The University of Nottingham

School of Computer Science

A Level 1 Module, Autumn Semester 2007-2008

Computer Systems Architecture (G51CSA)

Time Allowed: TWO Hours

Candidates must NOT start writing their answers until told to do so

Answer QUESTION ONE and THREE other questions

No calculators are permitted in this examination

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. **Subject-specific translation dictionaries are not permitted.**

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so

Question 1: (Compulsory) The following questions are multiple choice. There is at least one correct choice but there may be several. For each of the questions list all the roman numerals corresponding to correct answers but none of the incorrect ones.

Questions are marked as follows:

no errors	5 points
1 error	3 points
2 errors	1 point
≥3 errors	0 points

where an error is a correct answer left out, or an incorrect one included.

no error 5 points

1 error 2 points

2 or more errors 0 points

where an error is a correct answer left out or an incorrect one included.

a. Which of the following statements is correct?

(i) A word is always two bytes.

(ii) On MIPS an *aligned* half-word can only start at even addresses.

(iii) A Unicode character can be stored in 8 bits

(iv) On a little endian processor, the least significant byte of a word can be found at the same address as the word

(v) One byte can be represented by two hexadecimal digits

(5)

Solution

- *On MIPS an* aligned *half-word can only start at even addresses.*
- *On a little endian processor, the least significant byte of a word can be found at the same address as the word*
- *One byte can be represented by two hexadecimal digits*

b. What are the typical features of the *von Neumann architecture*?

(i) Programs cannot be modified while the processor is running

(ii) Data and programs share the same address space

(iii) The processor has a special register called the program counter (PC).

(iv) Each machine instruction contains the address of the next instruction

(v) Programs have to be written in assembly language.

(5)

Solution

- *The processor has a special register called the program counter (PC).*
- *Data and programs share the same address space*

c. Consider the following program fragment in MIPS32 assembly code:

```
li $s0, -1
srl $v0, $s0, 1
addiu $a0, $v0, 1
```

Which of the following statements are correct?

(i) The fragment will occupy 128 bits in memory

(ii) Register `$a0` will contain the largest positive representable signed number after executing the fragment

(iii) Register `$a0` will contain the least negative representable signed number after executing the fragment

(iv) The fragment will raise an overflow exception

(v) Register `$v0` will contain -1 after executing the fragment

(5)

Solution

- *The fragment will occupy 128 bits in memory*
- *Register `$a0` will contain the least negative representable signed number after executing the fragment*

d. Which of the following statements about 8-bit two's complement are correct?

(i) The most significant bit has a weight of -2^6

(ii) The representable range is -128 to $+127$

(iii) The representable range is -127 to $+128$

(iv) To calculate $-x$ from x, we flip all the bits and add 1

(v) To calculate x from $-x$, we flip all the bits and subtract 1

(5)

Solution

- *The representable range is* -128 *to* $+127$
- *To calculate* $-x$ *from* x*, we flip all the bits and add 1*

e. Which of the following statements about IEEE 754 single precision floating point numbers are correct?

(i) The exponent is represented in two's complement.

(ii) The word $\texttt{0000 0001}_{16}$ is the largest denormalised number

(iii) The word $\texttt{8000 0000}_{16}$ represents the number -0

(iv) $\infty - \infty = 0$

(v) The number $\texttt{1.111}_{10} \times 10^2$ can be represented exactly

(5)

Solution

- *The word* $\texttt{8000 0000}_{16}$ *represents the number* -0

Question 2

a. Implement a MIPS assembly fragment which, given the values of x and y in `$s0` and `$s1`, calculates the expression $x^3 + 6x^2y + 12xy^2 + 8y^3$, saving the result in `$s2`. *You don't have to check for overflows.*

Hint: use high-school algebra to make it easier. (8)

Solution *The hard way*

```
mul $t0, $s0, $s0  # $t0 = x^2
mul $t1, $s1, $s1  # $t1 = y^2

mul $a0, $t0, $s0  # $a0 = x^3

li $a0, 6
mul $a0, $a0, $t0
mul $a0, $a0, $s1  # $a0 = 6x^2y
add $s2, $s2, $a0  # $s2 = x^3 + 6x^2y

li $a0, 12
mul $a0, $a0, $s0
mul $a0, $a0, $t1  # $a0 = 12xy^2
add $s2, $s2, $a0  # $s2 = x^3 + 6x^2y + 12xy^2

mul $a0, $t1, $s1  # $a0 = y^3
sll $a0, $a0, 3    # $a0 = 8y^3
add $s2, $s2, $a0  # $s2 = x^3 + 6x^2y + 12xy^2 + 8y^3
```

Solution *The easy way: calculate* $(x + 2y)^3$

```
sll $t0, $s1, 1    # $t0 = 2y
add $t0, $t0, $s0  # $t0 = x + 2y
mul $s2, $t0, $t0  # $s2 = (x + 2y)^2
mul $s2, $s2, $t0  # $s2 = (x + 2y)^3
```

b. Consider the following program in MIPS32 assembler:

```
        .text
main:   li $v0,5
        syscall                 # read integer
        move $s0,$v0
start:  li $s1,0
```

```
        j test
loop:   andi $t0,$s0,1
        xor $s1,$s1,$t0
        srl $s0,$s0,1
test:   bne $s0,$zero,loop
out:    move $a0,$s1
        li $v0,1
        syscall                 # print integer
        li $v0,10
        syscall                 # exit
```

(i) Simulate the program by hand for the inputs: 15, 64, -1. In each case, how many times does the program loop, and what will it output? (6)

Solution *For* 15 *the program loops 4 times and then prints 0.*
For 64 *the program loops 6 times and then prints 1.*
For -1 *the program loops 32 times and then prints 0.*

(ii) Summarise the action the program performs in one sentence. (2)

Solution *The program prints 1 if the binary representation of the input contains an odd number of 1s and 0 otherwise.*

c. Implement *a loop* in MIPS assembly which given a positive integer n in register `$s0`, calculates the nth fibonacci number, i.e. the nth element in the series $1, 1, 2, 3, 8, 11 \ldots$ - each fibonacci number is the sum of the two previous ones.

For example, if `$s0` contains 5 then the code should store 8 in `$s1`. (9)

Solution *Three points for style: naming, layout, comments &c.; six points for functionality. Does it work for 0?*

```
    li $s1, 0
    li $t1, 1
    li $t0, 0
    j fib_cond
fib_loop:
        add $t2, $s1, $t1
        move $t1, $s1
        move $s1, $t2
        addi $t0, $t0, 1
```

```
fib_cond:
    blt $t0, $s0, sum_loop
```

Question 3

a. Calculate the 8-bit two's complement representation of the following decimal numbers:

$$a = 114 \qquad b = 17 \qquad c = -73 \tag{3}$$

Solution

$$\begin{array}{lrll} a = & 114_{10} = & \mathit{0111\,0010}_2 & \\ b = & 17_{10} = & \mathit{0001\,0001}_2 & \\ c = & -73_{10} = & -\mathit{0100\,1001}_2 & = \mathit{1011\,0111}_2 \end{array}$$

b. Showing details of your working, calculate using signed 8-bit two's complement the following sums:

$$a + b \qquad b + c \qquad a + c$$

Which case results in an overflow? (4)

Solution

$$\begin{array}{rcl} & \mathit{0\,1\,1\,1\,0\,0\,1\,0} & a = 114 \\ + & \mathit{0\,0\,0\,1\,0\,0\,0\,1} & b = 17 \\ \hline & \mathit{1\,0\,0\,0\,0\,0\,1\,1} & a + b = 131 \cong -125 \textit{ Overflow!} \end{array}$$

$$\begin{array}{rcl} & \mathit{0\,0\,0\,1\,0\,0\,0\,1} & b = 17 \\ + & \mathit{1\,0\,1\,1\,0\,1\,1\,1} & c = -73 \\ \hline & \mathit{1\,1\,0\,0\,1\,0\,0\,0} & b + c = -56 \end{array}$$

$$\begin{array}{rcl} & \mathit{0\,1\,1\,1\,0\,0\,1\,0} & a = 114 \\ + & \mathit{1\,0\,1\,1\,0\,1\,1\,1} & c = -73 \\ \hline & \mathit{0\,0\,1\,0\,1\,0\,0\,1} & b + c = 41 \end{array}$$

c. Calculate in detail using *signed* 8 bit arithmetic, the 16 bit result of $a \times b$ and $b \times c$. Give the final product in both decimal and binary. Take care with sign extension. (7)

Solution

$$\begin{array}{rcl} & \mathit{0\,1\,1\,1\,0\,0\,1\,0} & a = 114 \\ \times & \mathit{0\,0\,0\,1\,0\,0\,0\,1} & b = 17 \\ \hline & \mathit{0\,1\,1\,1\,0\,0\,1\,0} & \\ + & \mathit{0\,1\,1\,1\,0\,0\,1\,0} & \\ \hline & \mathit{0\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,1\,0\,0\,1\,0} & a \times b = 1938 \end{array}$$

$$\begin{array}{rcl} & \mathit{1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,0\,1\,1\,1} & c = -73 \\ \times & \mathit{0\,0\,0\,1\,0\,0\,0\,1} & b = 17 \\ \hline & \mathit{1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,0\,1\,1\,1} & \\ + & \mathit{1\,1\,1\,1\,1\,0\,1\,1\,0\,1\,1\,1} & \\ \hline \textit{Negate} & \mathit{1\,1\,1\,1\,1\,0\,1\,1\,0\,0\,1\,0\,0\,1\,1\,1} & b \times c = -1241 \\ \hline & \mathit{0\,0\,0\,0\,0\,1\,0\,0\,1\,1\,0\,1\,1\,0\,0\,1} & -b \times c = 1241 \end{array}$$

d. Calculate in detail using *unsigned* 8-bit arithmetic, the 8 bit quotient and remainder of $a \div b$. (4)

Solution

```
            0 0 0 0 0 1 1 0      a ÷ b = 6
 1 0 0 0 1 |0 1 1 1 0 0 1 0
   1 0 0 0 1                  0
     1 0 0 0 1                0
       1 0 0 0 1              0
         1 0 0 0 1            0
           1 0 0 0 1          0
         - 1 0 0 0 1          1
           0 0 1 0 1 1 1 0
         -   1 0 0 0 1        1
           0 0 0 0 1 1 0 0      Remainder: 12
                 1 0 0 1 0    0
```

e. If we perform

```
addu $s0, $s1, $s2
```

the MIPS processor will ignore overflows. Implement in MIPS32 assembly a test following the above instruction which jumps to a label named `overflow`, if an overflow has indeed occured. (7)

Solution

```
    xor $t0, $s1, $s2
    blt $t0, $zero, no_overflow  # sign($s1) !=
sign($s2)
    xor $t0, $s0, $s1
    blt $t0, $zero, overflow     # sign($s0) !=
sign($s1)
no_overflow:
```

continue normal processing

```
overflow:
```

error handler

Question 4

a. The following questions are about the binary representation of single-precision (32-bit) IEEE 754 floating-point numbers? (10 total)

(i) Show, using a diagram like the one below, how many bits are used for each of the *sign*, *exponent* and *mantissa* fields, and their ordering in a 32-bit word.

field	*field*	*field*
← x bits →	← y bits →	← z bits →

(3)

Solution

sign	exponent	mantissa
← *1 bit* →	← *8 bits* →	← *23 bits* →

(ii) How is the *exponent* encoded? How would you represent an exponent of 42? (2)

Solution *Exponent in excess-*127 *encoding. An exponent of* 42 *is represented by* $42 + 127 = 169 = \texttt{1010 1001}_2$.

(iii) In general, floating-point numbers are *normalised* before being encoded. What does this mean for the most significant bit of the mantissa? (1)

Solution *The MSB is always 1, so we do not need to store it explicitly for normalised numbers.*

(iv) Give the 32-bit IEEE 754 representation of the following floating point numbers as 8 hexadecimal digits: -1.125, 385 (4)

Solution $1.125 = \texttt{1.0010...}_2 \times 2^0$
Exponent: $0 + 127 = \texttt{0111 1111}_2$;
Drop MSB of mantissa: $\texttt{0010...}_2$
So $-1.125 = \texttt{1 01111111 001}_2$... *and 20 trailing* 0*s...*
$= \texttt{BF900000}_{16}$

Solution $385 = \texttt{1 1000 0001}_2 = \texttt{1.1000 0001...}_2 \times 2^8$
Exponent: $8 + 127 = \texttt{1000 0111}_2$
Drop MSB of mantissa: $\texttt{1000 0001...}_2$
So
$385 = \texttt{0 10000111 10000001000}_2$... *and 12 trailing* 0*s...*
$= \texttt{43C08000}_{16}$

b. Suppose there exists a 12-bit IEEE 754 floating point format, with 1 sign bit, 6 exponent bits, and 5 mantissa bits. (15 total)

(i) How would $-\infty$ be represented in this 12-bit format? And the smallest positive *normalised* number? Give the value in

decimal of the second number, and show both either as 12 bits or as 3 hexadecimal digits. (4)

Solution

Value	*Binary*	*Hex*
$-\infty$	$\mathit{1\,111111\,00000}_2$	$\mathit{FE0}_{16}$
2^{-30}	$\mathit{0\,000001\,00000}_2$	$\mathit{020}_{16}$

Two points for each representation; one point for 2^{-30}.

(ii) Give the *nearest* representation n of 5.612 in this format. (3)

Solution

0.612	.
~~1~~.224	1
0.448	0
0.896	0
~~1~~.692	1
~~1~~.284	1

$\mathit{5.612}_{10} = \mathit{101.10011...}_2$
$\approx \mathit{101.101}_2$
$= \mathit{1.01101}_2 \times 2^2$

*Excess-*31 *exponent is* $\mathit{100001}_2$, *so* $\mathit{0\,100001\,01101}_2$.
Drop MSB of mantissa: $\mathit{01101}_2$
One mark for evidence of working out, one for rounding to the nearest answer and one for giving the full 12 bits.

(iii) What is the *actual value* of n? Hence, work out its relative error r, to 3 significant digits.
You may use the fact that $a/5.612 \approx a \times 0.1782$. (2)

Solution $\mathit{101.101}_2 = 5 + 1/2 + 1/8 = 5.625$
Relative error

$$\begin{aligned} r &= (5.625 - 5.612)/5.612 \approx 0.013 \times 0.1782 \\ &= 13 \times 10^{-3} \times 1782 \times 10^{-4} \\ &= 23166 \times 10^{-7} \\ &\approx 2.32 \times 10^{-3} \end{aligned}$$

(iv) Calculate n^2 using binary floating point multiplication. Show rounding, normalisation and where you might check for overflow. Give the result as a 12-bit IEEE 754 number. (6)

Solution

$$\begin{array}{rl} & \mathit{1.0\ 1\ 1\ 0\ 1} \times 2^2 \\ \times & \mathit{1.0\ 1\ 1\ 0\ 1} \times 2^2 \\ \hline & \mathit{1.0\ 1\ 1\ 0\ 1} \\ & \quad \mathit{1\ 0\ 1\ 1\ 0\ 1} \\ & \quad\quad \mathit{1\ 0\ 1\ 1\ 0\ 1} \\ + & \quad\quad\quad\quad \mathit{1\ 0\ 1\ 1\ 0\ 1} \\ \hline \textit{Round} & \mathit{1.1\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1} \times 2^4 \\ \hline \textit{Result} & \mathit{1.1\ 1\ 1\ 1\ 1} \times 2^4 \end{array}$$

*No normalisation necessary; 6-bit excess-*31 *exponent can accommodate* $4 = \mathit{100011}_2$, *so no overflow.*
Result: $\mathit{0\,100011\,11111}_2 = \mathit{11111.1}_2 = 31.5_{10}$

Question 5

a. Following standard MIPS procedure calling conventions, implement a procedure `strrchr` (*str*ing *r*eversed *cha*racter) which receives the starting address of a NUL-terminated string in `$a0`, an ASCII character code in `$a1`, and returns the position of the *last* occurrence of the character in the string. If the character is NUL, you are to return the position of the terminating byte. If the character cannot be found, return -1. The first character of the string is position 0. (13)

Solution

```
strrchr(str, chr)
r = -1;
i = 0;
while(true)
if(str[i] == chr)
r = i;
if(str[i] == '\0')
break;
i++;
return r;
```

```
strrchr:  # $a0: haystack starting address, $a1: needle
    li $v0, -1
    move $t0, $zero
    j strrchr_load
strrchr_loop:
        addi $t0, $t0, 1
strrchr_load:
        lbu $t0, 0($a0)
        addi $a0, $a0, 1
        bne $t0, $a1, strrchr_skip
            move $v0, $t0
strrchr_skip:
        bne $t0, $zero, strrchr_loop
    j $ra
```

b. The following program reads in a string from the user and prints out a number, but is missing some code in the middle:

```
        .data
buffer: .space 1024
        .text
        .globl main
main:   la $a0, buffer
        li $a1, 1024  # size of buffer
        li $v0, 8     # read_string
        syscall

        # insert your code here

        li $v0, 1
        syscall         # print_int/$a0
        li $v0, 10
        syscall         # exit
```

Provide the missing code in the above program, calling the `strrchr` function from the previous part, such that it calculates and prints the *length* of the user input to the console. (4)

Solution

```
la $a0, buffer
move $a1, $zero
jal strrchr
move $a0, $v0
```

c. Describe the procedure calling conventions for MIPS32: (8 total)

(i) How are arguments passed? (2)

Solution *First four in registers `$a0`–`$a3`, with the remaining pushed onto the stack.*

(ii) How are results returned? (1)

Solution *In registers `$v0` and `$v1`.*

(iii) Which of the user registers (ignoring `$at`, `$k0` and `$k1`) have to be saved by the caller? (Assuming the caller needs them preserved.) Which ones are always saved by the callee? (4)

Solution *Caller: `$t0`–`$t9`, `$a0`–`$a3`, `$v0` and `$v1`.*
Callee: `$s0`–`$s7`, `$sp`, `$fp` and `$ra`.

(iv) Where are local variables stored? (1)

Solution *On the stack.*

Question 6

a. How does the technique of pipelining increase performance? Explain the increased instruction throughput, compared with a multi-cycle non-pipelined processor. Does pipelining reduce the execution time for individual instructions? Why? (4)

Solution *Pipelining splits the execution of instructions into several simpler stages, each of which is guaranteed to complete within one clock cycle.*

Instructions are fed through the pipeline overlapped, increasing the total throughput.

Each instruction will still take several cycles to pass through the pipeline completely, so the individual instruction execution time is not reduced.

b. Below is an idealised schematic for the 5-stage MIPS pipeline.

What do each of the abbreviations IF/ID/EX/MEM/WB stand for? Give a short description of what happens at each stage. (5)

Solution

IF	*Instruction Fetch*	*machine code retrieved from memory*
ID	*Instruction Decode*	*set control signals, load source registers*
EX	*Execute*	*Perform ALU operation*
MEM	*Memory*	*Read/write from/to memory*
WB	*Write-Back*	*Result written back into register file*

c. Give four features of MIPS instruction set architecture which make it especially suitable for a pipelined implementation. (3)

Solution

- *Fixed width 32-bit instructions simplifies instruction fetch*
- *Only three instruction formats; with operand fields in the same positions, operands can be fetched while the instruction is being decoded – no need for a separate register read stage*
- *Memory access only with load/store instructions, which means the register + offset address can be calculated during the execute stage by the ALU – avoids structural hazards*

- *Multi-cycle instructions such as* `mul` *write to dedicated registers, so it doesn't need to finish in the one cycle imposed by the EX stage*
- *Memory accesses must be aligned, so the data for each load/store can be transferred in one cycle*
- *(Not taught) Branch delay slot / delayed branches compensates for the cost of flushing the pipeline*

d. What is a data hazard? Name and describe a hardware technique that overcomes most data hazards. What happens during the execution of the following sequence?

```
lw $t0, 0($a0)
add $s0, $s0, $t0
```
(4)

Solution *Exercise 8.*

e. What is a control hazard? How can we avoid stalling the pipeline some of the time? What happens if this technique goes wrong?(4)

Solution *Exercise 8*

f. Identify the data hazards in the following code fragment and re-order the instructions to avoid any pipeline stalls.

```
lw $t0, ($sp)
lw $t1, ($t0)
addi $t1, $t1, 4
lw $t2, ($t0)
add $t3, $t2, $t1
sw $t3, 4($sp)
sw $zero, ($sp)
```
(5)

Solution *Exercise 8*