University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

SEMESTER II EXAMINATION – 2011/2012

COMP 30080

Processor Design

SOLUTIONS

Prof A. Mille

Mr J. Dunnion

Dr C.J. Bleakley*

Time allowed: 2 hours

Instructions for candidates

Answer question 1 and any three other questions. All questions carry equal marks. The paper is marked out of 80 in total. A MIPS Reference is provided at the end of the question paper.

1.

(a) Name the three steps in the Von Neumann execution cycle.

Fetch, decode, execute

(b) In terms of computer systems, what does the acronym *ISA* stand for?

Instruction set architecture

(c) In terms of computer architecture, what is *assembly language*?

"Assembly language is a symbolic representation of machine instructions".

(d) Provide two examples of MIP32 assembler directives.

.byte, .word, .ascii, .align

(e) What MIPS instruction causes the processor to simultaneously jump and save the value of PC+4 in the return address register?

jal (jump and link)

(f) Is the following statement true or false? "In the MIPS32 architecture, arithmetic instruction can take memory addresses as operands."

False

(g) An assembly language, what is a *pseudo-instruction*?

A commonly used instruction that is not in the target instruction set but which the compiler accepts and translates to the target ISA equivalent

(h) In terms of the assembly language tool flow, what does the *linker* do?

Creates an executable binary file from multiple object module files.

(i) In big endian architectures, which has the larger address - the least significant byte or the most significant byte?

LSB

(j) Name two commonly used processor benchmark suites.

SPEC CPU2000, SPECweb99, LINPACK

(k) In terms of pipelined processors, what is *forwarding*?

A method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer-visible registers or memory.

(l) In the context of pipelined processors, what is *branch prediction*?

A method of resolving a branch hazard that assumes a given outcome for the branch and proceeds on that assumption rather than waiting to ascertain the actual outcome

(m) In the context of processors, what does the term *multiple issue* mean?

"A scheme whereby multiple instructions are launched in one clock cycle."

(n) In the context of processors, what does the term *speculation* mean?

"Speculation is an approach whereby the compiler or processor guesses the outcome of an instruction to remove it as a dependence in executing other instructions."

(o) What does the term *ILP* stand for?

Instruction Level Parallelism

(20 marks in total)

2.

(a) Write a MIPS32 assembly language subroutine that counts the number of upper case letters in a string of ASCII characters (see Table 1 below). The base address of the string and the number of characters in the string should be inputs to the subroutine. The result should be the return value of the subroutine. Note: pseudo-instructions can be used.

(10 marks)

(b) Write a MIPS32 assembly language program that will test the function written in part (a). The program should call the subroutine with the test input "COMP 20200". The number of characters in the string is provided as an input to the program and is held in memory prior to execution. The program should store the result of the count in memory.

(10 marks)

(20 marks in total)

Table 1. ASCII codes.

Character	**ASCII code (decimal)**
space	32
A	65
Z	90
a	97
z	122

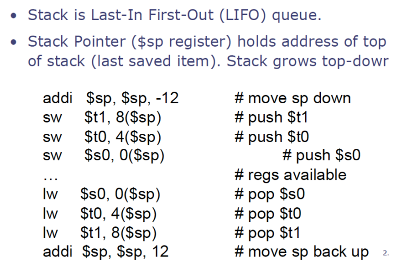
```
                .data
String:         .ascii  "COMP 20020"
Length:         .align  2
Count:          .word   10
                .word   0
                .text
                .globl  main
main:           la      $t0, String          # load data base address
                lw      $a0, 0($t0)          #
                la      $t1, Length          # load input data length
                lw      $a1, 0($t1)          #
                jal     Count                # call count
                la      $t2, Count           # store result
                sw      $v0, 0($t2)
                li      $v0, 10              # system call for exit
                syscall                      # Exit!


Count:          move    $v0, $zero           # v0 is number of spaces found
Loop:           lb      $t0, 0($a0)          # load character
                li      $t1, 65              # if < 65 then goto Next
                blt     $t0, $t1, Next
                li      $t1, 90              # if >90 then goto Next
                bgt     $t0, $t1, Next
                addi    $v0, $v0, 1          # else increment space count
Next:           addi    $a0, $a0, 1          # increase character address
                addi    $a1, $a1, -1         # decrement loop count
                bne     $a1, $zero, Loop     # if not finished then Loop
                jr      $ra
```

3.

(a) With the aid of an example, explain how the stack is used to preserve register values during procedure execution on a MIPS processor.

(5 marks)



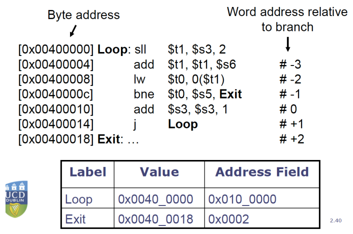
- Stack is Last-In First-Out (LIFO) queue.
- Stack Pointer ($sp register) holds address of top of stack (last saved item). Stack grows top-dowr

```
addi   $sp, $sp, -12        # move sp down
sw     $t1, 8($sp)          # push $t1
sw     $t0, 4($sp)          # push $t0
sw     $s0, 0($sp)                 # push $s0
…                           # regs available
lw     $s0, 0($sp)          # pop $s0
lw     $t0, 4($sp)          # pop $t0
lw     $t1, 8($sp)          # pop $t1
addi  $sp, $sp, 12          # move sp back up
```

2.

(5 marks)

(b) What form of addressing is used in branch instructions? With the aid of an example, explain how it works.


Byte address
Word address relative to branch

```
[0x00400000] Loop: sll     $t1, $s3, 2
[0x00400004]       add     $t1, $t1, $s6      # -3
[0x00400008]       lw      $t0, 0($t1)        # -2
[0x0040000c]       bne     $t0, $s5, Exit     # -1
[0x00400010]       add     $s3, $s3, 1        # 0
[0x00400014]       j       Loop               # +1
[0x00400018] Exit: …                          # +2
```

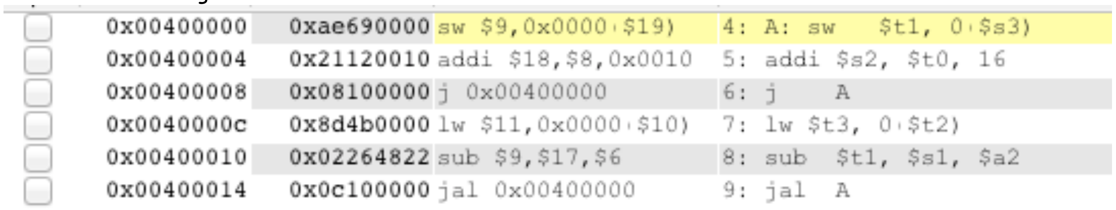
Label	Value	Address Field
Loop	0x0040_0000	0x010_0000
Exit	0x0040_0018	0x0002

2.40

(5 marks)

(c) With the aid of the MIPS Reference at the back of this booklet, convert the following assembly language instructions to the equivalent MIPS32 machine instructions expressed in hexadecimal, i.e. perform manual assembly:

```
sw   $t1, 0($s3)
addi $s2, $t0, 16
j    32
```

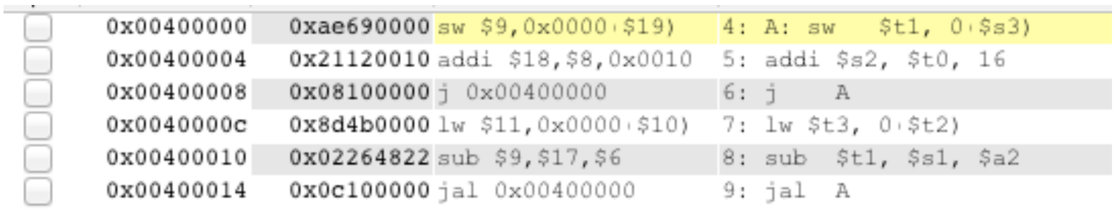


```
0x00400000   0xae690000 sw $9,0x0000($19)       4: A: sw    $t1, 0($s3)
0x00400004   0x21120010 addi $18,$8,0x0010      5: addi $s2, $t0, 16
0x00400008   0x08100000 j 0x00400000            6: j     A
0x0040000c   0x8d4b0000 lw $11,0x0000($10)      7: lw $t3, 0($t2)
0x00400010   0x02264822 sub $9,$17,$6           8: sub  $t1, $s1, $a2
0x00400014   0x0c100000 jal 0x00400000          9: jal   A
```

(5 marks)

(d) With the aid of the MIPS Reference at the back of this booklet, convert the following machine language instructions expressed in hexadecimal to the equivalent MIPS32 assembly instructions, i.e. perform manual disassembly:

```
8D4B0000
02264822
0C000010
```

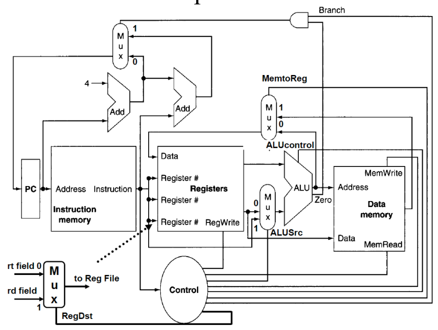


```
0x00400000   0xae690000 sw $9,0x0000($19)       4: A: sw    $t1, 0($s3)
0x00400004   0x21120010 addi $18,$8,0x0010      5: addi $s2, $t0, 16
0x00400008   0x08100000 j 0x00400000            6: j     A
0x0040000c   0x8d4b0000 lw $11,0x0000($10)      7: lw $t3, 0($t2)
0x00400010   0x02264822 sub $9,$17,$6           8: sub  $t1, $s1, $a2
0x00400014   0x0c100000 jal 0x00400000          9: jal   A
```

(5 marks)
(20 marks in total)

4.

(a) Provide a diagram showing the organization of a single-cycle MIPS processor. Include the main components and their inter-connection.



(5 marks)

(b) With the aid of a diagram, explain how the multi-cycle MIPS architecture can provide a speed up relative to the single-cycle MIPS architecture for the following program fragment:

```
          lw   $t1, 100($t0)
          add  $t2, $t1, $t2
          beq  $s0, $s1, Loop
          addi $t3, $t0, 1
```

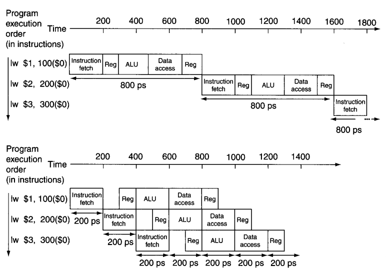
(5 marks)

Inst.	Inst. Mem.	Reg. Read	ALU	Data Mem.	Reg. Write	Total (ps)
R-type	200	50	100	▨	50	400
Load	200	50	100	200	50	600
Store	200	50	100	200	▨	550
Branch	200	50	100	▨	▨	350
Jump	200	▨	▨	▨	▨	200

(5 marks)

(c) With the aid of a diagram, explain how the pipelined MIPS architecture can provide a speed-up relative to both the single-cycle and multi-cycle MIPS architectures for the following program fragment:

```
          lw   $t1, 4($s0)
          lw   $t2, 8($s0)
          addi $t3, $t4, 16
          add  $t5, $t4, $t1
          addi $s0, $s0, 4
```



(5 marks)

(d) The following code fragment contains a Data Hazard when executed on a MIPS processor with a five-stage pipeline. Identity the hazards and propose workarounds:

```
calc:     lw       $t1, 0($s1)
          sub      $t3, $t1, $t0
          add      $t4, $t1, $t3
          lw       $t4, 0($s1)
          addi     $t3, $t4, 2
          sw       $t3, 0($s1)
          add      $t5, $s0, $s1
          addi     $t4, $t4, 2
```

Load-use $t1 and $t4
Move sub $t3, $t1, $t0 to between the $t4 load and use

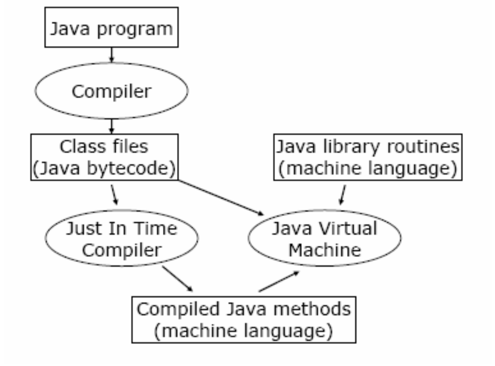
(5 marks)

(20 marks in total)

5.

(a) With the aid of a diagram, explain how a Java program is compiled and executed using a Java Virtual Machine.

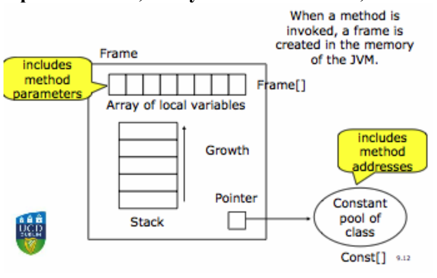
Java source code is compiled to bytecode.
• This instruction set is designed to be close to the Java language, so compilation is easy (i.e. very few optimizations).
• Java programs are distributed (e.g. downloaded) in bytecode format.
• An interpreter is used to execute Java bytecode. This simulates execution of the bytecode instruction set architecture (cf. SPIM).
• Java Virtual Machine: "The program that interprets Java bytecode."
• JVMs must be provided for every platform (combination of processor and operating system).


Java program
Compiler
Class files (Java bytecode)
Java library routines (machine language)
Just In Time Compiler
Java Virtual Machine
Compiled Java methods (machine language)

(5 marks)

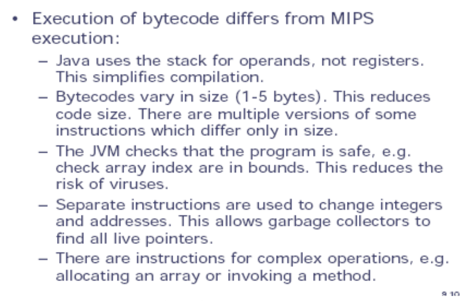
(b) What does a frame consist of within a Java Virtual Machine (JVM)? Provide a conceptual diagram of a JVM frame.

Operand stack, array of local variables, reference to the runtime pool of the class of the current method.


When a method is invoked, a frame is created in the memory of the JVM.
Frame
includes method parameters
Frame[]
Array of local variables
Growth
Stack
Pointer
includes method addresses
Constant pool of class
Const[]
9.12

(5 marks)

(c) Explain how Java bytecode execution differs from MIPS machine code execution. What are the reasons for these differences?



- Execution of bytecode differs from MIPS execution:
  - Java uses the stack for operands, not registers. This simplifies compilation.
  - Bytecodes vary in size (1-5 bytes). This reduces code size. There are multiple versions of some instructions which differ only in size.
  - The JVM checks that the program is safe, e.g. check array index are in bounds. This reduces the risk of viruses.
  - Separate instructions are used to change integers and addresses. This allows garbage collectors to find all live pointers.
  - There are instructions for complex operations, e.g. allocating an array or invoking a method.

Java bytecode design for compact code and secure execution. MIPS machine code designed for fast execution.

(5 marks)

(d) Determine the contents of the stack after execution of the following Java bytecode instructions. The contents of the stack and the array of local variables prior to execution of each instruction are given in Tables 2 and 3, respectively.

(i) `iadd`

2 3 14

(ii) `iconst_2`

2 3 10 6

(iii) `iload 1`

2 3 10 4 2

Table 2. Stack contents.

4
10
3
2

Table 3. Array contents.

Index	0	1	2	3	4
Contents	8	2	4	3	0

(5 marks)
(20 marks in total)

oOo

MIPS32 Reference

Machine Instruction Format

Instruction syntax	**Format**	**op**	**rs**	**rt**	**rd**	**shamt**	**funct**	**constant / address**
add *rd*,*rs*,*rt*	R	0	reg	reg	reg	0	32_{ten}	-
sub *rd*,*rs*,*rt*	R	0	reg	reg	reg	0	34_{ten}	-
addi *rt*,*rs*,*const*	I	8_{ten}	reg	reg	-	-	-	constant
lw *rt*,*addr*(*rs*)	I	35_{ten}	reg	reg	-	-	-	offset address
sw *rt*,*addr*(*rs*)	I	43_{ten}	reg	reg	-	-	-	offset address
j *addr*	J	2_{ten}	-	-	-	-	-	address
jal *addr*	J	3_{ten}	-	-	-	-	-	address

Formats

Type	**Fields**					
R	op	rs	rt	rd	shamt	funct
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
I	op	rs	rt	constant or address		
	6 bits	5 bits	5 bits	16 bits		
J	op	address				
	6 bits	26 bits				

Register List

Name	**Number**	**Usage**	**Preserved**
`$zero`	0	Constant	n.a.
`$v0 – $v1`	2-3	Result values	No
`$a0 – $a3`	4-7	Arguments	No
`$t0 – $t7`	8-15	Temporary	No
`$s0 – $s7`	16-23	Saved	Yes
`$t8 – $t9`	24-25	More temporary	No
`$gp`	28	Global pointer	Yes
`$sp`	29	Stack pointer	Yes
`$fp`	30	Frame pointer	Yes
`$ra`	31	Return address	Yes