UNIVERSIDAD CARLOS III DE MADRID. DEPARTAMENTO DE INFORMÁTICA
Bachelor in Computer Science and Engineering.
Computer Structure
10 de enero de 2020 **Examen final**

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

Exercise 1 (1 point). Considering the single precisión IEEE 754 standard::

a) Encode in this standard the decimal value -17.25 Express the final result in hexadecimal.
b) Indicate the difference (express this difference as a decimal value) between the smallest positive normalized value that can be represented in this standard and the largest positive non-normalized value that can be represented in this standard.

Exercise 2 (3 points). Consider the ADD routine. This routine accepts five input parameters (they are passed in the same order as described below):

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

If the row value is 0 the function adds up the values of all the elements in column i. If the row value is not 0 the function sums the values of all the elements located in row i. The function returns two values

- If the argument i is out of range depending on the value of the row argument the function will return -1 (indicating the error) as the first result and 0 as the second result.
- If the argument i is correct (it is inside the range of the row or column, depending on the value that the row argument has) it returns 0 as the first result and the value of the sum of the row or column i as the second result: if row is equal to 0 it returns the sum of the column i; if row is different from 0 it returns the sum of the row i as the second result.

State:

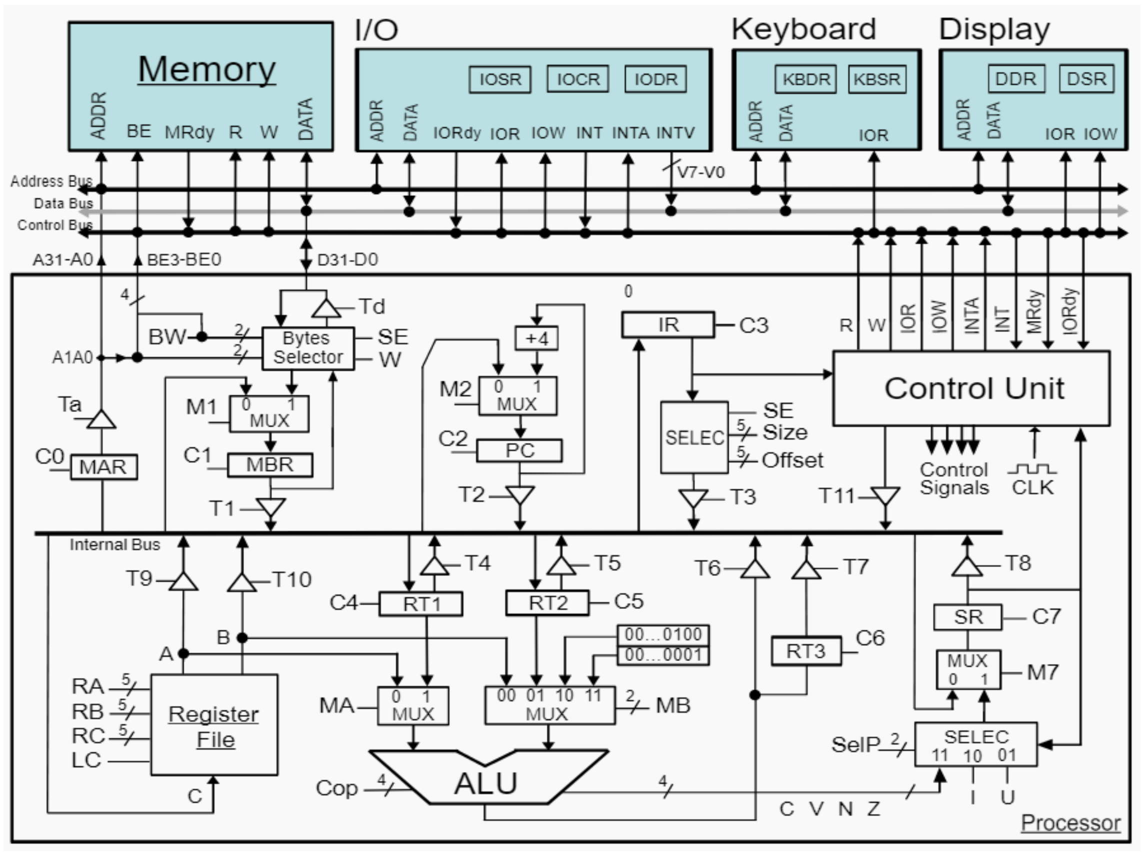
a) (2.5 points) Correctly code the Sum routine described above. You must strictly follow the MIPS parameter passage agreement seen in the course.

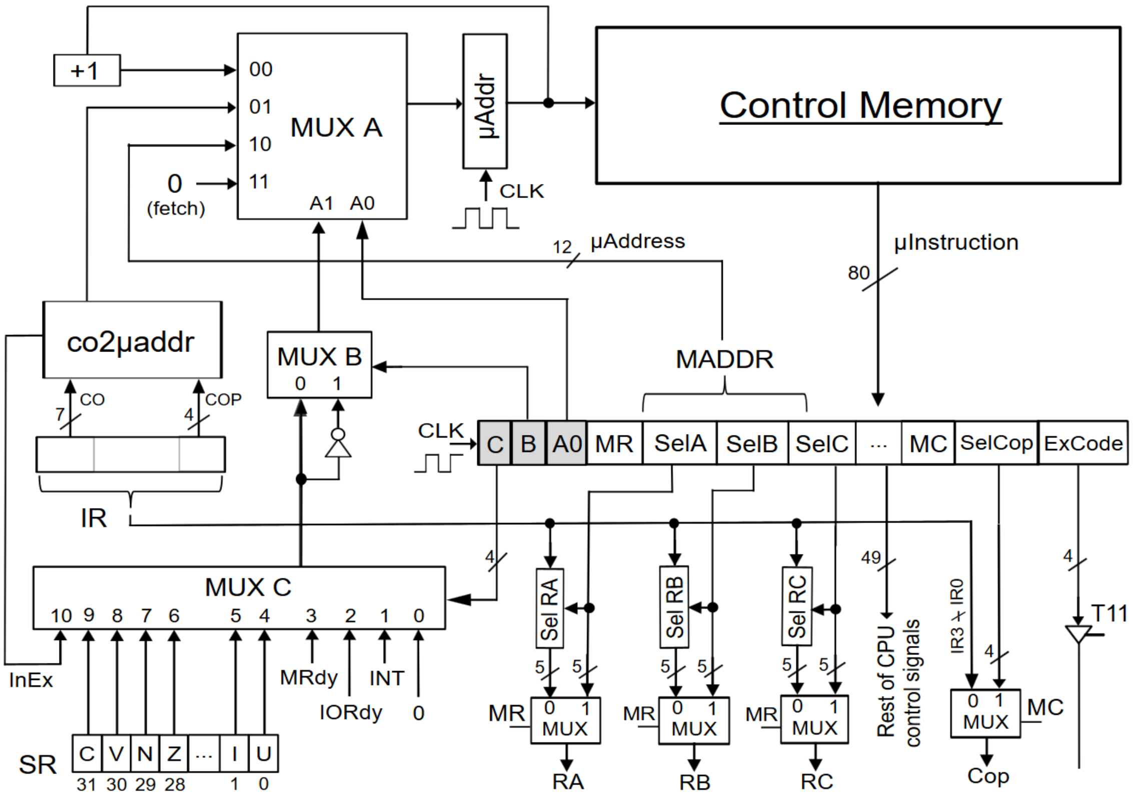
b) (0.5 points) Given the following data segment:

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

Encode the code fragment that allows to invoke correctly the function Sumar for the matrix A of dimension MxN and the values of row=1 and i=2. Then print the two values returned by the function.

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





State:

a) (0.5 points) Indicate a valid format, justifying your answer, for the following instruction: `addm Rd, n, address`. This instruction adds the integer value n to the content of the memory position `address` and leaves the result in the specified register Rd. The instruction must be able to specify any memory position in the processor and represent any integer value.
b) (1.25 points) Specify the elementary operations (including fetch) required to execute the instruction `lw Rd, address`. This instruction stores in the Rd register the value stored in the address memory location. The instruction occupies two words. In the first word is coded the operation code and the register, and in the second one the `address` field.
c) (1.25 points) Specify all the control signals required to execute the following four elementary operations. Indicate the solution on a separate sheet, using a structure similar to that shown below:

Reg ← RT2 (the Reg field is in the instruction between bits 25 and 21). After that, the next microinstruction is followed.	
RT1 ← regOr1 + RT2 (The regOr1 field is located in the instruction between bits 12 and 8. This operation updates the status register. After that, the next microinstruction is followed.	
If (SR(Z) ≠ 0) jumps to the micro address found on the label used in WepSim `jump`. Otherwise it continues with the next micro instruction.	
MBR ← MP[MAR] (reading of a word). Then follow with the following micro-instruction.	

Exercise 4 (1,5 points).

Consider a 32-bit computer with 64 registers that addresses the memory by bytes. Answer the following questions:

a) How much memory (expressed in MB) can this computer address?
b) How many bits are used to identify the registers in this computer?
c) What is the program counter register and what is it for?
d) Indicate the actions performed by the control unit in the cycle of recognition of an interruption

Exercise 5 (1,5 points).

Consider a computer with a 32-bit word width, which includes a 128KB data cache, associated by 8-way sets and 64-byte lines. Consider that in this computer the following program fragment is executed:

```
double     A[1024];
float      B[1024];
int        C[1024];

for (i=0; i < 1024; i++)
     C[i] = (int)A[i] + (int)B[i];
```

a) (0.25 points) Indicate the number of lines and sets of the data cache memory.
b) (1 point) Indicate the hit rate produced by the execution of the previous fragment in the data cache, assuming that the data cache is initially empty.
c) (0.25 points) If the data cache is changed to a fully associative one, what impact would it have on the calculated hit rate and on the overall cache performance? Justify your answer

Solutions

Exercise 1.

a) $-17{,}25_{(10} = -10001{,}01_{(2} = -1{,}000101 \times 2^4$ (normalized). The singo bit is 1 (the number is negative). The value that is stored in the exponent is 4 + 127 = 131 = 10000011. The mantissa that is stored (not the implicit bit) is 0001010000…..0000.

The representation is 1100 0001 1000 1010 0….000000 = 0xC18A0000

b) The smallest normalized positive number is the one with the smallest mantissa (all zeros) and the smallest exponent (1). Its representation in the IEEE 754 standard is the following:
0 00000001 000000000000

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

On the other hand, the largest non-standardized number corresponds to a representation where the exponent is 0 and the mantissa is the largest (all 1). Its representation is:

0 00000000 11111…1111111

The decimal value is $(1-2^{-23}) \times 2^{-126} = 2^{-126} - 2^{-149}$

Between these two numbers no other number can be represented in this standard, they are consecutive numbers.

The difference between them is $2^{-126} – (2^{-126} - 2^{-149}) = 2^{-149}$

Exercise 2.

a) This 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

```
ADD   :     lw    $t0, ($sp)        # stack Access, argument i

            beqz  $a3, column       # rwo = 0 -> add column i

row:        # in this case row <> 0 -> add row i
            blt   $t0, $0, error    # i < 0
            bge   $t0, $a1, error   # i >= M

            # the starting address of the row i is calculated
            muli  $t1, $a2, 4       #  N x 4
            muli  $t1, $t1, $t0     # (N x 4) x i
            add   $t1, $a0, $t1     #  $t1: starting address of row i
            move  $t2, $a2          # number of elements in row i to be added
            move  $t3, 4            # distance between elements in the row
            b addElements

column:     # in this case row = 0 -> add colum i
            blt   $t0, $0, error    # i < 0
            bge   $t0, $a2, error   # i>= N
            # the starting address of column i is calculated

            muli  $t1, $t0, 4       #  i x 4
            add   $t1, $a0, $t1     # $t1: starting address of column i
            move  $t2, $a1          # number of elements of the col. i to add
            muli  $t3, $a2, 4       # distance between column elements
                                    # 4 x N


addElements:      # the elements in the row or column are added
            # $t1: starting address of the first element
            # $t2: number of elements to be added
            # $t3: distance between one element and the next. In the case of
            #      adding one row the distance is 4 (they are int type
            #     elements).
            #      If a column is added, the distance is 4xN (you must
            #      skip N integer-type elements.

            li    $t4, 0      # index used to browse the row or column.
            li    $v1, 0      # $v1 stores the value to add
looop:      bge   $t4, $t2, end
            lw    $t5, ($t1)
            add   $v1, $v1, $t5
            addi  $t4, $t4, 1
            add   $t1, $t1, $t3     # next element
            b     loop

end:        li    $v0, 0
            jr    $ra
error:      li    $v0, -1
            li    $v1, 0
            jr    $ra
```

a) To invoke the above function we need the following code::

```
la    $a0, A
lw    $a1, M
lw    $a2, N
li    $a3, 1

# The value of i (i=2) is passed in stack
addiu $sp, $sp, -4
li    $t0, 2
sw    $t0, ($sp)

jal   ADXD

# Restore the stack
addiu $sp, $sp, 4

# Print the first value
move  $a0, $v0
li    $v0, 1
syscall


# Print the second value
move  $a0, $v1
syscall
```

Exercise 3.

a) The WepSim processor is a 32-bit processor and according to the scheme of the control unit uses 7 bits for the operation code. According to the statement the value n must be able to specify any integer and the address field must be able to specify any address. Therefore, each of these fields must occupy 32 bits (the word width of the computer). The operation code occupies 7 bits according to the scheme shown for the control unit. As for the Rd field, it is a register and 5 bits are needed for coding. This instruction needs, therefore, three words: in the first one it is codified the operation code (bits 31 to 25) and the register Rd (bits 24 to 20), in the second one in field n and in the third one the address field.

b) The instruction `lw Rd dirección` uses two words

- *Fetch*:

Cycle	Elmentary Operations
C0	MAR ← PC

C1	MBR ←MP[MAR]
	PC ← PC+4

C2	IR ← MBR
C3	Decoding and jump to operation ode

- Elementary operation for the execution:

Cycle	Elmentary Operations
C0	MAR ← PC

C1	MBR ←MP[MAR]
	PC ← PC+4

C2	MAR ← MBR
C3	MBR ←MP[MAR]
C4	Rd ← MBR

c) A continuación se indican las señales de control necesarias para estas operaciones elementales:

Reg ← RT2 (the Reg field is in the instruction between bits 25 and 21). After that, the next microinstruction is followed.	T5, SelC= 10101, MR = 0, LC, A0=0, B = 0, C = 0
RT1 ← regOr1 + RT2 (The regOr1 field is located in the instruction between bits 12 and 8. This operation updates the status register. After that, the next microinstruction is followed.	SelA=01000, MR = 0, MB = 01, Cop = 1010 (ADD), T6, C4, SelP = 11, M7, C7, A0=0, B = 0, C = 0
If (SR(Z) ≠ 0) jumps to the micro address found on the label used in WepSim jump. Otherwise it continues with the next micro instruction.	A0=0, C = 0111, B = 0, MADDR = jump
MBR ← MP[MAR] (reading of a word). Then follow with the following micro-instruction.	Ta, BW = 11, R, M1, C1, A0=0, B=, C=0

Exercise 4.

a) Since the computer has 32 bits, at most it can address 2^{32} bytes = $2^{32} / 2^{20} = 2^{12}$ MB.
b) We need $\log_2 64 = 6$ bits.
c) The program counter is a processor control register that stores the address of the next instruction to be executed. It is used to access in memory the instructions to be executed in the processor.
d) The control unit performs the following actions:
   - Check if a interrupt signal is activated.
   - If activated:
     - Saves the program counter and status register. They are saved in the stack or in special registers of the processor.
     - Switches from user mode to kernel mode (modifying the status register).
     - Gets the address of the interrupt treatment routine.
     - Stores in the program counter the obtained address (in this way the next instruction will be the one of the interrupt treatment routine).

Exercixe 5.

a) TYhe size of the cache is 128 KB = 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) In each iteration of the loop the following data accesses occur:
   - A reading of a double type element (vector C). This element occupies 8 bytes according to the IEEE 754 standard of double precision.
   - A reading of a float type element (vector B). This element occupies 4 bytes according to the IEEE 754 standard of simple precision.
   - A write of an int type element (vector A). This element occupies 4 bytes in a 32-bit computer as the statement.

In a cache line (64 bytes) fit 16 elements of type int and type float and 8 of type double.

The access pattern is as follows:

i=0: Reading of C [0]. A miss occurs. A line is cached: C[0]...C[7].
Reading of B[0]. A miss is produced. A line is cached: B[0]...C[15].
Writing in A[0]. A miss is produced. A line is cached: A [0]...A [15].

From i = 0 to i = 7 all accesses are hits, all elements are in the cache.

i=8: Reading of C [8]. A miss occurs. A line is brought to cache: C[8]...C[15].
Reading of B[8]. A hit is produced. The element is in the cache.
Writing in A [8]. A hit is produced. The element is in the cache.

From i=9 to i = 15 all accesses are hits. From the i=16 iteration, the pattern repeats, so it is enough to analyze what happens in the first 16 iterations (from i=0 to i =15). In these 16 iterations, 16x3 = 48 memory accesses are produced, from which 4 are failures. Therefore, in a general way, the miss ratio is 4/48 and the hit ratio is 44/48. It has been considered that the accesses to a double suppose an access to memory. In case of requiring two memory accesses, the access number would be 16x4 = 64 accesses and the miss rateio is 4/64.

c) As the cache is structured in sets of 8 lines, you can always keep in cache lines corresponding to the three vectors. In addition, the data is not reused. Therefore, there are no ejections and the behavior is similar to a fully 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.