Introduction to Computers
September 7, 2018. Partial exam (second term) / Final exam (part 2)

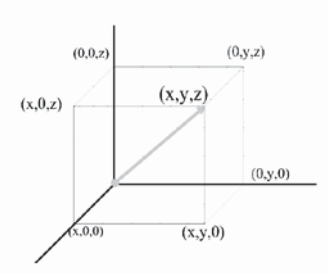
Name:_____ DNI: _____

Surname: _____

Students taking the Partial exam must answer all the questions.

Students taking the Final exam must answer questions: 1a, 2a, 2b, from 4a to 4c

Exercise 1 (3 points). Consider three arrays, X, Y and Z, of N elements, that contain, respectively, the x, y and z coordinates, of a set of N three-dimensional vectors. For example, the triplet (X[i], Y[i], Z[i]) defines the i-th vector of this set. In this exercise, you must write am ARM assembly program to find the index of the vector with the highest modulus, and the value of such modulus. If there are several vectors with the highest modulus, the program must report the index of the first one (i.e. the lowest index). The main program must follow the specification of the high-level pseudo-code given in the left box:




```
#define N, ...
int X[N]={a list of N integer values};
int Y[N]={a list of N integer values };
int Z[N]={a list of N integer values };
void main ()  {
int mod_max=0, vector_max=0, mod;
for (i=0; i<N;  i++){
   mod = modulus(X[i],Y[i],Z[i]);
   if (mod > mod_max){
         mod_max = mod;
         vector_max = i;
         }
   }
}
```

```
int modulus (int a,int b,int c)
{
   int result;
   result= square_root(a*a + b*b + c*c);
   return result;
}
```

a) (1.5 points) Write an assembly code for the main program. As shown in the left box, the program must call the subroutine `modulus(a,b,c)`, which returns the modulus of the vector (a,b,c). The code must be compliant with the procedure call standard studied in class.

b) (1.5 points) Write an ARM assembly code for the subroutine `modulus`, according to the specification given in the right box. We assume that the function `square_root` has been previously implemented, and returns the integer part of the square root of its input parameter. The code must be compliant with the procedure call standard studied in class.

Exercise 2 (3 points). We want to extend the instruction set of the BasicMIPS with the new instruction SWAP (interchange the contents of two memory words):

```
SWAP (Rs), (Rt)
```

which is encoded as shown below:

6	5	5	16
110011	rs	rt	Not used

and whose behavior is as follows:

```
Memory(RF(rs))<- Memory(RF(rt)), Memory(RF(rt))<- Memory(RF(rs)), PC <- PC + 4
```

In other words, the new content of the memory word at address RF(rs) is made equal to the original content of the memory word at address RF(rt) and, similarly, the new content of the memory word at address RF(rt) is made equal to the original content of the memory word at address RF(rs)
Please, complete the following tasks:
a) (1 point) Modify the datapath to allow the execution of the new instruction.
b**)** (1 point) Modify the state diagram to include the new instruction.
c) (1 point) Modify the truth table, adding any required rows and/or columns, in order to control the new datapath.
HINT: According to the definition of the new instruction, you cannot modify the content of any register in the register file. However, you can add new registers to the datapath.
REMARK: For each part of this exercise, you must explain the reasons for your decisions

Ejercicio 3. **(1 point)** The ARM code shown in the right box, has been extracted from the initialization program of a videogames console, where some instructions have been omitted due to legal reasons.
Taking into account the following data:

- The execution time of the different instruction types is as follows:
  - Register-transfer: 3 cycles.
  - Arithmetic-logical (including comparisons), branch and store: 4 cycles.
  - Load: 5 cycles
- The processor clock runs at 266 MHz

find the CPI and the execution time for this code fragment.

```
start:       mov r0, #0x0
             add r3, r0,#0xF
             ldr r4, =bss_start
             mov r1, #0
             ldr r2, =bss_end
             sub r2, r2, r0
             ldr sp, =stack_top
firmlaunch:  str r5, [r1]
             mov r0, #0x10
wait:        add r1, r1, #4
             cmp r1, r0
             bne wait
launched:    b   main
```

Exercise 4 (3 points). A byte-addressable memory hierarchy is composed of a Main Memory (MM) of 16 Gbytes and a direct mapped cache ($) of 1 Mbyte. The memory is organized in blocks of 256 bytes. The MM access time is 20 ns, the $ access time is 1 ns, and the miss penalty is 500 ns. Answer the following questions:
a) (0.25 points) Show the address format for the MM and the $.
b) (0.75 points) Assuming that the $ is initially empty, the following memory addresses are read: 0x45AD, 0x2455D007B, 0x2F and 0xB024AA4F. Complete the table below to show the $ blocks that are filled as a consequence of the generated misses , the value of the corresponding TAGs, and the range of memory addresses (in hexadecimal) that are copied to each $ block.

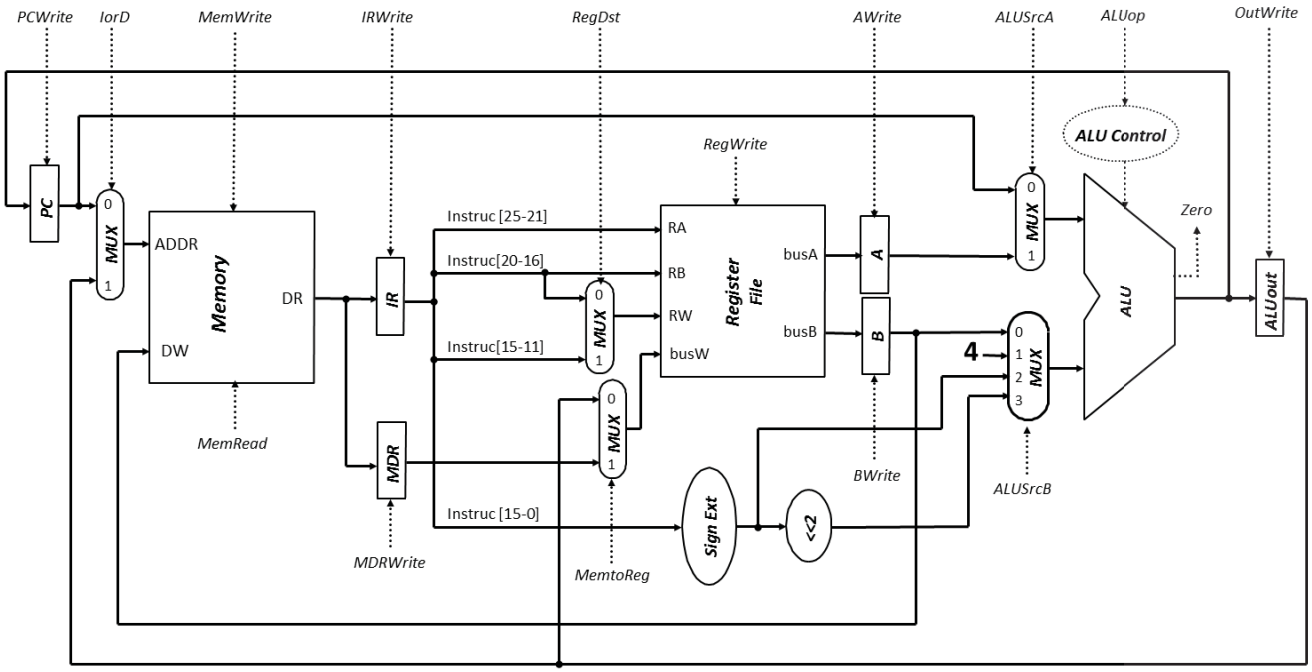
No. of $ block	TAG	Range of MM addresses
0x	0x	0x – 0x
0x	0x	0x – 0x
0x	0x	0x – 0x
0x	0x	0x – 0x

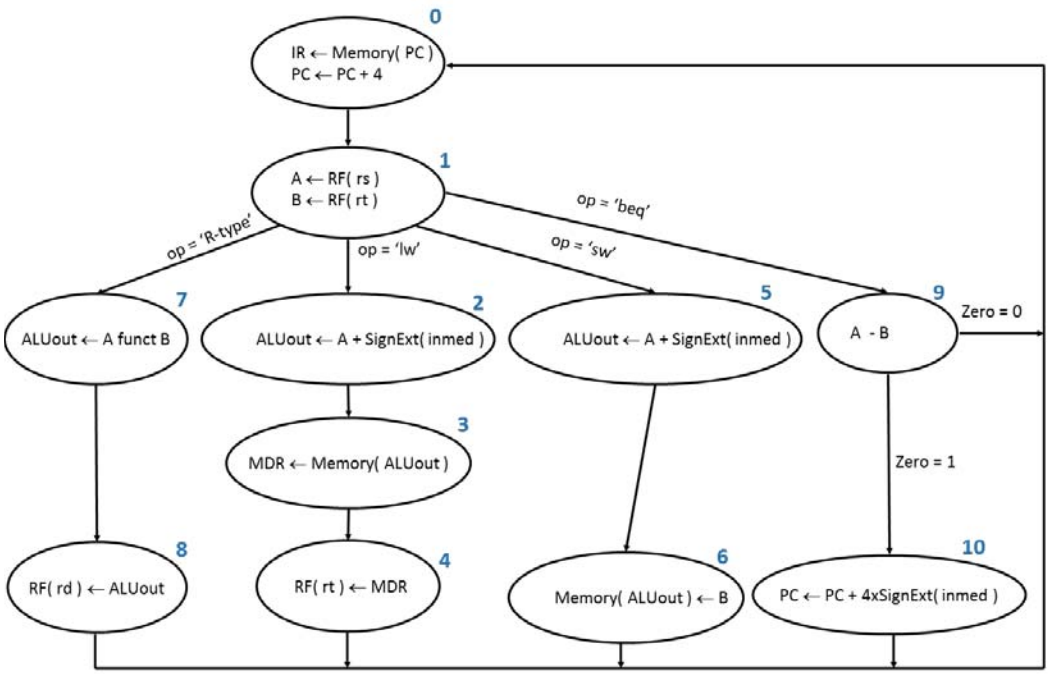
c) (1 point) Assuming again that the $ is empty, the processor generates 100 times the following sequence of memory references: from 0x3AB00AF00 to 0x3AB00B9FF, both inclusive. Find the miss rate.
d) (0.5 points) Find the time required to execute the sequence of references of the previous question.
e) (0.5 points) Taking into account the previous result, calculate the speed gain that is achieved in comparison to the same computer without the $.

Introduction to Computers
September 7, 2018. Partial exam (second term) / Final exam (part 2)

Name:_____ DNI: _____

Surname: _____





Current state		op	Zero	Next state	IRWrite	PCWrite	AWrite	BWrite	ALUSrcA	ALUScrB	ALUOp	OutWrite	MemWrite	MemRead	IorD	MDRWrite	MemtoReg	RegDest	RegWrite
0000	(fetch)	XXXXXX	X	0001	1	1			0	01	00 (add)		0	1	0				0
0001	(deco)	100011 (lw)	X	0010															
0001	(deco)	101011 (sw)	X	0101															
0001	(deco)	000000 (R-type)	X	0111	0	0	1	1					0	0					0
0001	(deco)	000100 (beq)	X	1001															
0010	(ex-lw)	XXXXXX	X	0011	0	0			1	10	00 (add)	1	0	0					0
0011	(mem-lw)	XXXXXX	X	0100	0	0							0	1	1	1			0
0100	(wb-lw)	XXXXXX	X	0000	0	0							0	0			1	0	1
0101	(ex-sw)	XXXXXX	X	0110	0	0		0	1	10	00 (add)	1	0	0					0
0110	(wb-sw)	XXXXXX	X	0000	0	0							1	0	1				0
0111	(ex-R)	XXXXXX	X	1000	0	0			1	00	10 (funct)	1	0	0					0
1000	(wb-R)	XXXXXX	X	0000	0	0							0	0			0	1	1
1001	(ex-beq)	XXXXXX	0	0000	0	0			1	00	01 (sub)		0	0					0
1001	(ex-beq)	XXXXXX	1	1010															
1010	(wb-beq)	XXXXXX	X	0000	0	1			0	11	00 (add)		0	0					0