UNIVERSIDAD CARLOS III DE MADRID. DEPARTAMENTO DE INFORMÁTICA
Bachelor in Computer Science and Engineering.
ESTRUCTURA DE COMPUTADORES
Janyary, 2019 **Final Exam**

2:30 hours are available for this exam.
No books, notes or calculators (or electronic devices) of any kind may be used. .

Exercisxe 1 (1 point). Indicate the decimal value of the largest non-standardized number that can be represented using the IEEE 754 single precision standard. Also indicate the decimal value of the next (larger than the largest non-standardized) number that can be represented. What is the difference between these two consecutive numbers that can be represented? Is this difference kept constant for all numbers that can be represented in this standard?

Exercise 2 (3 points). Consider the `Accounting` routine. This routine accepts five input parameters:

- The starting address of an matrix (stored by rows) of numbers of type int, of dimension MxN
- The number of rows in the matrix (M).
- The number of columns in the matrix (N).
- A number of row i.
- A column number j.

The function returns two values. The first is the value of the element stored in the position (i, j) of the array. In the second value is returned the number of elements of the array whose value is equal to the one located in the position (i,j). For this second value also has to be taken into account the element located in the position (i,j). That is, the second value returned by the function will always have a value greater than or equal to 1.

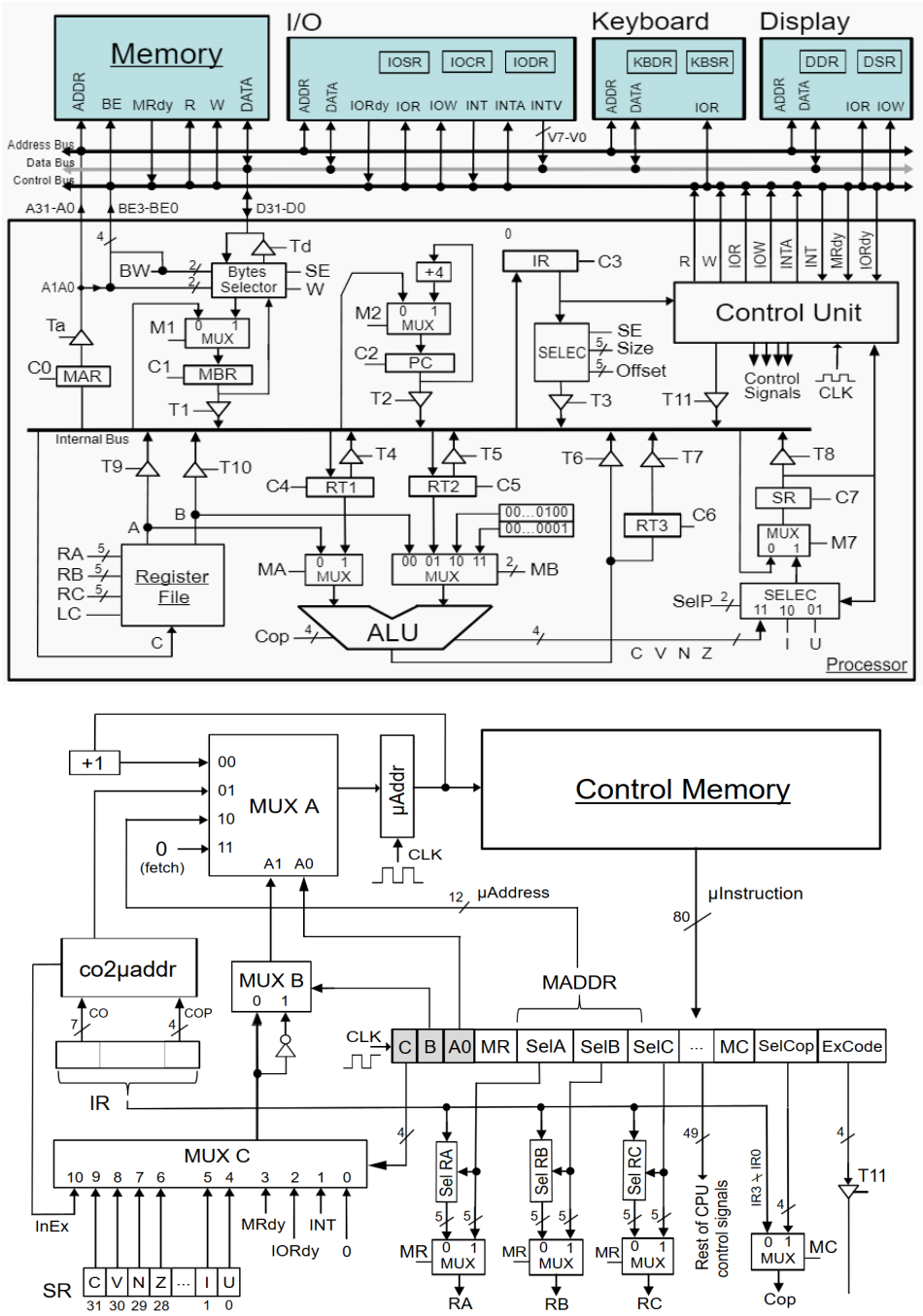
a) Correctly code the routine described above. You must strictly follow the MIPS parameter passage agreement seen in class. It will be positively valued that this function invokes other functions.

b) Given the following data segment:

```
.data
  A: .word      8, 4, 3, 3, 5,
                2, 3, 0, 4, 5,
                0, 0 ,1, 2, 3
  M: .word 3
  N: .word 5
```

Write the fragment that allows you to correctly invoke the function for the A matrix of the MxN dimension and the values of i=2 and j=3. Then print the two values returned by the function

Exercise 3 (3 points). Given the WepSim structure:



Specify the elementary operations and control signals needed to execute the `ADDM (Ra) (Rb) address` machine instruction (include the fetch cycle). This instruction adds the contents of the memory locations stored in the Ra and Rb registers. The result is stored in the memory `address`. The previous instruction occupies two words and the operation code 6 bits. The fields for the registers Ra and Rb are coded after the operation code. The address field is coded in the second word of the instruction.

NOTE: Assume that R29 acts as a stack pointer, that the stack pointer points to the top of the stack, and that the stack grows in decreasing memory directions. Also consider that register R0 is read-only and stores the value 0.

Exercise 4 (1 point). On the WepSim processor you want to execute a set of instructions that includes among others the following:

li R, valor (similar a la pseudoinstrucción li del MIPS)

000111	R	sin uso	valor
6 bits	5 bits	5 bits	16 bits

lw R, dirección (similar a la instrucción lw del MIPS)

001000	R	sin uso	dirección
6 bits	5 bits	5 bits	16 bits

lw Rd, (Rf) (similar a la instrucción lw del MIPS)

001001	Rd	Rf	sin uso
6 bits	5 bits	5 bits	16 bits

add R1, R2, R3 (similar a la instrucción lw del MIPS)

001010	R1	R2	R3	sin uso
6 bits	5 bits	5 bits	5 bits	16 bits

Consider the following portion of WepSim's main memory:

0x00000A00	0x1C600003
0x00000A04	0x24E30000
0x00000A08	0x28221800
0x00000A0C	

If the value of the program counter is 0x00000A04, indicate the values of the registers PC, MAR, MBR and IR once the instruction stored in the previous address is executed. Also indicate which assembly instruction (operation code and operands) corresponds to the machine instruction stored in the memory position 0x00000A04

Exercise 5 (2 points). Consider the following code:

```
.data:
            v1: .space 1024
            v2: .space 4096
.text:
            li   $t0, 0
            li   $t1, 0
            li   $t2, 0
            li   $t3, 1024
loop  :     bge  $t0,  $t3, end
            lb   $t4, v1($t1)
            lw   $t5, v2($t2)
            add  $t4, $t4, $t5
            sw   $t4, v2($t2)
            addi $t0, $t0, 1
            addi $t1, $t1, 1
            addi $t2, $t2, 4
            b    loop
end:
```

This code is executed in a 32-bit word width architecture, which includes a fully associative 64KB instruction cache and a 128KB data cache, associated by 8-way sets. Both caches have 64 byte lines. Assuming the cache is initially empty and the program starts at memory address 0x000000.

a) Indicate the number of lines and sets of the data cache memory.
b) Indicate the hit ratio produced by the execution of the previous fragment in the instruction cache.
c) Indicate the hit ratio that produces the execution of the previous fragment in the data cache.
d) If the data cache is changed to a fully associative one, what impact would it have on the calculated hit ratio and on the overall performance of the cache. Justify your answer.

Solutions

Exercise 1.

- The largest non-normalized number using the IEEE 754 standard of simple precision is the one that has all zeros in the exponent and the largest mantissa. That is to say:

  `0 00000000 11111111111111111111111`

  The value is: $0,11111111111111111111111_{(2} \times 2^{-126} = (1-2^{-23}) \times 2^{-126}$ in decimal.

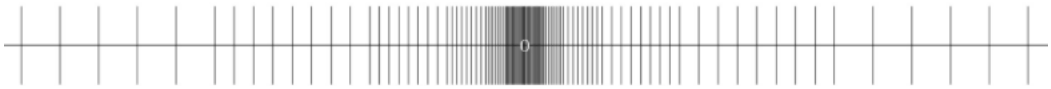
- The next representable number is the normalized number:

  `0 00000001 00000000000000000000000`

  The decimal value is $1,0 \times 2^{-126}$.

- As you can see the difference between both consecutive numbers is, quantitatively from $(2^{-23}) \times 2^{-126}$ nd qualitatively of being one the lesser standardized representable and the other the greater non-normalized representable.

- This difference does not remain constant for all representable numbers in this standard because the density of representable numbers decreases as we move away from 0.



Exercise 2.

a) his function receives 5 arguments, the first four are passed in $a0, $a1, $a2 and $a3 and the fifth is passed in the stack. The function returns two values, the first one in $v0 and the second one in $v1.

```
Counting   :    addi  $sp, $sp, -20
                sw    $a0, 0($sp)
                sw    $a1, 4($sp)
                sw    $a2, 8($sp)
                sw    $a3, 12($sp)
                sw    $ra, 16($sp)

                ; ObtainNumber function returns
                ; the element stiored in the position i, j.
                ; Receives the same 5 arguments
                ; you have to put the fifth one back in the pile
                lw    $t0, 24($sp)
                addi  $sp, $sp, -4
                sw    $t0, ($sp)
                jal   ObtainNumber

                addi  $sp, $sp, 4
                move  $v1, $v0

                lw    $a0,  ($sp)
                lw    $a1,  4($sp)
                lw    $a2,  8($sp)
                move  $a3, $v1

                jal CountElements     ; return the number of
                                      ; elements equal to the stored
                                      ; in (i,j) that is in $v1

                move  $t1, $v1
                move  $v1, $v0
                move  $v0, $t1

                lw    $a0, ($sp)
                lw    $a1, 4($sp)
                lw    $a2, 8($sp)
                lw    $a3, 12($sp)
                lw    $ra, 16($sp)

                addi  $sp, $sp, 20
                jr    $ra
```

```
ObtainNumber:    ; Obtain the memory address of (i,j)
                 ; Accept the 5 same arguments

                 lw   $t0, ($sp) ; fifth element on stack


                 li   $t1, 4
                 mul  $t2, $t1, $t0
                 mul  $t3, $a2, $t1
                 mul  $t3, $t3, $a3
                 add  $t3, $t3, $a0
                 add  $t3, $t3, $t2
                 lw   $v0, ($t3)
                 jr   $ra

CountElements:   li   $t0, 0
                 li   $v0, 0
                 mul  $t1, $a1, $a2 ; number of element in matrix
      loop  :    bge  $t0, $t1, end
                 lw   $t2, ($a0)
                 bneq $t2, $a3, notequal
                 addi $v0, $v0, 1
      notequal:  addi $t0, $t0, 1
                 addi $a0, $a0, 4
                 b    loop
           end:  jr   $ra
```

b) To invoke the function we need the following code:

```
la   $a0, A
lw   $a1, M
lw   $a2, N
li   $a3, 2
li   $t0, 3
addi $sp, $sp, -4
sw   $t0, ($sp)
jal  Counting
addi $sp, $sp, 4

move $a0, $v0
li   $v0, 1
syscall

move $a0, $v1
syscall
```

Exercise 3.

The Ra register is coded in bits 25-21 of the first word of the instruction and the Rb register is coded in bits 20-16 of that word.

- *Fetch*:

Cycle	Elementary operations	Control signals
C1	MAR ← PC	T2, C0
C2	MBR ←MP[MAR] PC ← PC+4	Ta, R, BW=11, M1=1, C1, M2=1, C2
C3	IR ← MBR	T1, C3
C4	Decoding and jumpo to operation code	C=0, B=0, A0=1

- Code of the requested microprogram. Since the instruction is arithmetic, it is considered that the state register is modified

Cycle	Elementary operations	Control signals
C1	MAR ← Ra	C0, T11, MR=0, SelA=**21** (10101)
C2	MBR ← MP[MAR]	Ta, R, BW=11, M1=1, C1
C3	RT1 ← MBR	T1, C4
C4	MAR ← Rb	C0, T11, MR=0, SelA=**16** (10000)
C5	MBR ← MP[MAR]	Ta, R, BW=11, M1=1, C1
C6	RT2 ← MBR	T1, C5
C7	RT3 ← RT1 + RT2 Update SR	MA=1, MB=1, MC=1, Cop=+, C6 SelP=11, M7, C7
C8	MAR ← PC	T2, C0
C9	MBR ←MP[MAR] PC ← PC+4	Ta, R, BW=11, M1=1, C1, M2=1, C2
C10	MAR ← MBR	T1, C0
C11	MBR ← RT3	T7, C1
C12	MP[MAR] ← MBR	Ta, Td, W, BW=11
C13	Jump to fetch	C=0, B=1, A0=0, MADDR=fetch

Exercise 4.

- The instruction is 0x24E30000 that in binary is:

  0010 0100 1110 0011 0000 0000 0000 0000

  The first 6 bits are the operation code: 001001 which corresponds to the lw Rd, (Rf)

  The field $Rd is 00111 (R7)
  The field $Rf is 00011 (R3)

  Therefore, assembly instruction is lw R7, (R3)

- Once the instruction is executed:

  PC = 0x00000A08
  MAR = <content of R3>
  MBR = < Content of the memory address indicated in R3 and found in MAR >
  IR = 0x24E30000

Ejercicio 5.

a) The size cache is 128KB = 2^{17} bytes. The lines have 64 bytes.
Number of lines: $2^{17} / 2^6 = 2^{11} = 2048$.
Number of sets: $2^{11} / 2^3 = 2^8 = 256$

b) Each line of the instruction cache has 64 bytes and can store 16 machine instructions. All the instructions of the MIPS occupy a word.

When the first instruction is accessed, a failure occurs and a block of 16 machine instructions (64 bytes) is brought into the instruction cache memory. This block includes all the instructions of the statement code fragment, so no more failures are produced.

The number of accesses to instructions that occur in the execution of the previous fragment is:

4 + 1024 x 9 + 1 = 4 + 9216 + 1 = 9221.

Therefore, the hit ratio is: (9221 – 1) / 9221

c) In each loop turn, the following data accesses are produced:
- A single byte reading of a byte vector.
- A 4-byte reading of an integer vector.
- A write of 4 bytes in a vector of integers.

Every 64 iterations occur:
- A miss in v1.
- Four missses in v2.
- The number of memory accesses is 64 x 3 = 192

As the total number of iterations (1024) is a multiple of 64, the ratio is maintained and therefore the success rate is:
(192-5) / 192 = 187 / 192 = (3*1024) – 5*(1024/64) / (3*1024)

c) Each set of 8 lines supposes 8*64 = 512 bytes, so v1 needs 2 sets and v2 needs 8 sets, without overlapping between those sets (having 256 different sets). Moreover, data are not reused. Therefore, there are no ejections and the behavior is similar to a totally associative cache (in terms of number of failures). An associative cache would require more bits for the tag. If the search of the tags is done in parallel, the performance would be similar, but a complex circuitry would be needed to examine in parallel the tags of all the lines of the cache. If the search is not done in parallel, the performance would be worse.