Facultad de Informática
Universidad Complutense de Madrid

FINAL EXAM – INTRODUCTION TO COMPUTERS II

24TH MAY 2024

The following program has been written in RISC-V assembler. Assume in all exercises that the pseudoinstructions `la` and `li` are translated to a single `addi` instruction in the assembly phase, the *.text* section is located from memory address 0x0 on and the *.data* and *.bss* sections are located one after another starting at address 0x10000.

```
.global main
.equ n, 6
.data
  f: .word  2
.bss
  Res: .space 4
.text
main:
  la   t1, f              # t1 = &f
  lw   s1, 0(t1)          # s1 = f
  li   s2, n              # s2 = n
  li   s3, 2              # s3 = 2 (index)
for:
  beq  s3, s2, fin_for
  add  s1, s1, s1
  addi s3, s3, 1
  jal  x0, for            # j for
fin_for:
  la t1, Res
  sw s1, 0(t1)
end:
  j  .
.end
```

1. **Single-cycle [0,5 pt]**. On the monocycle processor indicate the values of the registers and positions memory values shown below in each of the cycles of the first iteration of the loop.

Cycle num.	Instrucción	PC	s1	s2	s3	M[0x10000]
1						
2						
...						

2. **Single-cycle [0,5 pt]**. On the single-cycle processor, indicate the values of the control signals shown below in each of the cycles of the first iteration of the loop.

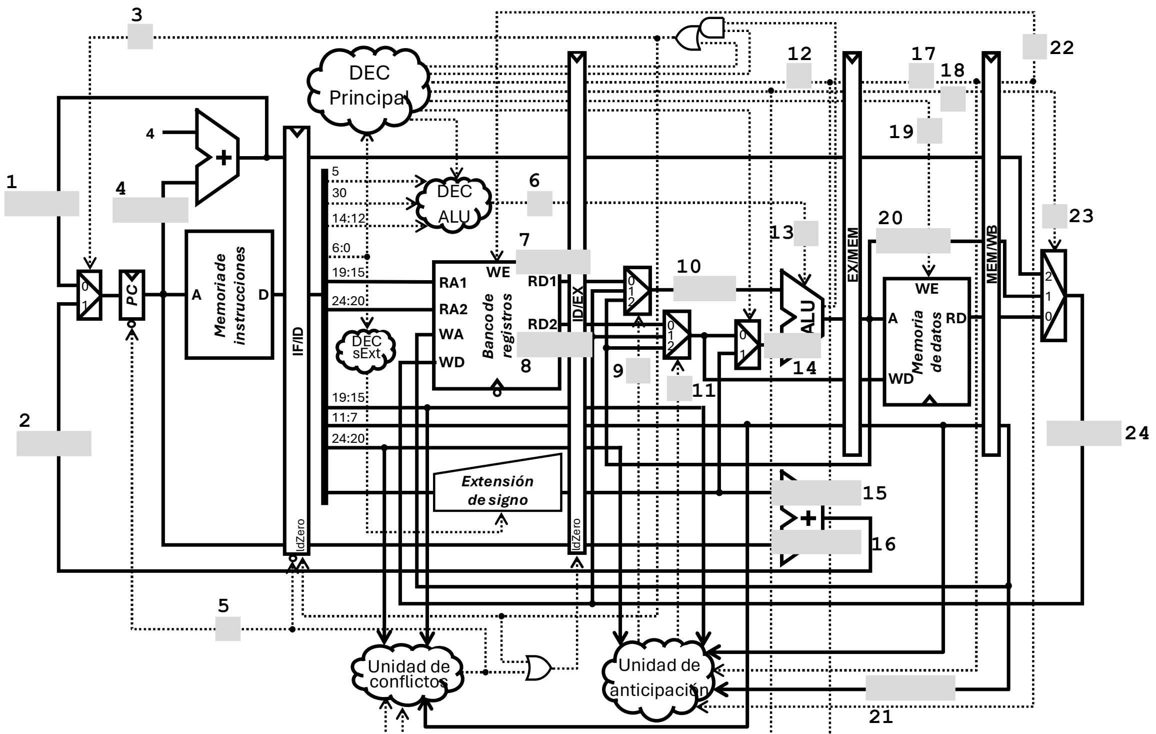
Cycle num.	Instrucción	PCsrc	Branch	Jump	BRwr	ALUsrc	MemWr	ResSrc	ImmSrc
1									
2									
...									

3. **Multicycle [1 pt]**. Assuming that the program (up to but not including instruction `j`) takes 58.7 ns to execute on a multicycle processor with a clock frequency of 1.5 GHz.
   a) Calculate the CPI of the program.
   b) Calculate the value of the MIPS metric.

4. **Pipelined [1,5 pt]**. On the pipelined processor without hazard management (RF write at the end of the cycle):
   a) Inserting the necessary **nop** instructions so that the previous processor gets a correct result.
   b) Draw the execution diagram of the program for the first iteration of the loop.
   c) Calculate the average cycles per instruction (CPI) for the whole execution of the loop*.

5. **Pipelined [1,5 pt]**. On the pipelined processor with full hazard management (RF write in the middle of the cycle, forwarding unit, and hazard unit with non-taken branch prediction):
   a) Draw the execution diagram of the program for the first five instructions.
   b) Calculate the number of cycles needed to execute them*.
   c) Calculate the average cycles per instruction (CPI) for the whole execution of the program (up to the fetching but not including instruction `j`)*.

6. **Pipelined [2 pt]**. On the pipelined processor with full hazard management shown below, indicate the instruction found in each stage of the processor and the values taken by the 24 signals indicated in cycle 7 of the program execution. The correspondence between the aliases and the register numbers are as follows: t1 = x6, s1= x9, s2 = x18 and s3 = x19.



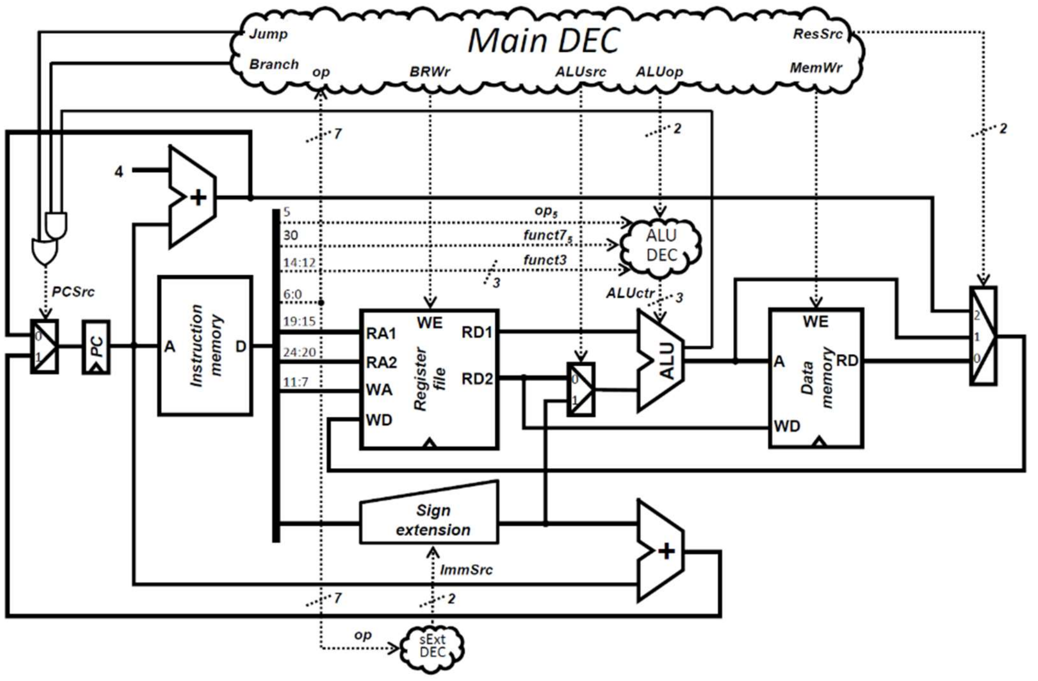
7. **Exceptions [1 pt]**. Suppose that when executing the program the instruction **`lw s1,0(t1)`**, it triggers an exception in the pipelined processor, indicate:
   a) What type of exception would it be?
   b) In what stage of the processor would it occur?
   c) In what cycle would it happen?
   d) What specific circumstance has occurred for this exception to occur and why?
   e) Which fetched instructions would be canceled?

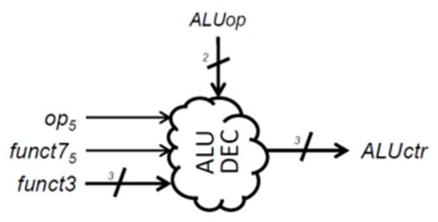
* When counting the cycles it takes to execute a block of code, you must count from the cycle in which the first instruction of the block is fetched (included) to the cycle in which the next instruction outside the block is fetched (not included).

8. **Assembly [1,5 pt]**. Complete the code so that, just before the **`end`** label, the program calls a subroutine name **`subr`** that meets the following conditions:
   - The calling program wants to preserve the value of **`t1`**.
   - The values that must be passed as parameters are found in registers **`s2`** y **`s3`**.
   - The result must be stored in the position defined by the **`res`** label.
   - Additionally, the program must initialize the stack first, assuming that the stack points to the address indicated by the external **`_stack`** label.

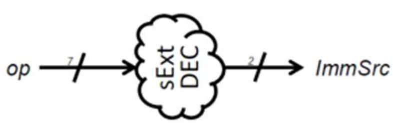
9. **Assembly [0,5 pt]**. Code the prologue and epilogue of the subroutine indicated in the previous exercise assuming that is a non-leaf type, that uses registers **`s3`** y **`s5`**, and that the final result must be returned to the calling function is stored in **`s5`**.

SINGLE-CYCLE PROCESSOR

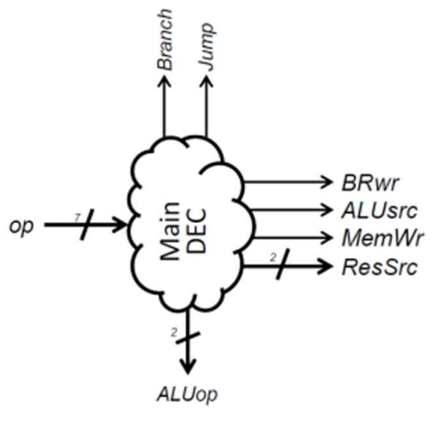




ALUop	op$_5$	funct7$_5$	funct3	ALUctr
00$^{\text{(add)}}$	X	X	XXX	000$^{\text{(A + B)}}$
01$^{\text{(subtract)}}$	X	X	XXX	001$^{\text{(A – B)}}$
10$^{\text{(operate)}}$	0	X	000$^{\text{(addi)}}$	000$^{\text{(A + B)}}$
10$^{\text{(operate)}}$	1	0	000$^{\text{(add)}}$	000$^{\text{(A + B)}}$
10$^{\text{(operate)}}$	1	1	000$^{\text{(sub)}}$	001$^{\text{(A – B)}}$
10$^{\text{(operate)}}$	X	X	010$^{\text{(slt/slti)}}$	101$^{\text{(A < B)}}$
10$^{\text{(operate)}}$	X	X	110$^{\text{(or/ori)}}$	011$^{\text{(A \| B)}}$
10$^{\text{(operate)}}$	X	X	111$^{\text{(and/andi)}}$	010$^{\text{(A \& B)}}$

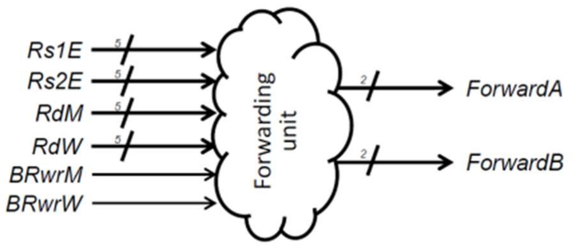


Op	ImmSrc
0000011$^{\text{(lw)}}$	00$^{\text{(I-type)}}$
0100011$^{\text{(sw)}}$	01$^{\text{(S-type)}}$
0010011$^{\text{(I-type)}}$	00$^{\text{(I-type)}}$
0110011$^{\text{(R-type)}}$	–
1100011$^{\text{(beq)}}$	10$^{\text{(B-type)}}$
1101111$^{\text{(jal)}}$	11$^{\text{(J-type)}}$



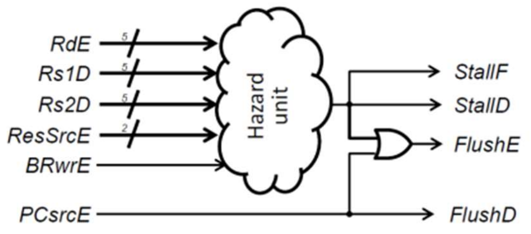
op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011$^{\text{(lw)}}$	0	0	1	1	00$^{\text{(add)}}$	0	00
0100011$^{\text{(sw)}}$	0	0	0	1	00$^{\text{(add)}}$	1	–
0010011$^{\text{(I-type)}}$	0	0	1	1	10$^{\text{(operate)}}$	0	01
0110011$^{\text{(R-type)}}$	0	0	1	0	10$^{\text{(operate)}}$	0	01
1100011$^{\text{(beq)}}$	1	0	0	0	01$^{\text{(subtract)}}$	0	–
1101111$^{\text{(jal)}}$	0	1	1	–	–	0	10

PIPELINE PROCESSOR



```
if ( (Rs1E ≠ 0) & BRwrM & (Rs1E = RdM) ) then      ( ForwardA ← 10 (forwarding MEM) )
  elsif( (Rs1E ≠ 0) & BRwrW & (Rs1E = RdW) ) then  ( ForwardA ← 01 (forwarding WB) )
    else                                           ( ForwardA ← 00 (no forwarding) )

if ( (Rs2E ≠ 0) & BRwrM & (Rs2E = RdM) ) then      ( ForwardB ← 10 (forwarding MEM) )
  elsif( (Rs2E ≠ 0) & BRwrW & (Rs2E = RdW) ) then  ( ForwardB ← 01 (forwarding WB) )
    else                                           ( ForwardB ← 00 (no forwarding) )
```



```
if ( (ResSrcE = 0) & BRwrE & ((Rs1D = RdE) | (Rs2D = RdE)) ) then ( lwStall ← 1 ) (stall pipeline)
else                                                              ( lwStall ← 0 ) (don't stall pipeline)
StallF = StallD = lwStall
FlushD = PCsrcE
FlushE = lwStall | PCsrcE
```

INSTRUCTION FORMAT

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	op	*R-type*
$imm_{11:0}$		rs1	funct3	rd	op	*I-type*
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	*S-type*
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	*B-type*
$imm_{20,10:1,11,19:12}$				rd	op	*J-type*

Instruction	Type	funct7 bits 31:25	funct3 bits 14:12	op bits 6:0
`lw`	I	–	010	0000011
`sw`	S	–	010	0100011
`add`	R	0000000	000	0110011
`sub`	R	0100000	000	
`slt`	R	0000000	010	
`or`	R	0000000	110	
`and`	R	0000000	111	
`addi`	I	–	000	0010011
`slti`	I	–	010	
`ori`	I	–	110	
`andi`	I	–	111	
`beq`	B	–	000	1100011
`jal`	J	–	–	1101111