

DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI  
 → NON USARE FOGLI NON TIMBRATI  
 → ANDARE IN BAGNO PRIMA DELL'INIZIO DELLA PROVA  
 → NO FOGLI PERSONALI, NO TELEFONI, SMARTPHONE/WATCH, ETC

COGNOME \_\_\_\_\_

NOME \_\_\_\_\_

NOTA: dovrà essere consegnato l'elaborato dell'es.1 come file <COGNOME>.s e quelli dell'es. 4 come files <COGNOME>.v e <COGNOME>.png

1) [10/30] Trovare il codice assembly RISC-V corrispondente al seguente micro-benchmark (utilizzando solo e unicamente istruzioni dalla tabella sottostante), rispettando le convenzioni di uso dei registri dell'assembly (riportate qua sotto, per riferimento).

```
char daytab[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
};

int main() {
    int m=day_of_year(2023, 9, 14);
    print_int(m);
    exit(0);
}
```

Nota: 'int' è un intero a 64 bit.

```
int day_of_year(int year, int month, int day) {
    int i, leap;
    if (year < 1752 || month < 1 || month > 12 || day < 1)
        return -1;
    leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;
    if (day > daytab[leap][month])
        return -1;
    for (i = 1; i < month; i++)
        day += daytab[leap][i];
    return day;
}
```

RISCV Instructions (RV64IMFD)

v221117

| Instruction coding (hexadecimal) |        |        | Instruction                             | Example                    | Register operation                                      | Meaning<br><small>(** instructions available only in RV64, i.e. 64-bit case)</small>              |
|----------------------------------|--------|--------|---|----------------------------|---|---|
| funct7/imm                       | funct3 | opcode |   |                            |   |   |
| 00                               | 0      | 33/3b  | add                                     | add/addw x5,x6,x7          | x5 ← x6 + x7  | Add two operands; exception possible (addw**)   |
| 20                               | 0      | 33/3b  | subtract                                | sub/subw x5,x6,x7          | x5 ← x6 - x7  | Subtracts two operands; exception possible (subw**)   |
| imm                              | 0      | 13/1b  | add immediate                           | addi/addiw x5,x6,100       | x5 ← x6 + 100   | Add a constant; exception possible (addiw**)  |
| 01                               | 0      | 33/3b  | multiply                                | mul/mulw x5,x6,x7          | x5 ← x6 * x7  | (signed/word) Lower 64 bits of 128-bits product (mulw**)  |
| 01                               | 1      | 33     | multiply high                           | mulh x5,x6,x7              | x5 ← x6 * x7  | Higher 64bits of 128-bits product   |
| 01                               | 4      | 33/3b  | division                                | div/divw x5,x6,x7          | x5 ← x6/x7  | (signed/word) division (divw**)   |
| 01                               | 6      | 33/3b  | remainder                               | rem/remw x5,x6,x7          | x5 ← x6 % x7  | Remainder of the division (remw**)  |
| 00                               | 2/3    | 33     | set on less than                        | slt/sltu x5,x6,x7          | if (x6 < x7) x5 ← 1; else x5 ← 0                        | signed compare x6 and x7 (less than)  |
| imm                              | 2/3    | 13     | set on less than immediate              | slti/sltiu x5,x6,100       | if (x6 < 100) x5 ← 1; else x5 ← 0                       | unsigned compare x6 and 100 (less than)   |
| 00                               | 7/6/4  | 33     | and / or / xor                          | and/or/xor x5,x6,x7        | x5 ← x6&x7 / x6 x7 / x6^x7                              | Logical AND/OR/XOR register operand   |
| imm                              | 7/6/4  | 13     | and / or / xor immediate                | andi/ori/xori x5,x6,100    | x5 ← x6&100 / x6 100 / x6^100                           | Logical AND/OR/XOR constant operand   |
| 0                                | 1      | 33/3b  | shift left logical                      | sll/sllw x5,x6,x7          | x5 ← x6 << x7   | Shift left by register (sllw**)   |
| imm                              | 1      | 13/1b  | shift left logical immediate            | slli/slliw x5,x6,10        | x5 ← x6 << 10   | Shift left by the immediate value (slliw**)   |
| 0                                | 5      | 33/3b  | shift right logical                     | srl/srlw x5,x6,x7          | x5 ← x6 >> x7   | Shift right by register (srlw**)  |
| imm                              | 5      | 13/1b  | shift right logical immediate           | srli/srliw x5,x6,10        | x5 ← x6 >> 10   | Shift left by immediate value (srliw**)   |
| 20                               | 5      | 33/3b  | shift right arithmetic                  | sra/sraw x5,x6,x7          | x5 ← x6 >> x7 (arith.)                                  | Shift right by register (sign is preserved) (sraw**)  |
| imm                              | 5      | 13/1b  | shift right arithmetic immediate        | srai/sraiw x5,x6,10        | x5 ← x6 >> 10 (arith.)                                  | Shift right by immediate value (sraiw**)  |
| imm                              | 3/2/0  | 03     | load dword / word / byte                | ld/lw/lb x5,100(x6)        | x5 ← MEM[x6+100]  | Data from memory to register  |
| imm                              | 6/4    | 03     | load word / byte unsigned               | lwu/lbu x5,100(x6)         | x5 ← MEM[x6+100]  | Data from mem. To reg.; no sign extension (lbu**)   |
| imm                              | 3/2    | 23     | store dword / word / byte               | sd/sw/sb x5,100(x6)        | MEM[x6+100] ← x5  | Data from register to memory (sw**)   |
| imm[31:12]                       | -      | 37     | load upper immediate                    | lui x5,0x12345             | x5 ← 0x12345000   | Load most significant 20 bits   |
| PSEUDOINSTRUCTION                |        |        | load address                            | la x5,var                  | x5 ← &var (PSEUDO INST.)<br>load address of 'var' in x5 | REAL: lui x5,H20(&var);ori x5,L12(&var) INST. (H20=high 20 bits of &var; L12=low 12 bits of &var) |
| imm[31:12](rd=0)                 | -      | 6/6/3  | jump/branch                             | j/b label                  | PC+=off (off=PC-&label) (PS.INST.)                      | REAL INST.: jal x0,offset/beq x0,x0,offset  |
| imm[11:0](rs1=rs2=0)             | -      | 6/6/3  | jump and link (offset)                  | jal label                  | x1←(PC+4); PC+=offset (PS. INST.)                       | REAL INST.: jal x1,offset (offset=PC-&label)  |
| imm[31:12](rd=1)                 | -      | 6f     | return from procedure                   | ret                        | PC←x1 (PSEUDO INST.)                                    | REAL INST.: jalr x0,0(x1)   |
| imm (rd=0,rs=1)                  | 0      | 67     | jump and link register                  | jalr x1,100(x5)            | x1 ← (PC + 4); PC=x5+100                                | Procedure return; indirect call   |
| imm+2                            | 0/1    | 63     | branch on equal / not-equal             | beq/bne x5,x6,100          | if (x5 ==/= x6) PC=PC+100                               | Equal / Not-equal test; PC relative branch  |
| 00 (rs1=0,rs2=0,rd=0)            | 0      | 73     | ecall                                   | ecall                      | SEPC←PC;PC←STVEC;save PLIE;PL=1;IE=0                    | Call OS (service number in a7); PL= privilege lev; IE=int.en.                                     |
| 08 (rs1=0,rs2=2,rd=0)            | 0      | 73     | sret                                    | sret                       | PC←SEPC; restore PL/IE                                  | Exit supervisor mode; PL= privilege lev; IE=int.en.   |
| PSEUDOINSTRUCTION                |        |        | move                                    | mv x5,x6                   | x5 ← x6 (PSEUDO INST.)                                  | REAL INST.: add x5,x0,x6  |
| PSEUDOINSTRUCTION                |        |        | load immediate                          | li x5,100                  | x5 ← 100 (PSEUDO INST.)                                 | REAL INST.: addi x5,x0,100  |
| PSEUDOINSTRUCTION                |        |        | no operation (nop)                      | nop                        | do nothing (PSEUDO INST.)                               | REAL INST.: addi x0,x0,0  |
| (0,1) / (4,5)                    | 0      | 53     | floating point add/sub                  | fadd/fsub.{s,d} f0,f1,f2   | f0←f1+f2 / f0←f1-f2                                     | Single or double precision add / subtract   |
| (8,9) / (c,d)                    | 0      | 53     | floating point multiplication/division  | fmul/fdiv.{s,d} f0,f1,f2   | f0←f1*f2 / f0←f1/f2                                     | Single or double precision multiplication / division  |
| PSEUDOINSTRUCTION                |        |        | floating point move between f-reg       | fmv.{s,d} f0,f1            | f0←f1 (PSEUDO INST.)                                    | REAL INST.: fsgnj.{s,d} f0,f1,f1  |
| PSEUDOINSTRUCTION                |        |        | floating point negate                   | fneg.{s,d} f0,f1           | f0←-(f1) (PSEUDO INST.)                                 | REAL INST.: fsgnjn.{s,d} f0,f1,f1   |
| PSEUDOINSTRUCTION                |        |        | floating point absolute value           | fabs.{s,d} f0,f1           | f0← f1  (PSEUDO INST.)                                  | REAL INST.: fsgnjx.{s,d} f0,f1,f1   |
| (50,51)                          | 0/1/2  | 53     | floating point compare                  | fle/flt/feq.{s,d} x5,f0,f1 | x5←(f0<f1)  | Single and double: compare f0 and f1 <,<=   |
| (70,71) (rs2=0)                  | 0      | 53     | move between x (integer) and f regs     | fmv.x.{s,d} x5,f0          | x5←f0 (no conversion)                                   | Copy (no conversion)  |
| (78,79) (rs2=0)                  | 0      | 53     | move between f and x regs               | fmv.{s,d}.x f0,x5          | f0←x5 (no conversion)                                   | Copy (no conversion)  |
| imm                              | 2      | 7      | load/store floating point (32bit)       | flw/fsw f0,0(x5)           | f0←MEM[x5] / MEM[x5]←f0                                 | Data from FP register to memory   |
| imm                              | 3      | 7      | load/store floating point (64bit)       | fld/fsd f0,0(x5)           | f0←MEM[x5] / MEM[x5]←f0                                 | Data from FP register to memory   |
| 21/20 (rs2=0)                    | 7      | 53     | convert to/from double from/to single   | fcvt.d.s/fcvt.s.d f0,f1    | f0←(double)f1 / f0←(single)f1                           | Type conversion   |
| (60,61) (rs2=0)                  | 7      | 53     | convert to integer from (single,double) | fcvt.w.{s,d} x5,f0         | x5←(int)f0  | Type conversion   |
| (68,69) (rs2=0)                  | 7      | 53     | convert to (single,double) from integer | fcvt.{s,d}.w f0,x5         | f0←((single,double))x5                                  | Type conversion   |
| (2c,2d) (rs2=0)                  | 0      | 53     | square root                             | fsqrt.{s,d} f0,f1          | f0←square root of f1                                    | Single or double square root  |
| (10,11)                          | 0/1/2  | 53     | sign injection                          | fsgnj/jn/jx.{s,d} f0,f1,f2 | f0←sgn(f2) f1 / -sgn(f2) f1 / sgn(f2)f1                 | Extract the mantissa and exp. from f1 and sign from f2  |

Register Usage

| Register      | ABI Name     | Usage                 |
|---------------|--------------|-----------------------|
| x10-x11       | a0-a1        | arguments and results |
| x9, x18-x27   | s1, s2-s11   | Saved                 |
| x5-7, x28-x31 | t0-t2, t3-t6 | Temporaries           |
| x12-x17       | a2-a7        | Arguments             |

| Register | ABI Name | Usage                          |
|----------|----------|--------------------------------|
| x0       | zero     | The constant value 0           |
| x8, x2   | s0fp, sp | frame pointer, stack pointer   |
| x1, x3   | ra, gp   | return address, global pointer |
| x4       | tp       | thread pointer                 |

| Register       | ABI Name          | Usage                      |
|----------------|-------------------|----------------------------|
| f10-f11        | fa0-fa1           | Argument and Return values |
| f8-f9, f18-f27 | fs0-fs1, fs2-fs11 | Saved registers            |
| f0-f7, f28-f31 | ft0-ft7, ft8-ft11 | Temporaries registers      |
| f12-17         | fa2-fa7           | Function arguments         |

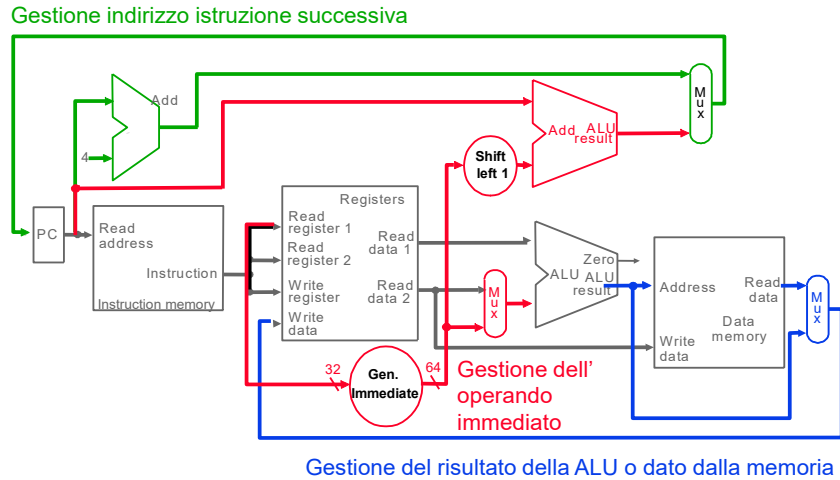
System calls

| Service Name | Serv.No.(a7) | INPUT Arguments                     | OUTPUT Args |
|--------------|--------------|-------------------------------------|-------------|
| print_int    | 1            | a0=integer to print                 | ---         |
| print_float  | 2            | fa0=float to print                  | ---         |
| print_double | 3            | fa0=double to print                 | ---         |
| print_string | 4            | a0=address of ASCII string to print | ---         |
| read_int     | 5            | ---                                 | a0=integer  |

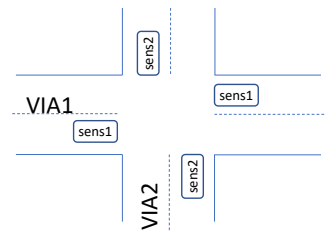
| Service Name | Serv.No.(a7) | INPUT Arguments                                  | OUTPUT Arguments               |
|--------------|--------------|--|--------------------------------|
| read_float   | 6            | ---  | fa0=float                      |
| read_double  | 7            | ---  | fa0=double                     |
| read_string  | 8            | a0=address of input buffer, a1=max chars to read | ---                            |
| sbrk         | 9            | a0=Number of bytes to be allocated               | a0=pointer to allocated memory |
| exit         | 10           | ---  | ---                            |

2) [5/30] Si consideri una cache di dimensione 32B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 4 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 125, 170, 167, 245, 183, 119, 235, 163, 288, 309, 310, 308, 213, 196, 377, 166, 362, 233. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso alla cache, riportare i tag contenuti in cache al termine, i bit di modifica (se presenti) e la lista dei blocchi (ovvero il loro indirizzo) via via eliminati durante il rimpiazzamento ed inoltre in corrispondenza di quale riferimento il blocco e' eliminato.

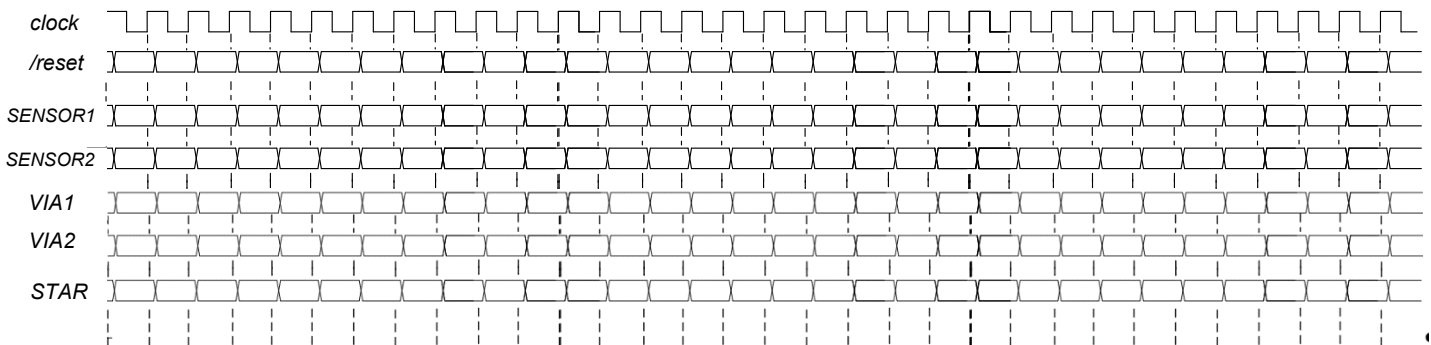
3) [5/30] Descrivere i segnali di controllo per far funzionare il processore di figura: quali funzioni hanno, con riferimento al processore RISC-V?



4) [10/30] Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Mealy Ritardato che realizzi un controllore semaforico che agisce nel modo seguente. Ad un incrocio sono presenti due vie con dei sensori che rilevano la presenza dei veicoli in prossimità dell'incrocio come illustrato in figura. Ogni 5 secondi i semafori possono cambiare a seconda di quello che indicano tali sensori. Se ci sono veicoli su una via soltanto, il semaforo resta verde su tale via e rosso sull'altra. Il semaforo commuta dal verde al giallo solo se non passano più veicoli su uno dei sensori, poi commuta dal giallo al rosso e poi ritorna verde sempre in corrispondenza di un fronte in salita del clock (con periodo 5) secondi e in funzione della presenza o meno di veicoli sui sensori. Gli stimoli di ingresso sono dati dal seguente modulo Verilog Testbench.



**Tracciare il diagramma di temporizzazione** [5/10 punti] come verifica della correttezza dell'unità. Nota: si può svolgere l'esercizio su carta oppure con ausilio del simulatore salvando una copia dell'output (diagramma temporale) e del programma Verilog su USB-drive del docente. Modello del diagramma temporale da tracciare:



```

`timescale 1s/10ms
module Testbench;
reg reset_; initial begin reset_ = 0; #6 reset_ = 1; #300; $stop; end
reg clock; initial clock = 1; always #2.5 clock <= (!clock);
reg SENSOR1, SENSOR2;
wire[1:0] VIA1 = sem.via1;
wire[1:0] VIA2 = sem.via2;
wire[1:0] STAR = sem.STAR;
initial begin
    SENSOR1 <= 0; SENSOR2 <= 0; #0.5
    SENSOR1 <= 1; SENSOR2 <= 0; #10
    SENSOR1 <= 1; SENSOR2 <= 1; #5
    SENSOR1 <= 0; SENSOR2 <= 1; #15
    SENSOR1 <= 1; SENSOR2 <= 0; #15
    SENSOR1 <= 0; SENSOR2 <= 0; #5
    $finish;
end
SEMAFORO sem(VIA1, VIA2, SENSOR1, SENSOR2, clock, reset_);
endmodule
    
```