COMPITO di ARCHITETTURA DEI CALCOLATORI del 05-04-2022

MATRICOLA _____

COGNOME _____

NOME _____

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

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) [12/30] 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).

```
double a[3][3] = {11, 1, 2, 3, 12, 4, 1, 2, 13};
double x[3][3], y[3][3];

double sqr(double x) {
   return(x*x);
}

double sign(double x) {
   if (x<0.0) return(-1);
   if (x==0.0) return(0);
   else return(1);
}

int main () {
   genqr(a,x,y,3);
   print_double(y[2][2]);
}
```

```
void genqr(double a[][3], double q[][3], double r[][3], int s) {
   int i, j, k; double c;
   for(i=0;i<s;++i){

      c=0.0; for(k=0;k<s;++k) c=c+sqr(a[k][i]);
      c=sqrt(c);
      r[i][i] = c;
      for(k=0;k<s;++k) q[k][i]=a[k][i]/c;

      for(j=0;j<i;++j) r[i][j] = 0.0;
      for(j=i+1;j<s;++j) {
         c = 0.0; for(k=0;k<s;++k) c += q[k][i]*a[k][j];
         r[i][j] = c;
         for(k=0;k<s;++k) a[k][j] -= c * q[k][i];
      }
   }
}
```

RISCV Instructions (RV64IMFD) **v210622**

Instruction coding (hexadecimal) opcode+funct3+{funct7,imm}	Instruction	Example	Register operation	Meaning (** instructions available only in RV64, i.e. 64-bit case)
33+0+00/3b+0+00	**add**	**add/addw x5,x6,x7**	x5 ← x6 + x7	Add two operands; exception possible (addw**)
33+0+20/3b+0+20	**subtract**	**sub/subw x5,x6,x7**	x5 ← x6 – x7	Subtracts two operands; exception possible (subw**)
13+0+imm/1b+0+imm	**add immediate**	**addi/addiw x5,x6,100**	x5 ← x6 + 100	Add a constant ; exception possible (addiw**)
33+0+01/3b+0+01	**multiply**	**mul/mulw x5,x6, x7**	x5 ← x6 * x7	(signed/word) Lower 64 bits of 128-bits product (mulw**)
33+01+01	**multiply high**	**mulh x5,x6,x7**	x5 ← x6 * x7	Higher 64bits of 128-bits product
33+4+01/3b+4+01	**division**	**div/divw x5,x6,x7**	x5 ← x6/x7	(signed/word) division (divw**)
33+6+01/3b+6+01	**reminder**	**rem/remw x5,x6,x7**	x5 ← x6 % x7	Reminder of the division (remw**)
33+2+0/33+3+0	**set on less than**	**slt/sltu x5,x6,x7**	if (x6 < x7) x5 ← 1; else x5 ← 0	(signed/unsigned) compare x6 and x7 (less than)
13+2+imm/13+3+imm	**set on less than immediate**	**slti/sltiu x5,x6,100**	if (x6 < 100) x5← 1; else x5 ← 0	(signed/unsigned) compare x6 and 100 (less than)
33+7+0/33+6+0/33+4+0	**and / or / xor**	**and/or/xor x5,x6,x7**	x5← x6&x7 / x6|x7 / x6^ x7	Logical AND/OR/XOR
13+7+imm/13+6+imm/13+4+imm	**and /or / xor immediate**	**andi/ori/xori x5,x6,100**	x5 ← x6&100 / x6|100 / x6^100	Logical AND/OR/XOR register, constant
33+1+0/3b+1+0	**shift left logical**	**sll/sllw x5,x6,x7**	x5 ← x6 << x7	Shift left by register (sllw**)
13+1+imm/1b+1+imm	**shift left logical immediate**	**slli/slliw x5,x6,10**	x5 ← x6 << 10	Shift left by the immediate value (slliw**)
33+5+0/3b+5+0	**shift right logical**	**srl/srlw x5,x6,x7**	x5 ← x6 >> x7	Shift right by register (srlw**)
13+5+imm/1b+5+imm	**shift right logical immediate**	**srli/srliw x5,x6,10**	x5 ← x6 >> 10	Shift left by immediate value (srliw**)
33+5+20/3b+5+20	**shift right arithmetic**	**sra/sraw x5,x6,x7**	x5 ← x6 >> x7 (arith.)	Shift right by register (sign is preserved) (sraw**)
13+5+imm/1b+5+imm	**shift right arithmetic immediate**	**srai/sraiw x5,x6,10**	x5 ← x6 >> 10 (arith.)	Shift right by immediate value (sraiw**)
03+3+imm/03+2+imm/03+0+imm	**load dword / word / byte**	**ld/lw/lb x5,100(x6)**	x5 ← MEM[x6+100]	Data from memory to register
03+6+imm/03+4+imm	**load word / byte unsigned**	**lwu/bu x5,100(x6)**	x5 ← MEM[x6+100]	Data from mem. To reg.; no sign extension (lwu**)
23+3+imm/23+2+imm/23+0+imm	**store dword / word / byte**	**sd/sw/sb x5,100(x6)**	MEM[x6+100] ← x5	Data from register to memory (sw**)
37+imm[31:12] (no funct3)	**load upper immediate**	**lui x5,0x12345**	x5 ← 0x1234'5000	Load most significant 20 bits
PSEUDOINSTRUCTION	**load address**	**la x5,var**	x5 ← &var **(PSEUDO INST.)** load address of 'var' in x5	**REAL: lui x5,H20(&var);ori x5, L12(&var) INST.** (H20=high 20 bit of &var; L12=low 12 bits of &var)
6f+imm[31:12](rd=0) 63+0+imm[11:0](rs1=rs2=0)	**jump/branch**	**j/b label**	PC+=off (off=PC-&label) **(PS.INST.)**	**REAL INST.: jal x0,offset/beq x0,x0,offset**
6f+0+imm[31:12](rd=1,no funct3)	**jump and link (offset)**	**jal label**	x1←(PC+4);PC+=offset **(PS. INST.)**	**REAL INST.: jal x1,offset** (offset=PC-&label)
67+0+imm (rd=0,rs1=1)	**return from procedure**	**ret**	PC←x1 **(PSEUDO INST.)**	**REAL INST.: jalr x0,0(x1)**
67+0+imm	**jump and link register**	**jalr x1, 100(x5)**	x1 ← (PC + 4);PC=x5+100	Procedure return; indirect call
63+0+(imm÷2)/63+1+(imm÷2)	**branch on equal / not-equal**	**beq/bne x5,x6,100**	if (x5 = =/!= x6) PC=PC+100	Equal / Not-equal test; PC relative branch
73+0+0 (rs1=0,rs2=0,rd=0)	**ecall**	**ecall**	SEPC←PC;PC←STVEC;save PL/IE;PL=1;IE=0	Call OS (service number in a7); PL= privilege lev; IE=int.en.
73+0+8 (rs1=0,rs2=2,rd=0)	**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**
53+0+{0,1}/53+0+{4,5}	**floating point add/sub**	**fadd/fsub.{s,d} f0,f1,f2**	f0←f1+f2 / f0←f1-f2	Single or double precision add / subtract
53+0+{8,9}/53+0+{c,d}	**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-regs**	**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**
53+0/1/2+{50,51}	**floating point compare**	**fle/flt/feq.{s,d} x5,f0,f1**	x5← (f0<f1)	Single and double: compare f0 and f1 <=,<,==
53+0+{70,71} (rs2=0)	**move between x (integer) and f regs**	**fmv.x.{s,d} x5,f0**	x5←f0 (no conversion)	Copy (no conversion)
53+0+{78,79} (rs2=0)	**move between f and x regs**	**fmv.{s,d}.x f0,x5**	f0←x5 (no conversion)	Copy (no conversion)
7+2+imm/27+2+imm	**load/store floating point (32bit)**	**flw/fsw f0,0(x5)**	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
7+3+imm/27+3+imm	**load/store floating point (64bit)**	**fld/fsd f0,0(x5)**	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
53+7+21(rs2=0)/53+7+20(rs2=0)	**convert to/from double from/to single**	**fcvt.d.s/fcvt.s.d f0,f1**	f0← (double)f1 / f0← (single)f1	Type conversion
53+7+{60,61} (rs2=0)	**convert to integer from {single,double}**	**fcvt.w.{s,d} x5,f0**	x5← (int)f0	Type conversion
53+7+{68,69} (rs2=0)	**convert to {single,double} from integer**	**fcvt.{s,d}.w f0,x5**	f0← ({single,double})x5	Type conversion
53+0+{2c,2d} (rs2=0)	**square root**	**fsqrt.{s,d} f0,f1**	f0← square root of f1	Single or double square root
53+0/1/2+{10,11}	**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	**s0/fp, 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 ASCIIZ 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	---	---

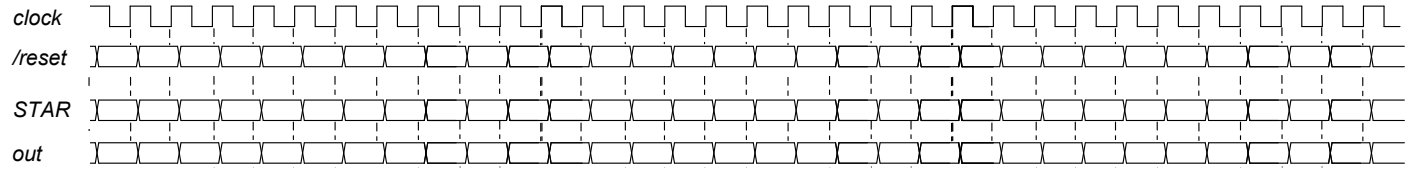
COMPITO di ARCHITETTURA DEI CALCOLATORI del 05-04-2022

2) [4/30] Si consideri una cache di dimensione 128B e a 2 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 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: 277, 1263, 323, 2281, 400, 3321, 275, 1284, 2182, 3201, 4176, 8173, 2176, 9183, 8251, 4176, 2201, 3180, 5171, 7178. 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) [6/30] Disegnare lo schema del datapath del processore RISC-V elementare evidenziando i principali blocchi architetturali, i nomi e le ampiezze in termini di bit dei segnali di ingresso e di uscita di ciascun blocco e degli 6 segnali di controllo. Per i segnali di controllo indicare lo scopo di ciascuno di essi, ricordano che uno di essi è di ampiezza 3-bit.

4) [8/30] Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Moore in cui l'uscita valga 1 per un ciclo di clock ogni tre. Gli stimoli di ingresso sono dati dal seguente modulo Verilog Testbench.

   Tracciare il diagramma di temporizzazione [4/8 punti] come verifica della correttezza dell'unità. Nota: si puo' 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:




```
module Testbench;
  reg reset_; initial begin reset_=0; #7 reset_=1; #300; $stop; end
  reg clock; initial clock=0; always #5 clock<=(!clock);
  wire[1:0] STAR=Xxx.STAR;
  initial begin #200 $finish; end
  XXX Xxx(clock,reset_,  out);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 05-04-2022

SOLUZIONE

ESERCIZIO 1

```
.data
p1:   .double 1.0
m1:   .double -1.0
a:    .double 11.0, 1.0, 2.0, 3.0, 12.0, 4.0, 1.0,
2.0, 13.0
x:    .space 72
y:    .space 72

.text
.globl main
#-----------------------------------------
sqr:
  fmul.d fa0, fa0, fa0 # x*x
  ret
#-----------------------------------------
sign:
  fmv.d.x ft0,x0
  flt.d   t0,fa0,ft0 # x<?0.0
  bne     t0,x0,sign_exitp1 #si: ritorna +1
  feq.d   t0,fa0,ft0 # x=?0.0
  fmv.d   fa0,ft0   # x=0.0
  beq     t0,x0,sign_exitm1 #no: ritorna -1
  ret
sign_exitp1:
  la      t0, p1
  fld     fa0,0(t0) # ritorna +1
  ret
sign_exitm1:
  la      t0, m1
  fld     fa0,0(t0) # ritorna -1
  ret
#-----------------------------------------
genqr:  # a0=a a1=q a2=r a3=s
  #PROLOGO
  addi sp,sp,-28
  sw  ra, 0(sp)
  sw  s0, 4(sp)
  sw  s1, 8(sp)
  sw  s2,12(sp)
  sw  s3,16(sp)
  sw  s4,20(sp)
  sw  s5,24(sp)

# save a0, a1, a2, a3
  mv  s0,a0
  mv  s1,a1
  mv  s2,a2
  mv  s3,a3

# for(i=0;i<s;++i){
  li  s4,0      # i=0
gqr_fori:
  slt t0,s4,s3 # i<?s
  beq t0,x0,gqr_foriend #no: forendi

#  c=0.0; for(k=0;k<s;++k) c=c+sqr(a[k][i]);
  fmv.d.x fs0,x0 #c = 0.0
  li s5,0       # k=0
gqr_fork1:
  slt t0,s5,s3 # k<?s
  beq t0,x0,gqr_fork1end
  mul t0,s5,s3 # k*s
  add t0,t0,s4 # k*s+i
  slli t0,t0,3 # *8 (offset)
  add t1,t0,s0 # &a[k][i]
  fld fa0,0(t1)# a[k][i]
  call sqr
  fadd.d fs0,fs0,fa0 # c+=(.)
  addi s5,s5,1 # ++k
  b gqr_fork1
gqr_fork1end:
#  c=sqrt(c);
  fsqrt.d fs0,fs0 # sqrt(.)
#  r[i][i] = c;
  mul t0,s4,s3 # i*s
  add t0,t0,s4 # i*s+i
  slli t0,t0,3 # *8 (offset)
  add t0,t0,s2 # &r[i][i]
  fsd fs0,0(t0)# r[i][i]

#  for(k=0;k<s;++k) q[k][i]=a[k][i]/c;
  li s5,0       # k=0
gqr_fork2:
  slt t0,s5,s3 # k<?s
  beq t0,x0,gqr_fork2end
  mul t0,s5,s3 # k*s
  add t0,t0,s4 # k*s+i
  slli t0,t0,3 # *8 (offset)
  add t1,t0,s0 # &a[k][i]
  fld ft0,0(t1)# a[k][i]
  fdiv.d ft1,ft0,fs0 # a/c
  add t1,t0,s1 # &q[k][i]
  fsd ft1,0(t1)# q[k][i]=(.)
  addi s5,s5,1 # ++k
  b gqr_fork2
gqr_fork2end:

#  for(j=0;j<i;++j) r[i][j] = 0.0;
  li t2,0       # j=0
  fmv.d.x ft0,x0 # 0.0
gqr_forj1:
  slt t0,t2,s4 # j<?i
  beq t0,x0,gqr_forj1end
  mul t0,s4,s3 # i*s
  add t0,t0,t2 # i*s+j
  slli t0,t0,3 # *8 (offset)
  add t1,t0,s2 # &r[i][j]
  fsd ft0,0(t1)# r[i][j]=0.0
  addi t2,t2,1 #++j
  b gqr_forj1
gqr_forj1end:

#  for(j=i+1;j<s;++j) {
  addi t2,s4,1 # j=i+1
gqr_forj2:
  slt t0,t2,s3 # j<?s
  beq t0,x0,gqr_forj2end

#   c = 0.0; for(k=0;k<s;++k) c += q[k][i]*a[k][j];
  fmv.d.x ft0,x0 # 0.0
  li s5,0       # k=0
gqr_fork3:
  slt t0,s5,s3 # k<?s
  beq t0,x0,gqr_fork3end

  mul t0,s5,s3 # k*s
  add t0,t0,s4 # k*s+i
  slli t0,t0,3 # *8 (offset)
  add t1,t0,s1 # &q[k][i]
  fld ft1,0(t1)# q[k][i]

  mul t0,s5,s3 # k*s
  add t0,t0,t2 # k*s+j
  slli t0,t0,3 # *8 (offset)
  add t1,t0,s0 # &a[k][j]
  fld ft2,0(t1)# a[k][j]

  fmul.d ft1,ft1,ft2 # q*a
  fadd.d ft0,ft0,ft1 #c+= (.)

  addi s5,s5,1 # ++k
  b gqr_fork3
gqr_fork3end:

#   r[i][j] = c;
  mul t0,s4,s3 # i*s
  add t0,t0,t2 # i*s+j
  slli t0,t0,3 # *8 (offset)
  add t1,t0,s2 # &r[i][j]
  fsd ft0,0(t1)# r[i][j]=(.)

#   for(k=0;k<s;++k) a[k][j] -= c * q[k][i];
  li s5,0 #k=0
gqr_fork4:
  slt t0,s5,s3 # k<?s
  beq t0,x0,gqr_fork4end

  mul t0,s5,s3 # k*s
  add t0,t0,s4 # k*s+i
  slli t0,t0,3 # *8 (offset)
  add t1,t0,s1 # &q[k][i]
  fld ft1,0(t1)# q[k][i]

  fmul.d ft2, ft0, ft1 # c*q
  mul t0,s5,s3 # k*s
  add t0,t0,t2 # k*s+j
  slli t0,t0,3 # *8 (offset)
  add t1,t0,s0 # &a[k][j]
  fld ft3,0(t1)# a[k][j]
  fsub.d ft3,ft3,ft2 #a -= c*q
  fsd ft3,0(t1)# a[k][j]=(.)

  addi s5,s5,1 #++k
  b gqr_fork4
gqr_fork4end:

  addi t2,t2,1 #++j
  b gqr_forj2
#  }
gqr_forj2end:

  addi s4,s4,1 # ++i
  b gqr_fori
# }
gqr_foriend:

  #EPILOGO
  lw   s5,24(sp)
  lw   s4,20(sp)
  lw   s3,16(sp)
  lw   s2,12(sp)
  lw   s1, 8(sp)
  lw   s0, 4(sp)
  lw   ra, 0(sp)
  addi sp,sp,28
  ret

#-----------------------------------------
main:
  la a0, a
  la a1, x
  la a2, y
  li a3, 3
  call genqr

#    print_double(y[2][2]);
  la   t0, y
  addi t0,t0,64 # &y[2][2]
  fld  fa0,0(t0)# y[2][2]
  li   a7, 3 # print_double
  ecall
  li   a7, 10 # exit
  ecall
```

```
Run I/O
12.083850926221755
-- program is finished running (0) --
```

ESERCIZIO 2

Sia X il generico riferimento, A=associativita', B=dimensione del blocco, C=capacita' della cache.
Si ricava S=C/B/A=# di set della cache=128/8/2, XM=X/B, XS=XM%S, XT=XM/S.
A=2, B=8, C=128, RP=LRU, Thit=4, Tpen=40, 20 references:

```
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  277   34   4   2   5  0 [2]:1,0 [2]:0,0  [2]:4,-
=== W 1263  157  19   5   7  0 [5]:1,0 [5]:0,0  [5]:19,-
=== R  323   40   5   0   3  0 [0]:1,0 [0]:0,0  [0]:5,-
=== W 2281  285  35   5   1  0 [5]:0,1 [5]:0,0  [5]:19,35
=== R  400   50   6   2   0  0 [2]:0,1 [2]:0,0  [2]:4,6
=== W 3321  415  51   7   1  0 [7]:1,0 [7]:0,0  [7]:51,-
=== R  275   34   4   2   3  1 [2]:1,0 [2]:0,0  [2]:4,6
=== W 1284  160  20   0   4  0 [0]:0,1 [0]:0,0  [0]:5,20
=== R 2182  272  34   0   6  0 [0]:1,0 [0]:0,0  [0]:34,20
=== W 3201  400  50   0   1  0 [0]:0,1 [0]:0,0  [0]:34,50
=== R 4176  522  65   2   0  0 [2]:0,1 [2]:0,0  [2]:4,65
=== W 8173 1021 127   5   5  0 [5]:1,0 [5]:0,0  [5]:127,35
=== R 2176  272  34   0   0  1 [0]:1,0 [0]:0,0  [0]:34,50
=== W 9183 1147 143   3   7  0 [3]:1,0 [3]:0,0  [3]:143,-
=== R 8251 1031 128   7   3  0 [7]:0,1 [7]:0,0  [7]:51,128
=== W 4176  522  65   2   0  1 [2]:0,1 [2]:0,1  [2]:4,65
=== R 2201  275  34   3   1  0 [3]:0,1 [3]:0,0  [3]:143,34
=== W 3180  397  49   5   4  0 [5]:0,1 [5]:0,0  [5]:127,49
=== R 5171  646  80   6   3  0 [6]:1,0 [6]:0,0  [6]:80,-
=== W 7178  897 112   1   2  0 [1]:1,0 [1]:0,0  [1]:112,-
-------------------------------------
                               [4]:0,0 [4]:0,0  [4]:-,-
P1 Nmiss=17   Nhit=3  Nref=20  mrate=0.850000   AMAT=th+mrate*tpen=38
```

LISTA BLOCCHI USCENTI:

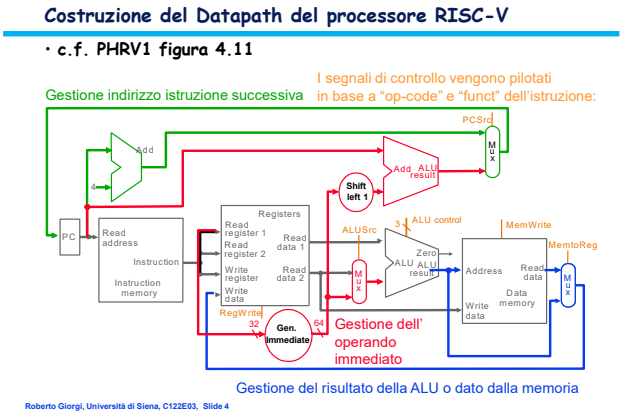
```
(out: XM=40 XT=5  XS=0 )
(out: XM=160 XT=20 XS=0 )
(out: XM=50 XT=6  XS=2 )
(out: XM=157 XT=19 XS=5 )
(out: XM=285 XT=35 XS=5 )
```

CONTENUTI dei SET al termine

COMPITO di ARCHITETTURA DEI CALCOLATORI del 05-04-2022

SOLUZIONE

ESERCIZIO 3



- **RegWrite**: abilita la scrittura nel registro destinazione
- **PCSrc**: sceglie se caricare nel PC, il valore "PC+4" o il target del salto
- **ALUSrc**: sceglie se usare, come operando B della ALU, il secondo valore letto nei registri o il valore immediato
- **AluControl**[2:0]: determina quale operazione deve essere eseguita dalla ALU
- **MemWrite**: abilita la scrittura nella memoria
- **MemtoReg**: sceglie se scrivere nel registro destinazione il valore letto dalla memoria o il risultato prodotto dalla ALU

ESERCIZIO 4

Codice Verilog del modulo da realizzare (possibile soluzione con Moore):

```
module XXX(clock,reset_,  z);
  input       clock, reset_;
  output      z;
  reg [1:0]   STAR;
  parameter   S0='B00, S1='B01, S2='B10;
  always @(reset_==0) #1 STAR<=S0;
  assign z=(STAR==S0)?1:0;
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: STAR<=S1;
      S1: STAR<=S2;
      S2: STAR<=S0;
  endcase
```

Diagramma di Temporizzazione: