CARLOS III UNIVERSITY OF MADRID
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT BACHELOR IN INFORMATICS ENGINEERING. COMPUTER STRUCTURE January 8th, 2018 Final Exam

To conduct this final exam, count with 2 hours and a half**.** Textbooks and calculators are **NOT** allowed.

Exercise 1 (1 point). Given a 32-bit computer with a byte-level addressing mode and that includes a register bank of 32 registers. State:

a) What is the program counter and what type of register is?
b) How many MB of main memory this computer can support?
c) How many bits can be stored in a register and in a memory address?
d) Represented the value -2.5 for this computer, when the IEEE 754 standard of simple precision is used. Express the result in hexadecimal.

Exercise 2 (3 points). Consider the `Insert` subroutine. This routine accepts four input parameters

- The memory address of a matrix of elements of type `float` with dimension `NxN`
- The memory address of a vector of elements of type `float` with dimension `N`.
- The size of both matrix and vector (`N`).
- A number `j`.

The subroutine inserts the vector passed as the second argument into row `j` of the matrix. It also inserts this vector into column `j` of the matrix. Insertion involves copying the vector into the corresponding row and column. If the value of `j` is out of range, the subroutine returns -1 and does not perform any insertion, otherwise, it returns 0 and performs the insertion in the corresponding row and column.

State:

a) Encode the `Insert` subroutine described above. This routine must obligatorily to invoke two other subroutines, which must also be developed:
   a. A subroutine that inserts a vector in a specific row.
   b. A subroutine that inserts a vector in a specific column.

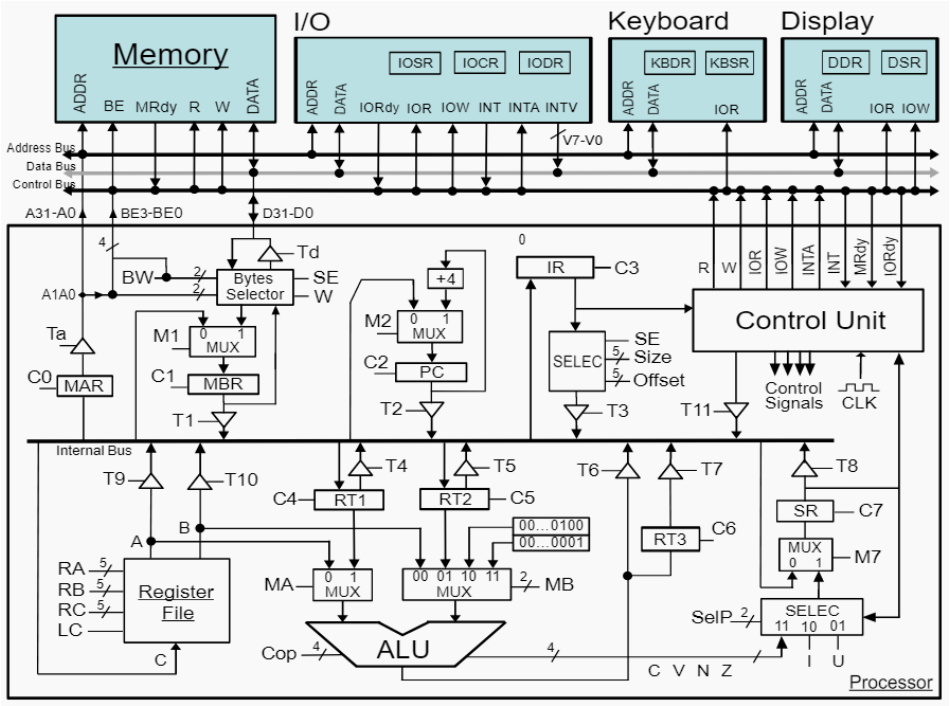
   Strictly follow the standard agreement of parameters passing explained in this course.

b) Given the following definition of 4x4 matrix and 4 elements vector:

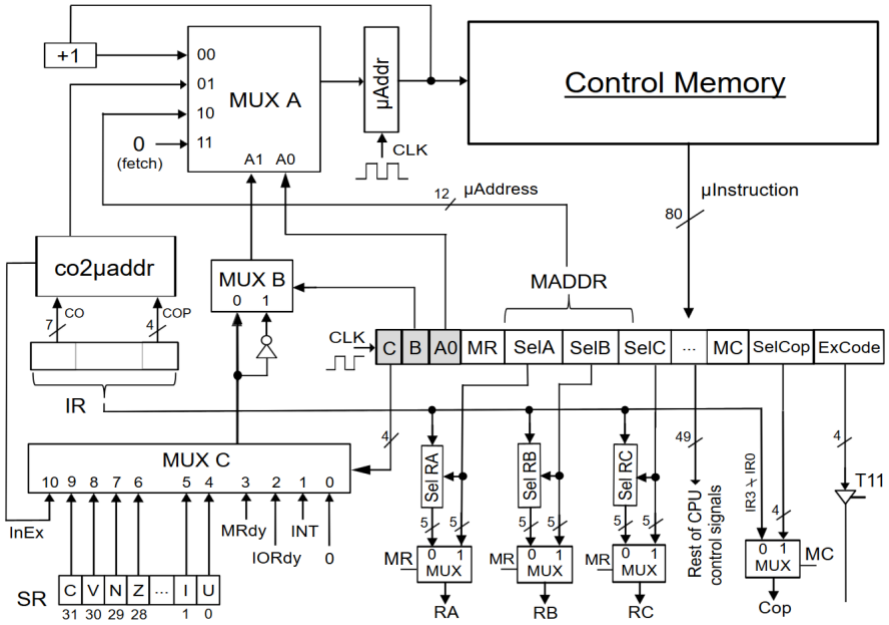
```
.data
  Matrix:  .float     0.0, 0.1, -0.2, 4.0
                      1.0, 1.1, 1.2, 3.0
                      2.0, 2.1, 2.2, 2.9
                      0.0, -2.3, 4.5, 8.0
  Vector: .float  4.0, 5.0, 6.0, 7.0
```

Encode the code snippet that permits the `Insert` subroutine to be invoked, inserting the vector into the row and column 2.

Exercise 3 (3 points). Given the WepSIM processor with the following structure:



This processor has a Control Unit represented by the following figure:



State:

a) Specify the elementary operations and control signals necessary to execute the machine instruction `strchr Ra Rb Rc` (include the fetch cycle). This instruction receives in the register indicated by `Rb` the non-zero address of a string of characters ending in the value zero and in the register indicated by `Rc` the ASCII code of a character. The instruction must return in `Ra` the address of the first occurrence of that character in the string. The only register that the instruction is allowed to modify is the one indicated by `Ra`.

b) Decide reasonably what will happen if the character does not appear in the string, considering the actions of the assembler programmer that uses the aforementioned instruction.

NOTE: Assume that R29 acts as a stack pointer, and that the stack pointer points to the top of the stack, and that the stack grows toward decreasing memory directions.

Exercise 4 (1 point). Figure out that the client company related to strlen_2 and skipasciicode_2 instructions (implemented in Assignment 2) has a new "smart" clinical sound level meter device that they are developing. There is a device prototype with an associated I/O module that employs a MIPS 32 architecture with a shared I/O map and programmed I/O technique.

The I/O module has three 32-bit registers:

- **Data register** (address 0x104). Stores the number of decibels read.
- **Control register** (address 0x108). When the value 0x123 is written in the register, the sensor is turned on. When the value 0x321 is written, the device is turned off. When the value 0x111 is written, the sensor is requested to do a new measurement.
- **Status register** (address 0x100). If the value of the register is 0, no measurement has been completed. If the value is 1, a measurement has been completed and the controller has the value in the data register. If the value is -1, an error has occurred and the sensor must be turned off again and turned on again (reset). A value of 0 indicates that the device has not yet finished its power-up and that it has not yet finished shutting down. A value of 1 indicates that the boot and shutdown have been completed

The company asks you to implement an assembler routine that will turn on the sensor and continuously read the sound level. The read value is printed on the screen (using the syscall with code 1). When an error is detected, the sensor must be turned off and on again to continue the reading process. The function never ends its execution.

Exercise 5 (2 point). Consider the following code snippet:

```
float  A[10000];
double B[10000];

for (i=0; i<10000; i++) {
     B[i] = A[i] + A[i];
}
```

This code is executed in an architecture with 32-bit words, which has a data cache of 128KB, 8-ways set-associative, and lines of 64 bytes. State:

a) Indicate the number of lines and sets of this configuration.

b) Indicate the hit rate produced by the execution of the previous code fragment taking into account only access to vectors A and B.

c) Modify the previous code fragment to reduce the number of accesses to memory.

SOLUTIONS

Exercise 1

a) The program counter is a control register that stores the address of the next executed instruction.
b) The program counter permits to address $2^{32} / 2^{20} = 2^{12}$ MB.
c) Each register stores 32 bits (word width). Each memory location stores 1 byte (byte addressing).
d) $-2.5_{(10} = 10.1_{(2} = 1.01 \times 2^{1}{}_{(2}$

Sign = 1 (negative)
Exponent = $127+1 = 128_{(10} = 10000000_{(2}$
Mantissa = 010000…. 000000 (23 bits)

The number will be 1 1000000 0100000…..000000 = 0xC0400000

Exercise 2

a)

```
insert: # check out-of-range values
          li   $v0, -1
          blt  $a3, $a0, error
          bgt  $a3, $a2, error

          # the return address and $ai registers
          # are stored in the stack
          addi $sp, $sp, -20
          sw   $ra ($sp)
          sw   $a0, 4($sp)
          sw   $a1, 8($sp)
          sw   $a2, 12($sp)
          sw   $a3, 16($sp)

          jal  InsertRow

          # Restore $ai registers
          # Just in case previous function modified them
          lw   $a0, 4($sp)
          lw   $a1, 8($sp)
          lw   $a2, 12($sp)
          lw   $a3, 16($sp)

          jal  InserCol

          lw   $ra ($sp)
          lw   $a0, 4($sp)
          lw   $a1, 8($sp)
          lw   $a2, 12($sp)
          lw   $a3, 16($sp)
          addi $sp, $sp, -20

          li   $v0, 0
error:    jr   $ra


InsertRow: #assumes that j is not out of range

          # We get the starting position of the first
```

```
                # element of row j
                li    $t0, 4
                mul   $t0, $t0, $a2
                mul   $t0, $t0, $a3

                add  $a0, $a0, $t0  # a0 points to row j

                li    $t0, 0
    loop1:      bge   $t0, $a2, end1
                lw    $t1,($a1)
                sw    $t1, ($a0)
                addi  $a1, $a1, 4
                addi  $a0, $a0, 4
                addi $t0, $t0,1
                b     loop1
    end1:       jr    $ra

    InserCol: # assumes that j is not out of range

                # We get the starting position of the first
                # element of column j

                li    $t0, 4
                mul   $t0, $t0, $a2

                add   $a0, $a0, $t0

                # we need to advance N*4 bytes
                # between consecutive elements
                li    $t1, 4
                li    $t1, $t1, $a2

                li    $t0, 0
    loop2:      bge   $t0, $a2, end2
                lw    $t1,($a1)
                sw    $t1, ($a0)
                addi  $a1, $a1, 4
                addi  $a0, $a0, t1  # Advances to next col.
                addi $t0, $t0,1
                b     loop2
    end2:       jr    $ra
```

b)

```
                la    $a0, Matrix
                la    $a1, Vector
                li    $a2, 4
                li    $a3, 2
                jal   Insert
```

Exercise 3

a)

Operation	Control signals
MAR <- PC	T2, C0
M[MAR] -> MBR, PC <- PC+4	Ta, R, BW=11, M1=1, C1, M2=1, C2
MBR -> RI	T1, C3
Decode	C=0, B=0, A0=1
RT2 <- SR	T8, C5
{MAR, R1} <- R2	MR=0, SelA=<R2>, SelC=<R1>, LC=1, T9, C0
loop1: MBR <- Mem[R1]	Ta, Td, BW=11, R, M1, C1
RT1 <- MBR	T1, C4
RT1 * 1 -> flags	MA=1, MB=11, SelCOP=*, MC=1, SelP=11, M7, C7
bZ end1	C=6, B=0, A0=0, MADDR=end1
RT1 – R3 -> flags	MA=1, SelB=<R3>, SelCOP=-, MC=1, M7, C7
bZ end2	C=6, B=0, A0=0, MADDR=end2
R1 <- R1 + 1	MR=0, SelA=<R1>, SelC=<R1>, LC=1, MB=11, SelCOP=+, MC=1, T6
B loop1	C=0, B=1, A0=0, MADDR=loop1
end1: R1 <- 0	MR=0, SelA=<R0>, SelC=<R1>, LC=1, T9, C0
end2: SR <- RT2	MB=01, SelA=<R0>, SelCOP=+, MC=1, T6, C7
Branch to fetch	C=0, B=1, A0=0, MADDR=fetch

b) Given that address return value is expected, the address 0 could be used to code a no existing value (if in the first memory locations there is not user-related information, this value could be used apart of the standard use of strchr).

Exercise 4

A possible solution would be

```
device:
                # turn-on the device

   ini:             li    $t0, 0x123
                    sw    $t0, 0x108

                     # read control register until a non-zero value is get
   bwait:           lw     $t0,  0x100
                    beqz   $t0,  bwait

   measurements:    li    $t1, 0x111
                    li    $t2, -1

   measurement:     sw    $t1, 0x108

   bread:           lw    $t0, 0x100
                    beqz  $t0, bread

                    #read the value of the data register
                    lw    $a0,  0x104

                    # checks for errors
                    beq   $a0, $t1, error
```

```
            # prints the read value and reads again
            li    $v0  1
            syscall
            b measurement

error:      # turn off command
            li   $t0, 0x321
            sw   $t0, 0x108

            # waits for turning off the device
boff:       lw   $t0, 0x100
            beqz $t0, boff

            # returns to the beginning
            b    ini

            jr   $ra
```

Exercise 5.

a) The cache size is 128KB = 2^{17} bytes. Line width is 64 bytes, consequently, the number of lines is $2^{17} / 2^6 = 2^{11} = 2048$. The number of sets is $2^{11} / 2^3 = 2^8$ = 256

b) In each cache line of 64 bytes, 64/4 = 16 entries of A can be stored (floating-point entries) and 64/8 = 8 entries of B (double-precision entries).

Firstly, we will consider the accesses to A:

Iteration i = 0;

- Read of A[0] miss block A[0]… A[15] is transferred to cache
- Read of A[0] hit
- Read of A[1] hit
- Read of A[1] hit
- …. the Access pattern is repeated

Every 16 iterations there are 32 accesses to A (two reads) and a single miss. In 10000 iterations there will be 20000 accesses to A with 20000 x 1/32 = 625 misses

Secondly, we will consider the accesses to B:

Iteration i = 0;

- Write to B[0] miss block B[0]… B[7] is transferred to cache
- Write to B[1] hit
- …. the Access pattern is repeated
-

Every 8 iterations there are 8 accesses to B. One of them is a miss. In 10000 iterations there are 10000 accesses, with 10000 x 1/8 = 1250 misses.

The total number of accesses is 30000 with 30000-(625+1250) = 28125 hits.

The hit rate is 28125/30000 = 0.9375

c) One way of reducing the number of accesses is the following code example:

```
for (i=0; i<10000; i++) {
     B[i] = 2* A[i];
}
```