COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-02-2022

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) [9/30] 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).

```
int a[5] = {12, 23, 34, 45, 56};

int binPacking(int *a, int size, int n) {
    int bincount = 1, i, s = size;
    for (i = 0; i < n; i++) {
        if (s - *(a + i) > 0) {
            s -= *(a + i);
        } else {
            bincount++;
            s = size;
            i--;
        }
    }
    return(bincount);
}
```

```
int main() {
    int b = binPacking(a, 70, 5);
    print_int(b);
    exit(0);
}
```

Nota: 'int' è un intero a 64 bit.

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

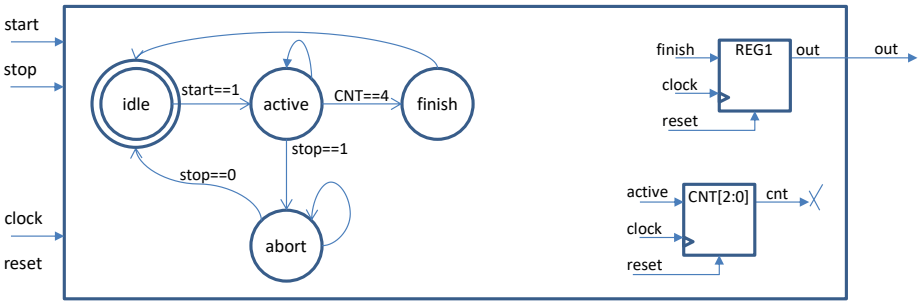
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	fa0=float
read_double	7	---	fa0=double
read_string	8	a0=address of input buffer, a1=max chars to read	---
sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
exit	10	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-02-2022

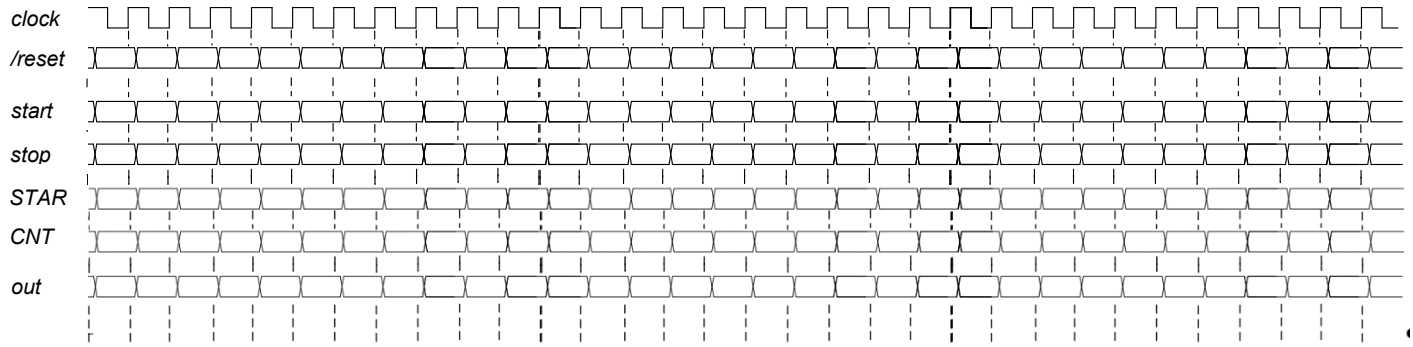
2) [5/30] Si consideri una cache di dimensione 64B e a 2 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 2123, 2339, 2327, 2339, 2328, 2139, 2333, 2354, 2325, 2354, 2322, 2354, 2339, 2126, 2354, 2324, 2554, 2629, 2754, 2828. 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 un possibile schema logico/architetturale del meccanismo di paginazione inversa (nota: dovranno essere esplicitati tutti i blocchi logico/architetturali usando elementi noti visti durante il corso, eccetto il blocco che genera un hash di 20 bit a partire da 52 bit) che riceve in ingresso un indirizzo virtuale a 64 bit, ha una dimensione di pagina pari a 4KiB, uno spazio di indirizzamento fisico a 32 bit.

4) [10/30] Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Moore i cui ingressi e uscita sono descritti in figura; al suo interno la rete è descritta dal diagramma a stati della stessa figura e conterrà due registri: un contatore CNT che usa interi da 0 a 4 e un registro di uscita REG1. 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 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);
  reg start, stop;
  wire[1:0] STAR=Xxx.STAR;
  wire [2:0] CNT=Xxx.CNT;
  initial begin start<=0; stop<=0; wait(reset_==1);
    #20  start=1; #10 start=0;  #100 start=1; #10 start=0;  #20  stop=1;  #10 stop=0;  #50 start=1;  #200
    $finish;
  end
  XXX Xxx(start,stop,clock,reset_,  out);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-02-2022

SOLUZIONE

ESERCIZIO 1

```
.data
a: .dword 12 23 34 45 56

.text
.globl main

binpacking:
   # a0=&a, a1=size, a2=n
   mv   a3, a0       # a3=&a
   mv   a4, a1       # a4=s=size
   li   a0, 1        # bincount=1
   li   t0, 0        # t0=i=0
for_ini:
   slt  t1, t0, a2   # i<?n
   beq  t1, x0, for_end # false-->
   slli t2, t0, 3    # offset(i)=i*8
   add  t2, a3, t2   # a+offset(i)
   ld   t3, 0(t2)    # *(a+i)
   sub  t4, a4, t3   # s - (.)
   slt  t1, x0, t4   # 0 <? (.)
   beq  t1, x0, if_else # false-->
   mv   a4, t4       # s = s - (.)
   b    for_next
if_else:
   addi a0, a0, 1
   mv   a4, a1       # a4=a1=size
   addi t0, t0, -1   # i--
for_next:
   addi t0, t0, 1    # i++
   b    for_ini
for_end:
   ret

main:
   la   a0, a        # &a
   li   a1, 70
   li   a2, 5
   jal binpacking
   li   a7,1         # a0=b
   ecall             # print_int
   li   a7,10
   ecall             # exit
```

Run I/O

```
3
-- 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=64/8/2,     XM=X/B,    XS=XM%S,   XT=XM/S.
A=2, B=8, C=64, RP=FIFO, Thit=4, Tpen=40, 20 references:
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R 2123  265  66   1   3  0 [1]:1,0 [1]:0,0  [1]:66,-
=== W 2339  292  73   0   3  0 [0]:1,0 [0]:0,0  [0]:73,-
=== R 2327  290  72   2   7  0 [2]:1,0 [2]:0,0  [2]:72,-
=== W 2339  292  73   0   3  1 [0]:1,0 [0]:1,0  [0]:73,-
=== R 2328  291  72   3   0  0 [3]:1,0 [3]:0,0  [3]:72,-
=== W 2139  267  66   3   3  0 [3]:0,1 [3]:0,0  [3]:72,66
=== R 2333  291  72   3   5  1 [3]:0,1 [3]:0,0  [3]:72,66
=== W 2354  294  73   2   2  0 [2]:0,1 [2]:0,0  [2]:72,73
=== R 2325  290  72   2   5  1 [2]:0,1 [2]:0,0  [2]:72,73
=== W 2354  294  73   2   2  1 [2]:0,1 [2]:0,1  [2]:72,73
=== R 2322  290  72   2   2  1 [2]:0,1 [2]:0,1  [2]:72,73
=== W 2354  294  73   2   2  1 [2]:0,1 [2]:0,1  [2]:72,73
=== R 2339  292  73   0   3  1 [0]:1,0 [0]:1,0  [0]:73,-
=== W 2126  265  66   1   6  1 [1]:1,0 [1]:1,0  [1]:66,-
=== R 2354  294  73   2   2  1 [2]:0,1 [2]:0,1  [2]:72,73
=== W 2324  290  72   2   4  1 [2]:0,1 [2]:1,1  [2]:72,73
=== R 2554  319  79   3   2  0 [3]:1,0 [3]:0,0  [3]:79,66 (out: XM=291 XT=72 XS=3 )
=== W 2629  328  82   0   5  0 [0]:0,1 [0]:1,0  [0]:73,82
=== R 2754  344  86   0   2  0 [0]:1,0 [0]:0,0  [0]:86,82 (out: XM=292 XT=73 XS=0 )
=== W 2828  353  88   1   4  0 [1]:0,1 [1]:1,0  [1]:66,88
---------------------------------------
P1 Nmiss=10    Nhit=10   Nref=20   mrate=0.500000    AMAT=th+mrate*tpen=24
```

LISTA BLOCCHI USCENTI:

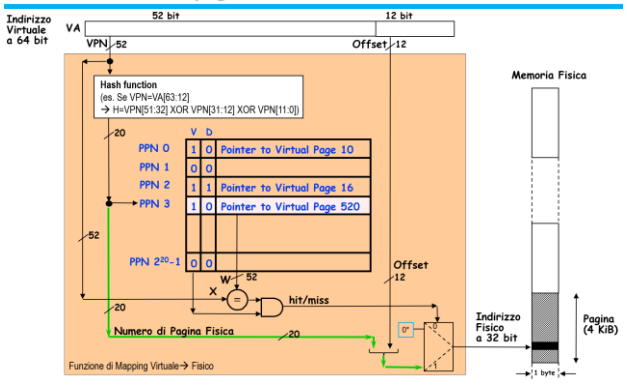
CONTENUTI dei SET al termine

ESERCIZIO 3

Tabella delle Pagine Inversa - descrizione

- Una entry nella tabella delle pagine per ogni pagina fisica
- Vantaggi
  - La dimensione scala con la dimensione della memoria fisica anziche' con quella della memoria virtuale
  - La dimensione e' fissa a prescindere dal numero di processi !
- Chiamata "inversa" perche' le entry sono indicizzate per PPN anziche' per VPN
  - In ogni caso la ricerca avviene fornendo un VPN e ottenendo un PPN
  - Il PPN e' dedotto dalla posizione della VPN trovata
- Tecnica
  - Si applica una funzione di hash al VPN
  - L'hash costituisce un indice nella tabella inversa corrispondente al PPN
  - Poiche' diversi VPN possono produrre lo stesso hash e' necessario un meccanismo per risolvere le collisioni (es. ripetere l'accesso dopo che il VPN e' caricato nella tabella delle pagine al PPN corrispondente)
  - E' possibile che si perda un po' tempo in caso di collisioni sull'hash

Tabella delle pagine inversa - schema architetturale



COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-02-2022

SOLUZIONE

ESERCIZIO 4

Codice Verilog del modulo da realizzare (possibile soluzione con Mealy-Ritardato):

```
module XXX(start,stop,clock,reset_,  out);
  input    clock,reset_;
  input    start,stop;
  output   out;
  reg      REG1;
  reg[1:0] STAR;
  reg[2:0] CNT;
  parameter idle='B00,active='B01,finish='B10,abort='B11;
  always @(reset_==0) #1 begin STAR<=idle;REG1<=0;CNT<=0; end
  assign out=(STAR==finish)?1:0;

  always @(posedge clock) if(reset_==1) #3
  casex(STAR)
    idle:   begin STAR<=(start==1)?active:idle; CNT<=0; end
    active: begin STAR<=(CNT==4)?finish:
                  (stop==1)?abort:active; CNT<=CNT+1;   end
    finish: begin STAR<=idle;                           end
    abort:  begin STAR<=(stop==0)?idle:abort;           end
  endcase
endmodule
```

Diagramma di Temporizzazione: