 0xff
9:         and  $2, $2, $3
10:        or   $5, $5, $2
11:        slli $5, $5, 8
12:        srli $3, $3, 8
13:        subi $4, $4, 1
14:        bnez $4, loop
15:        jr   $ra
```

Figure 5: WRAMP program for Question 15

16. Figures 6 and 7 show the "C" code and the WRAMP code generated by the `wcc` compiler for a recursive function `fact`.

    (a) For the C function indicate which lines of the WRAMP code were generated for:

    i. `if ( value != 0 )`

    ii. `result = value * fact(value);`

    (4 marks)

    (b) Draw a diagram to show what the stack frame created when the function `fact` is called will look like. On your diagram clearly indicate the purpose of each entry in the stack frame. Your diagram should also clearly indicate where the two parameters that are passed to the function are stored on the stack.

    (8 marks)

```
1:   int fact(int value)
2:   {
3:   int result;
4:
5:   result = 1;
6:
7:   if ( value != 0 )
8:   {
9: result = value * fact(value);
10:   }
11:
12:   return result;
```

Figure 6: C code for Question 16

```
1:  .global fact
2:  .text
3:  fact:
4:          subui $sp,  $sp,  5
5:          sw    $12,  1($sp)
6:          sw    $13,  2($sp)
7:          sw    $ra,  3($sp)
8:          addui $13,  $0,   1
9:          sw    $13,  4($sp)
10:         lw    $13,  5($sp)
11:         seq   $13,  $13,  $0
12:         bnez  $13,  L.2
13:         lw    $13,  5($sp)
14:         sw    $13,  0($sp)
15:         jal   fact
16:         addu  $12,  $0,   $1
17:         mult  $13,  $13,  $12
18:         sw    $13,  4($sp)
19:  L.2:
20:         lw    $1,   4($sp)
21:  L.1:
22:         lw    $12,  1($sp)
23:         lw    $13,  2($sp)
24:         lw    $ra,  3($sp)
25:         addui $sp,  $sp,  5
26:         jr    $ra
```

Figure 7: WRAMP code generated by `wcc` for C program in Figure 6

17. (a) Figure 8 shows the architecture of the WRAMP CPU that contains three internal buses. Label the copy this diagram on your answersheet. Identify all the major components, including the buses and ports.

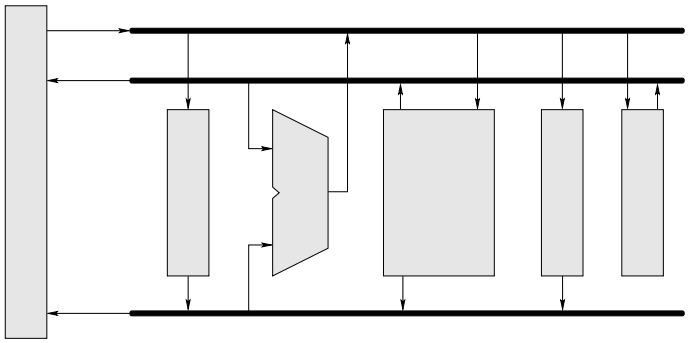

Figure 8: Datapath Architecture for WRAMP CPU

(6 marks)

(b) Name and describe the operation of the cycle a CPU goes through in order to execute each instruction. Name the steps that are repeated on every cycle.

(6 marks)

(c) The computer architecture where program instructions and data are stored separately from the CPU is named after which mathematician?

(2 marks)

A 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 1: 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 2: 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 3: 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 4: Other Instructions