**COMPITO di ARCHITETTURA DEI CALCOLATORI del 10-02-2016**

**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**

**SVOLGIMENTO DELLA PROVA:**
PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16": es. N.1 + es. N.3 + es. N.4
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1,2,3,5
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo CALCOLATORI es. N.1,2,3.
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.4,5

NOTA: per l'esercizio1 (e analogamente per l'esercizio 4) dovranno essere consegnati due files: il file del programma MIPS (ovvero VERILOG) e il file relativo all'output (screenshot o copy/paste)

1. [18] Utilizzando il simulatore SPIM, codificare in assembly MIPS il seguente codice (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), **rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** (riportate in calce). Al termine della codifica consegnare 2 files: il programma in MIPS e l'output relativo.
   Nota: le funzioni main, seidel2, report, test_convergence e compute sono date (stampate in fondo al testo e fornita copia elettronica).

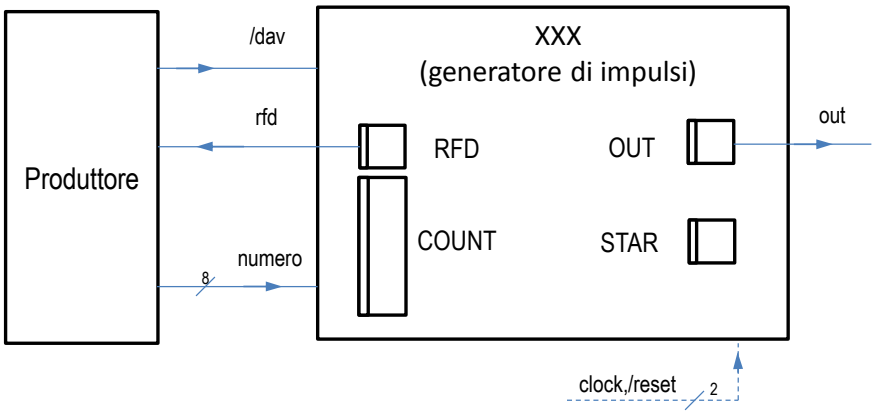
```
#define N 3
#define MAXITER 20
#define EPSILON 0.0001

float A[N*N] = {  4.0, -1.0, -1.0,
                 -2.0,  6.0,  1.0,
                 -1.0,  1.0,  7.0 };
float B[N] = { 3.0, 9.0, -6.0 };
float e2, x[N], y[N], c[N], R[N*N], T[N*N];
int iter = 0;

void setup() {
   int i, j, k;
   float t;
   for (i = 0; i < N; ++i) {
      x[i] = 0.0;
      for (j = 0; j < N; ++j) {
         t = 0.0;
         if (i>j) for (k = j; k < i; ++k)
            t -= A[i*N+k] * T[k*N+j];
         if (i==j) t = 1.0;
         T[i*N+j] = (i<j) ? 0.0 : t / A[i*N+i];
      }
   }
   for (i = 0; i < N; ++i) {
      for (j = 0; j < N; ++j) {
         t = 0.0;
         for (k = 0; k < N; ++k)
            if (k<j) t += T[i*N+k] * A[k*N+j];
         R[i*N+j] = t;
      }
      t = 0.0;
      for (k = 0; k < N; ++k)
         t += B[k] * T[i*N+k];
      c[i] = t;
   }
}
```

```
void seidel2(float *x, float *y, float *a, float c) {
   int j;
   *x = c;
   for (j = 0; j < N; ++j) *x -= a[j] * y[j];
}

void report() {
    int i;
    print_string("X: ");
    for (i = 0; i < N; ++i) {
       print_float(x[i]); print_string(" ");
    }
    print_string(" – iter="); print_int(iter);
    print_string("  e2="); print_float(e2);
    print_string("\n");
}
int test_convergence() {
    int j, r;
    ++iter; e2 = 0;
    for (j = 0; j < N; ++j)
       e2 += (y[j] – x[j]) * (y[j] – x[j]);
    r = (e2 < (EPSILON * EPSILON) || iter == MAXITER);
    report();
    return I;
}

int compute() {
    int i;
    do {
       for (i = 0; i < N; ++i) seidel2(&y[i], x, &R[N*i], c[i]);
       for (i = 0; i < N; ++i) seidel2(&x[i], y, &R[N*i], c[i]);
    } while (!test_convergence());
}

int main()
{
   setup(); compute(); report(); exit(0);
}
```

2. [7] Si consideri una cache di dimensione 128B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 16 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: 177, 1163, 223, 2181, 200, 3221, 175, 1184, 2182, 3201, 4176, 8173, 2176, 9183, 8251, 4176, 2201, 3180, 5171, 7178, 3191, 181. 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] Per la funzione seidel2 dell'esercizio 1 determinare il tempo di esecuzione del codice assumendo che giri su un processore MIPS senza pipeline con frequenza di clock pari a 5 GHz e con i seguenti valori per il CPI di ciascuna categoria di istruzioni: aritmetico-logiche-salti 1, branch 3, load-store 10, floating-point 5.

4) [10] L'unità XXX preleva dal Produttore un byte 'numero' (8-bit) e genera sull'uscita 'out' un impulso con una durata N cicli di clock essendo N il numero naturale prelevato su 'numero'. Per gestire 'numero' viene effettuato l'handshake tramite la coppia di segnali '/dav' (data valid – attivo basso) e 'rfd' (ready for data).

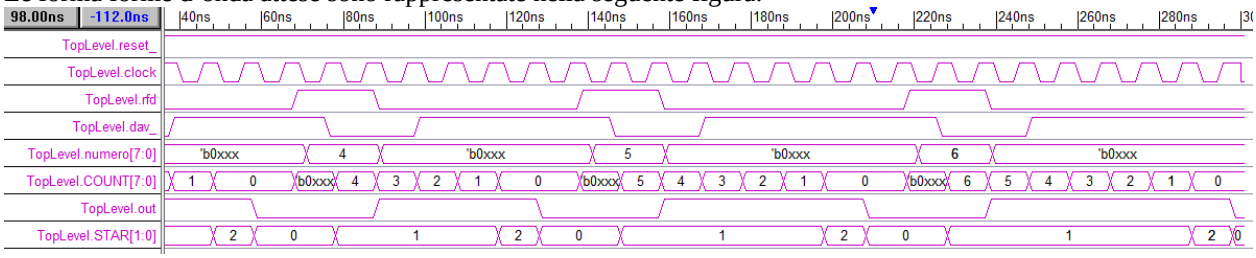



Handshake /dav, rfd: partendo da una condizione inziale in cui /dav è 1 (per indicare che nessun nuovo byte è fornito dal produttore attraverso il segnale 'numero') e rfd è 1 (per indicare che la rete XXX è disponibile a prelevare ed elaborare un nuovo byte), il produttore agisce per primo presentanto un nuovo byte attraverso il segnale 'numero' e notifica questo fatto ponendo /dav a 0. Dopodiché la rete XXX preleva il byte e pone rfd=0 (per indicare che non è ulteriormente disponibile a prelevare altri byte) e inizia ad elaborare il byte prelevato. Il produttore può allora riportare /dav a 1 e attendere che il consumatore, utilizzando tale byte, riporti rfd a 1 (ripristino delle condizioni iniziali).

Uso da parte di XXX del byte prelevato: XXX intepreta il byte prelevato come un numero naturale N e porta a 1 la variabile di uscita out (che normalmente vale 0) per N periodi di clock se N != 0 e per 256 periodi di clock se N=0.

Si descriva e si sintetizzi l'unità XXX e se ne tracci l'evoluzione nell'ipotesi che il Produttore fornisca le rappresentazioni di (numero)= (4), (5), (6) indicando chiaramente nel grafico tali rappresentazioni (il codice Verilog del produttore e del top-module e' fornito nell'ultima pagina del testo).

Le forma forme d'onda attese sono rappresentate nella seguente figura:




5) [12] Sintetizzare una rete sequenziale utilizzando il modello di Moore con un ingresso X su due bit e una uscita Z su singolo bit che si comporta nel seguente modo: se X1=0 e all'ingresso X0 si presenta la successione di quattro bit 0101 allora l'uscita diventa 1; se X1=1 e all'ingresso X0 si presenta la successione di quattro bit 1011, l'uscita diventa 1; in tutti gli altri casi l'uscita è zero. Se X1 cambia da 0 a 1 o viceversa, la rete abbandona l'esame della successione corrente e passa ad esaminare l'altra. Rappresentare la macchina a stati finiti per tale riconoscitore, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.

**Instructions**

| Instruction | Example | | Meaning | Comments |
|---|---|---|---|---|
| add | `add` | `$1,$2,$3` | $1 = $2 + $3 | 3 operands; exception possible |
| subtract | `sub` | `$1,$2,$3` | $1 = $2 - $3 | 3 operands; exception possible |
| add immediate | `addi` | `$1,$2,100` | $1 = $2 + 100 | + constant ; exception possible |
| subtract immediate | `subi` | `$1,$2,100` | $1 = $2 – 100 | - constant; exception possible |
| multiplication | `mult` | `$1, $2` | Hi,Lo= $1 x $2 | 64-bit Signed Product ; result in Hi,Lo |
| division | `div` | `$1, $2` | Hi= $1 % $2, Lo = $1 / $2 | Signed division |
| move from Hi | `mfhi` | `$1` | $1 = Hi | Create copy of Hi |
| move from Lo | `mflo` | `$1` | $1 = Lo | Create copy of Lo |
| and | `and` | `$1,$2,$3` | $1 = $2 & $3 | 3 register operands; Logical AND |
| or | `or` | `$1,$2,$3` | $1 = $2 \| $3 | 3 register operands; Logical OR |
| nor | `nor` | `$1,$2,$3` | $1 = !($2 \| $3) | 3 register operands; Logical NOR |
| xor | `xor` | `$1,$2,$3` | $1 = $2 ^ $3 | 3 register operands; Logical XOR |
| and immediate | `andi` | `$1,$2,100` | $1 = $2 & 100 | Logical AND register, constant |
| or immediate | `ori` | `$1,$2,100` | $1 = $2 \| 100 | Logical OR register, constant |
| xor immediate | `xori` | `$1,$2,100` | $1 = $2 ^ 100 | Logical XOR register, constant |
| shift left logical | `sll` | `$1,$2,10` | $1 = $2 << 10 | Shift left by constant |
| shift right logical | `srl` | `$1,$2,10` | $1 = $2 >> 10 | Shift right by constant |
| load word | `lw` | `$1,100($2)` | $1 = Memory[$2+100] | Data from memory to register |
| load byte | `lb` | `$1,100($2)` | $1 = Memory[$2+100] | Data from memory to register |
| load byte unsigned | `lbu` | `$1,100($2)` | $1 = Memory[$2+100] | Data from mem. To reg.; no sign extension |
| store word | `sw` | `$1,100($2)` | Memory[$2+100] = $1 | Data from register to memory |
| store byte | `sb` | `$1,100($2)` | Memory[$2+100] = $1 | Data from register to memory |
| load address | `la` | `$1,var` | $1 = &var | Load variable address |
| branch unconditional | `b` | `100` | go to PC+4+100 | PC relative branch |
| branch on equal | `beq` | `$1,$2,100` | if ($1 = = $2) go to PC+4+100 | Equal test; PC relative branch |
| branch on not equal | `bne` | `$1,$2,100` | if ($1 != $2) go to PC+4+100 | Not equal test; PC relative |
| set on less than | `slt` | `$1,$2,$3` | if ($2 < $3) $1 = 1; else $1 = 0 | Compare less than; 2`s complement |
| set on less than immediate | `slti` | `$1,$2,100` | if ($2 < 100) $1 = 1; else $1 = 0 | Compare < constant; 2`s complement |
| set on less than unsigned | `sltu` | `$1,$2,$3` | if ($2 < $3) $1 = 1; else $1 = 0 | Compare less than; natural number |
| set on less than imm.unsigned | `sltiu` | `$1,$2,100` | if ($2 < 100) $1 = 1; else $1 = 0 | Compare constant; natural number |
| jump | `j` | `10000` | go to 10000 | Jump to target address |
| jump register | `jr` | `$31` | go to $31 | For switch, procedure return |
| jump and link | `jal` | `10000` | $31 = PC + 4;go to 10000 | For procedure call |
| no operation | `nop` | | Do nothing | Do nothing |
| load-linked | `ll` | `$1,100($2)` | $1=Memory[$2+100] | Read and start to monitor the given memory location |
| store-conditional | `sc` | `$1,100($2)` | Memory[$2+100]=$1 or → | return 0 if a coherence action happens since the previous ll ($1 must be different from 0) |
| add.s add.d | `add.x` | `$f0,f2,$f4` | $f0=$f2+$f4 | Single and double precision add |
| sub.s sub.d | `add.x` | `$f0,f2,$f4` | $f0=$f2-$f4 | Single and double precision subtraction |
| mul.s mul.d | `mul.x` | `$f0,f2,$f4` | $f0=$f2*$f4 | Single and double precision multiplication |
| div.s div.d | `div.x` | `$f0,f2,$f4` | $f0=$f2/$f4 | Single and double precision division |
| mov.s mov.d | `mov.x` | `$f0,$f2` | $f0←$f2 | Single and double precision move |
| abs.s abs.d | `abs.x` | `$f0,$f2` | $f0=ABS($f2) | Single and double precision absolute value |
| neg.s neg.d | `neg.x` | `$f0,$f2` | $f0= – ($f2) | Single and double precision opposite value |
| c.lt.s c.lt.d (eq,ne,le,gt,ge) | `c.lt.x` | `$f0,$f2` | Temp=($f0<$f2) | Single and double: compare $f0 and $f2 <,=,!=,<=,>,>= |
| mtc1 | `mtc1` | `$1,$f2` | $f2=$1 | Data from gen.reg. $1 to C1 reg. $f2 (no conversion) |
| mfc1 | `mfc1` | `$f1,$1` | $1=$f2 | Data from gen.reg. to C1 reg. (no conversion) |
| branch on false | `bc1f` | `label` | If (Temp = = false) go to label | Temp is 'Condition-Code' |
| branch on true | `bc1t` | `label` | If (Temp = = true) go to label | Temp is 'Condition-Code' |
| load floating point (32bit) | `lwc1` | `$f0,0($1)` | $f0←Memory[$1] | Data from FP (C1) register to memory |
| store floating point (32bit) | `swc1` | `$f0,0($1)` | Memory[$1]←$f0 | Data from memory to FP (C1) register |
| convert single into double | `cvt.d.s` | `$f0,$f2` | $f0=(double)$f2 | Also cvt.s.d (viceversa) |
| convert single into integer | `cvt.w.s` | `$f1,$f0` | $f1=(int)$f0 | Also cvt.s.w (viceversa) |

**Register Usage**

| Name | Reg. Num. | Usage |
|---|---|---|
| **$zero** | 0 | The constant value 0 |
| **$s0-$s7** | 16-23 | Saved |
| **$t0-$t9** | 8-15,24-25 | Temporaires |
| **$a0-$a3** | 4-7 | Arguments |

| Name | Reg.Num. | Usage |
|---|---|---|
| **$v0-$v1** | 2-3 | Results |
| **$fp, $sp** | 30,29 | frame pointer, stack pointer |
| **$ra, $gp** | 31,28 | return address, global pointer |
| **$k0-$k1** | 26,27 | Kernel usage |

| Reg. Num. | Usage |
|---|---|
| **$f0, $f2** | Return values |
| **$f12,$f14** | Function arguments |
| **$f20,$f22,$f24,$f26,$f28,$f30** | Saved registers |
| **$f4,$f6,$f8,$f10,$f16,$f18** | Temporaries registers |

**System calls**

| Service Name | Service Num. ($v0) | INPUT Arguments | OUTPUT Arguments |
|---|---|---|---|
| **print_int** | 1 | $a0=integer to print | --- |
| **print_float** | 2 | $f12=float to print | --- |
| **print_double** | 3 | ($f12,$f13)=double to print | --- |
| **print_string** | 4 | $a0=address of ASCIIZ string to print | --- |
| **read_int** | 5 | --- | $v0=integer |
| **read_float** | 6 | --- | $f0=float |
| **read_double** | 7 | --- | $f0-f1=double |
| **read_string** | 8 | $a0=address of input buffer, $a1=max characters to read | --- |
| **sbrk** | 9 | $a0=Number of bytes to be allocated | $v0=pointer to the allocated memory |
| **exit** | 10 | --- | --- |

```
module TopLevel;
  reg reset_; initial begin reset_=0;  #1 reset_=1; #300; $stop; end
  reg clock ; initial clock =0; always #5clock <=(!clock);
  wire rfd, dav_;
  wire[7:0] numero;
  wire[7:0] COUNT=Xxx.COUNT; wire out;
  wire[1:0] STAR=Xxx.STAR;
  XXX Xxx(rfd, dav_,numero, out, clock,reset_);
  Produttore  PRO(rfd,dav_,numero);
endmodule


module Produttore(rfd, dav_,numero);
 input         rfd;
 output        dav_;
 output [7:0]  numero;
 reg DAV_;    assign dav_=DAV_;
 reg [2:0]    APP1_X, APP2_X; assign numero=APP1_X;
 initial begin APP2_X='B00000011; DAV_=1; end
 always
   begin #5; wait(rfd==1); #3 APP1_X=APP2_X; APP2_X=APP2_X+1;
         #5 DAV_=0; wait(rfd==0); #1 APP1_X='HXX; #9 DAV_=1;end
endmodule
```

---

```
seidel2:
#________NO CALL FRAME___________
# INPUT: a0=*x, a1=*y, a2=*a, f12=c
swc1 $f12, 0($a0)        # store c into *x
add  $t0, $0, $0            # t0=j=0
Seidel2_start_for1:
  slti $t9, $t0, 3        # N <? 3
  beq $t9, $0, Seidel2_end_for1
  #_____________________________________
  sll   $t3, $t0, 2     # j*sizeof(float)
  add   $t4, $t3, $a2    # &a[j]
  lwc1  $f4, 0($t4)
  add   $t4, $t3, $a1    # &y[j]
  lwc1  $f6, 0($t4)
  mul.s $f8, $f4, $f6   # a[j]*y[j]
  lwc1  $f10, 0($a0)    # load *x
  sub.s $f10, $f10, $f8   # *x-=
  swc1  $f10, 0($a0)    # store *x
  #_____________________________________
  addi $t0, $t0, 1
  j Seidel2_start_for1
Seidel2_end_for1:
jr $ra

report:
#________CALL FRAME___________
# saved variables: s0       4B
# ra                        4B
#____________________Totale  8B
addi $sp, $sp, -8
sw   $s0, 4($sp)
sw   $ra, 0($sp)
addi $v0, $0, 4      # print "X: "
la   $a0, x_str
syscall
add  $s0, $0, $0    # s0=j=0
Report_start_for1:
  slti $t9, $s0, 3    # j <? N
  beq  $t9, $0, Report_end_for1
  #_____________________________________
  sll  $t3, $s0, 2  # t3=j*sizof(float)
  la   $t4, x
  add  $t4, $t4, $t3  # t4= x[j]
  lwc1 $f12, 0($t4)
  addi $v0, $0, 2    # print x[j]
  syscall
  addi $v0, $0, 4    # print " "
  la $a0, spazio
  syscall
  #_____________________________________
  addi $s0, $s0, 1
  j Report_start_for1
Report_end_for1:
addi $v0, $0, 4    # print " - iter="
la $a0, iter_str
syscall
addi $v0, $0, 1    # print iter
la $t0, iter
lw $a0, 0($t0)
syscall
addi $v0, $0, 4    # print "  e2="
la $a0, e_str
syscall
addi $v0, $0, 2
la $t0, e2      # print e2
lwc1 $f12, 0($t0)
syscall
addi $v0, $0, 4    # print "\n"
la $a0, ret
syscall
lw    $s0, 4($sp)
lw    $ra, 0($sp)
addi $sp, $sp, 8
jr $ra

test_convergence:
#________CALL FRAME___________
# ra            4B
#____________________Totale  4B
addi $sp, $sp, -4
sw   $ra, 0($sp)
la   $t0, iter
lw   $t1, 0($t0)
addi $t1, $t1, 1    # ++iter
sw   $t1, 0($t0)
la   $t0, e2      # &e2
sw   $0, 0($t0)      # e2=0
add  $t0, $0, $0    # t0=j=0
Test_Convergence_start_for:
  slti $t9, $t0, 3    # j <? N
  beq  $t9, $0, Test_Convergence_end_for
  #_____________________________________
  la     $t2,  x
  la     $t3,  y
  sll   $t5,  $t0,  2    # j*sizeof(float)
  add   $t2,  $t2,  $t5  # &x[j]
  lwc1  $f4,  0($t2)
  add   $t3,  $t3,  $t5  # &y[j]
  lwc1  $f6,  0($t3)
  sub.s $f8,  $f6,  $f4  # y-x
  mul.s $f8,  $f8,  $f8   # ()^2
  la     $t6,  e2
  lwc1  $f10, 0($t6)    # load e2
  add.s $f10, $f10, $f8  # e2+=()
  swc1  $f10, 0($t6)    # store e2
  #_____________________________________
  addi $t0, $t0, 1
  j Test_Convergence_start_for
Test_Convergence_end_for:
jal report
la     $t6,  e2
lwc1   $f10, 0($t6)
la     $t0, EPSILON
lwc1   $f4, 0($t0)
mul.s  $f4, $f4, $f4   # EPSILON^2
c.lt.s $f10, $f4      # e2 < eps
add    $v0, $0, $0    # r=0
bc1f   Test_Convergence_c1_f
#_____________________________________
  addi   $v0, $0, 1
  j      Test_Convergence_c2_end
Test_Convergence_c1_f:
  #_____________________________________
  la   $t0, iter
  lw   $t0, 0($t0)
  slti $t2, $t0, 20   # iter ==? MAXITER(20)
  bne  $t2, $0, Test_Convergence_c2_end
  addi $v0, $0, 1
  #_____________________________________
#_____________________________________
Test_Convergence_c2_end:
lw   $ra, 0($sp)
addi $sp, $sp, 4
jr   $ra

compute:
#________CALL FRAME___________
# saved variables: s0       4B
# ra                        4B
#____________________Totale 8B
addi $sp, $sp, -8
sw    $s0, 4($sp)
sw    $ra, 0($sp)
Compute_start_do:
  add $s0, $0, $0      # s0=i=0
  Compute_start_for_1:
    slti $t9, $s0, 3    # i <? N
    beq  $t9, $0, Compute_end_for_1
    #_____________________________________
    sll  $t3, $s0, 2   # t3=i*sizeof(float)
    la   $a0, y
    add  $a0, $a0, $t3  #
a0=&y+i*sizeof(float)
    la   $a1, x      # a1=&x
    addi $t1, $0, 3    # N=3
    mult $t3, $t1    # N*i*sizeof(float)
    mflo $t0      #
    la   $a2, R
    add  $a2, $a2, $t0  #
a2=R+N*i*sizeof(float)
    la   $t4, c      # t4=&c[0]
    add  $t4, $t4, $t3  # t4=&c[i]
    lwc1 $f12, 0($t4)
    jal seidel2
    #_____________________________________
    addi $s0, $s0, 1
    j Compute_start_for_1
  Compute_end_for_1:
  add $s0, $0, $0      # s0=i=0
  Compute_start_for_2:
    slti $t9, $s0, 3    # i <? N
    beq  $t9, $0, Compute_end_for_2
    #_____________________________________
    sll  $t3, $s0, 2  # t3=i*sizeof(float)
    la   $a0, x
    add  $a0, $a0, $t3  #
a0=&x+i*sizof(float)
    la   $a1, y      # a1=&y
    addi $t1, $0, 3    # N=3
    mult $t3, $t1    # N*i*sizeof(float)
    mflo $t0      #
    la   $a2, R
    add  $a2, $a2, $t0  #
a2=R+N*i*sizeof(float)
    la   $t4, c      # t4=&c[0]
    add  $t4, $t4, $t3  # t4=&c[i]
    lwc1 $f12, 0($t4)
    jal seidel2
    #_____________________________________
    addi $s0, $s0, 1
    j Compute_start_for_2
  Compute_end_for_2:
  jal test_convergence # Output:v0
  beq $v0, $0, Compute_start_do
Compute_end_do:
lw  $s0, 4($sp)
lw  $ra, 0($sp)
addi $sp, $sp, 8
jr $ra

main:
#________NO CALL FRAME___________
jal setup
jal compute
jal report
addi $v0, $0, 10
syscall
```

**COMPITINO di ARCHITETTURA DEI CALCOLATORI del 10-02-2016**
**SOLUZIONE**

ESERCIZIO 1)

```
.data
A:      .float  4.0, -1.0, -1.0
        .float -2.0,  6.0,  1.0
        .float -1.0,  1.0,  7.0
B:      .float  3.0,  9.0, -6.0
e2:     .space   4
x:      .space  12
y:      .space   12
c:      .space   12
R:      .space   36
T:      .space   36
iter:    .word   0
spazio:    .asciiz "  "
ret:    .asciiz "\n"
iter_str:  .asciiz " - iter ="
e_str:    .asciiz "  e2="
x_str:    .asciiz "X: "
EPSILON:  .float 0.00001

.text
.globl main

setup:
#________NO CALL FRAME___________
# t2=i, t3=j, t4=k, f0=t, t5=i*N
addi $t6, $0, 3  # 3
la   $t7, x    # &x
la   $t8, A    # &A
la   $t9, T     # &T

add $t2, $0, $0      # t2=i=0
Setup_start_for1:
  slti $t0, $t2, 3  # i <? N(3)
  beq  $t0, $0, Setup_end_for1
  #_____________________________
  mult  $t2, $t6    # i*N
  mflo  $t5      # t5=i*N
  sll   $t0, $t2, 2   # t0= i*sizeof(float)
  add   $t1, $t7, $t0 # &x[i]
  sw    $0,  0($t1)  # x[i]=0.0
  add   $t3, $0, $0      # t3=j=0
  Setup_start_for2:
    slti $t0, $t3, 3  # j <? N(3)
    beq  $t0, $0, Setup_end_for2
    #_____________________________
    mtc1  $0, $f0    # t=0
    slt   $t0, $t3, $t2 # j <? i
    beq    $t0, $0,  Setup_fine_if1
    add   $t4, $t3, $0  # t4=k=j
    Setup_start_for3:
      slt   $t0, $t4, $t2  # k <? i
      beq   $t0, $0, Setup_end_for3
      #_____________________________
      add   $t0, $t5, $t4 # i*N+k
      sll   $t0, $t0, 2   # t0= i*sizeof(float)
      add    $t1, $t8, $t0  # &A[i,k]
      lwc1  $f2, 0($t1)  # f2=A[i,k]
      mult  $t4, $t6    # k*N
      mflo  $t0      # t0=k*N
      add   $t0, $t0, $t3 # k*N+j
      sll   $t0, $t0, 2   # t0= i*sizeof(float)
      add    $t1, $t9, $t0  # &T[k,j]
      lwc1  $f4, 0($t1)  # f4=T[k,j]
      mul.s $f6, $f2, $f4 # f6=f2*f4
      sub.s $f0, $f0, $f6 # f0+=f6
      #_____________________________
      addi  $t4, $t4, 1   # ++k
      j Setup_start_for3
    Setup_end_for3:
    Setup_fine_if1:
    Setup_start_if2:
      bne $t2, $t3, Setup_fine_if2 # i ==? j
      #_____________________________
      addi  $t0, $0, 1
      mtc1  $t0, $f8
      cvt.s.w   $f0, $f8 # t=1.0
      #_____________________________
    Setup_fine_if2:
    Setup_start_if3:
      slt $t0, $t2, $t3 # i <? j
      beq $t0, $0, Setup_else_if3
      #_____________________________
      mtc1 $0, $f12  # f12=0.0
      j Setup_fine_if3
      #_____________________________
    Setup_else_if3:
      #_____________________________
      add   $t0, $t5, $t2 # t0=i*N+i
      sll   $t0, $t0, 2   # t0= i*sizeof(float)
      add    $t1, $t8, $t0  # &A[i,k]
      lwc1  $f2, 0($t1)  # f2=A[i,i]
      div.s $f12,$f0, $f2 # f12=f0/f2
      #_____________________________
    Setup_fine_if3:
      add   $t0, $t5, $t3 # t0=i*N+j
      sll   $t0, $t0, 2   # t0= i*sizeof(float)
      add    $t1, $t9, $t0  # &T[i,j]
      swc1  $f12, 0($t1)  # T[i,j]=f12
      #_____________________________
      addi $t3, $t3, 1    # ++j
      j Setup_start_for2
  Setup_end_for2:
  #_____________________________
  addi $t2, $t2, 1  # ++i
  j Setup_start_for1
Setup_end_for1:

la   $t7, B      # &B
add  $t2, $0, $0  # t2=i=0
Setup_start_for4:
  slti $t0, $t2, 3  # i <? N(3)
  beq  $t0, $0, Setup_end_for4
  #_____________________________
  mult  $t2, $t6    # i*N
  mflo  $t5      # t5=i*N
  add   $t3, $0, $0    # t3=j=0
  Setup_start_for5:
    slti $t0, $t3, 3  # j <? N(3)
    beq  $t0, $0, Setup_end_for5
    #_____________________________
    mtc1  $0, $f0    # t=0
    add  $t4, $0, $0  # t4=k=0
    Setup_start_for6:
      slti $t0, $t4, 3  # k <? N(3)
      beq  $t0, $0, Setup_end_for6
      #_____________________________
      Setup_start_if4:
        slt $t0, $t4, $t3 # k <? j
        beq $t0, $0, Setup_end_if4
        #_____________________________
        add  $t0, $t5, $t4  # i*N+k
        sll  $t0, $t0, 2     # *sizeof(float)
        add  $t1, $t9, $t0   # &T[i,k]
        lwc1 $f2, 0($t1)  # f2=T[i,k]
        mult $t4, $t6    # k*N
        mflo $t0
        add  $t0, $t0, $t3  # k*N+j
        sll  $t0, $t0, 2     # *sizeof(float)
        add  $t1, $t8, $t0   # &A[i,k]
        lwc1 $f4, 0($t1)  # f4=A[i,k]
        mul.s $f6,$f2,$f4   # f6=f2*f4
        add.s $f0,$f0,$f6  # t+=f6
        #_____________________________
      Setup_end_if4:
      #_____________________________
      addi $t4, $t4, 1  # ++k
      j Setup_start_for6
    Setup_end_for6:
    add   $t0, $t5, $t3 # t0=i*N+j
    sll   $t0, $t0, 2   # t0= i*sizeof(float)
    la    $t1, R        # &R
    add   $t1, $t1, $t0 # &R[i,j]
    swc1  $f0, 0($t1)  # R[i,j]=t
    #_____________________________
    addi $t3, $t3, 1  # ++j
    j Setup_start_for5
  Setup_end_for5:
  mtc1  $0, $f0    # t=0
  add   $t4, $0, $0  # t4=k=0
  Setup_start_for7:
    slti  $t0, $t4, 3  # k <? N(3)
    beq   $t0, $0, Setup_end_for7
    #_____________________________
    sll  $t0, $t4, 2     # k*sizeof(float)
    add  $t1, $t7, $t0   # &B[k]
    lwc1 $f2, 0($t1)  # f2=B[k]
    add  $t0, $t5, $t4  # i*N+k
    sll  $t0, $t0, 2     # *sizeof(float)
    add  $t1, $t9, $t0   # &T[i,k]
    lwc1 $f4, 0($t1)  # f4=T[i,k]
    mul.s $f6, $f2, $f4 # f6=f2*f4
    add.s $f0, $f0, $f6 # t+=f6
    #_____________________________
    addi $t4, $t4, 1  # ++k
    j Setup_start_for7
  Setup_end_for7:
  sll $t0, $t2, 2     # i*sizeof(float)
  la  $t1, c          # &c
  add $t1, $t1, $t0   # &c[i]
  swc1 $f0, 0($t1)  # c[i] = t
  #_____________________________
  addi $t2, $t2, 1
  j Setup_start_for4
Setup_end_for4:
jr $ra
```

**COMPITINO di ARCHITETTURA DEI CALCOLATORI del 10-02-2016**
**SOLUZIONE**

ESERCIZIO 2)
A=4,B=16,C=128.
Si ricava S=C/B/A=# di set della cache=128/16/4=2, XM=X/B, XS=XM%S, XT=XM/S:

```
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  177   11   5   1   1  0 [1]:3,0,0,0 [1]:0,0,0,0  [1]:5,-,-,-
=== W 1163   72  36   0  11  0 [0]:3,0,0,0 [0]:0,0,0,0  [0]:36,-,-,-
=== R  223   13   6   1  15  0 [1]:2,3,0,0 [1]:0,0,0,0  [1]:5,6,-,-
=== W 2181  136  68   0   5  0 [0]:2,3,0,0 [0]:0,0,0,0  [0]:36,68,-,-
=== R  200   12   6   0   8  0 [0]:1,2,3,0 [0]:0,0,0,0  [0]:36,68,6,-
=== W 3221  201 100   1   5  0 [1]:1,2,3,0 [1]:0,0,0,0  [1]:5,6,100,-
=== R  175   10   5   0  15  0 [0]:0,1,2,3 [0]:0,0,0,0  [0]:36,68,6,5
=== W 1184   74  37   0   0  0 [0]:3,0,1,2 [0]:0,0,0,0  [0]:37,68,6,5 (out: XM=72 XT=36 XS=0 )
=== R 2182  136  68   0   6  1 [0]:2,3,0,1 [0]:0,0,0,0  [0]:37,68,6,5
=== W 3201  200 100   0   1  0 [0]:1,2,3,0 [0]:0,0,0,0  [0]:37,68,100,5 (out: XM=12 XT=6  XS=0 )
=== R 4176  261 130   1   0  0 [1]:0,1,2,3 [1]:0,0,0,0  [1]:5,6,100,130
=== W 8173  510 255   0  13  0 [0]:0,1,2,3 [0]:0,0,0,0  [0]:37,68,100,255 (out: XM=10 XT=5  XS=0 )
=== R 2176  136  68   0   0  1 [0]:0,3,1,2 [0]:0,0,0,0  [0]:37,68,100,255
=== W 9183  573 286   1  15  0 [1]:3,0,1,2 [1]:0,0,0,0  [1]:286,6,100,130 (out: XM=11 XT=5  XS=1 )
=== R 8251  515 257   1  11  0 [1]:2,3,0,1 [1]:0,0,0,0  [1]:286,257,100,130 (out: XM=13 XT=6  XS=1 )
=== W 4176  261 130   1   0  1 [1]:1,2,0,3 [1]:0,0,0,1  [1]:286,257,100,130
=== R 2201  137  68   1   9  0 [1]:0,1,3,2 [1]:0,0,0,1  [1]:286,257,68,130 (out: XM=201 XT=100 XS=1 )
=== W 3180  198  99   0  12  0 [0]:3,2,0,1 [0]:0,0,0,0  [0]:99,68,100,255 (out: XM=74 XT=37 XS=0 )
=== R 5171  323 161   1   3  0 [1]:3,0,2,1 [1]:0,0,0,1  [1]:161,257,68,130 (out: XM=573 XT=286 XS=1 )
=== W 7178  448 224   0  10  0 [0]:2,1,3,0 [0]:0,0,0,0  [0]:99,68,224,255 (out: XM=200 XT=100 XS=0 )
=== R 3191  199  99   1   7  0 [1]:2,3,1,0 [1]:0,0,0,1  [1]:161,99,68,130 (out: XM=515 XT=257 XS=1 )
=== W  181   11   5   1   5  0 [1]:1,2,0,3 [1]:0,0,0,0  [1]:161,99,68,5 (out: XM=261 XT=130 XS=1 )
-------------------------------------
P1 Nmiss=19   Nhit=3  Nref=22  mrate=0.863636   AMAT=38.5455
```

Situazione finale cache:
```
[0]:99,68,224,255
[1]:161,99,68,5
```

Lista blocchi uscenti:
```
(out: XM=72 XT=36 XS=0 )
(out: XM=12 XT=6  XS=0 )
(out: XM=10 XT=5  XS=0 )
(out: XM=11 XT=5  XS=1 )
(out: XM=13 XT=6  XS=1 )
(out: XM=201 XT=100 XS=1 )
(out: XM=74 XT=37 XS=0 )
(out: XM=573 XT=286 XS=1 )
(out: XM=200 XT=100 XS=0 )
(out: XM=515 XT=257 XS=1 )
(out: XM=261 XT=130 XS=1 )
```

ESERCIZIO 3)

```
seidel2:
swc1 $f12, 0($a0)        # store c into *x
add  $t0, $0, $0           # t0=j=0
```
BB1

```
Seidel2_start_for1:
  slti $t9, $t0, 3        # N <? 3
  beq $t9, $0, Seidel2_end_for1
```
BB2

```
  sll   $t3, $t0, 2     # j*sizeof(float)
  add   $t4, $t3, $a2    # &a[j]
  lwc1  $f4, 0($t4)
  add   $t4, $t3, $a1    # &y[j]
  lwc1  $f6, 0($t4)
  mul.s $f8, $f4, $f6   # a[j]*y[j]
  lwc1  $f10, 0($a0)    # load *x
  sub.s $f10, $f10, $f8   # *x-=
  swc1  $f10, 0($a0)    # store *x
  addi $t0, $t0, 1
  j Seidel2_start_for1
```
BB3

```
Seidel2_end_for1:
jr $ra
```
BB4

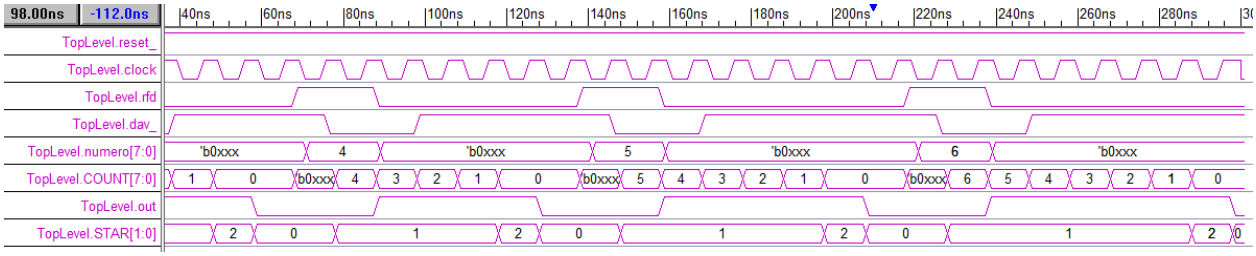
| Basic Block | Num. Iteraz. Ni | ALJ (1 ciclo) | B (3 cicli) | L/S (10 cicli) | FP (5 cicli) | Cbbi=cbbi*Ni |
|---|---|---|---|---|---|---|
| BB1 | 1 | 1 | | 1 | | 11 * 1 = 11 |
| BB2 | 4 | 1 | 1 | | | 4 * 4 = 16 |
| BB3 | 3 | 5 | | 4 | 2 | 55 * 3 = 165 |
| BB4 | 1 | 1 | | | | 1 |
| | | | | | | $C_{CPU}$=193 |

$\mathbf{T_{CPU}}= C_{CPU}/f_{CPU} = 193/(5*10^9)$ = **38.6 ns**

**COMPITINO di ARCHITETTURA DEI CALCOLATORI del 10-02-2016**
**SOLUZIONE**

ESERCIZIO 4)

```
module XXX(rfd, dav_,numero, out, clock,reset_);
 input        clock, reset_;
 output       out;
 output       rfd;
 input        dav_;
 input  [7:0] numero;
 reg       OUT;       assign    out=OUT;
 reg [7:0] COUNT;
 reg       RFD;       assign    rfd=RFD;
 reg [1:0] STAR;      parameter S0=0, S1=1, S2=2;
 always @(reset_==0) begin OUT<=0; RFD<=1; STAR<=S0; end
 always @(posedge clock) if (reset_==1)  #3
  casex (STAR)
   S0:     begin RFD<=1; OUT<=0; COUNT<=numero;  STAR<=(dav_==1) ?S0:S1; end
   S1:     begin RFD<=0; OUT<=1; COUNT<=COUNT-1; STAR<=(COUNT==1)?S2:S1; end
   S2:     begin RFD<=0; OUT<=0; STAR<=(dav_==0)?S2:S0; end
  endcase
endmodule
```




ESERCIZIO 5)

x1=0, x0=0101 → z=1
x1=1, x0=1011 → z=1
altrimenti → z=0

x1x0

Macchina a Stati Finiti:

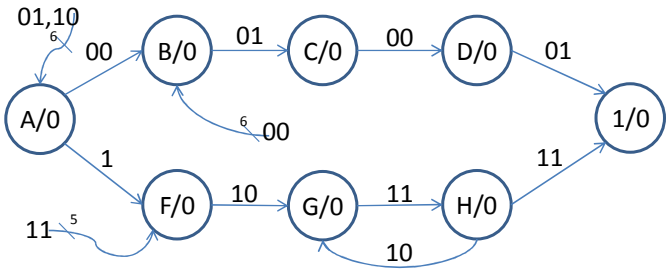



Tabella delle transizioni (e sua possibile codifica):

| $x_1x_0$ | 00 | 01 | 11 | 10 | z |
|---|---|---|---|---|---|
| A | B | A | F | A | 0 |
| B | B | C | F | A | 0 |
| C | D | A | F | A | 0 |
| D | B | E | F | A | 0 |
| E | B | A | F | A | 1 |
| F | B | A | F | G | 0 |
| G | B | A | H | A | 0 |
| H | B | A | E | G | 0 |

| $y_2y_1y_0$ \ $x_1x_0$ | 00 | 01 | 11 | 10 | z |
|---|---|---|---|---|---|
| 000 | 001 | 000 | 101 | 000 | 0 |
| 001 | 001 | 011 | 101 | 000 | 0 |
| 011 | 010 | 000 | 101 | 000 | 0 |
| 010 | 001 | 100 | 101 | 000 | 0 |
| 100 | 001 | 000 | 101 | 000 | 1 |
| 101 | 001 | 000 | 101 | 111 | 0 |
| 111 | 001 | 000 | 110 | 000 | 0 |
| 110 | 001 | 000 | 100 | 111 | 0 |

$a_2a_1a_0$

Equazioni booleane:
Per CN1:
a2=x1x0+/y2y1/y0x0+y2/y1y0x1+y2y1/y0x1

a1=/y2/y1y0/x1x0+ /y2y1y0/x1/x0+ y2y1y0x1x0
+ y2/y1y0x1/x0+ y2y1/y0x1/x0

a0=/y1/x1/x0+/y1x1x0+y2/y1y0x1+/y2/y1y0/x1
+/y2x1x0+y2/x1x0+y2y1/y0/x0+y1/y0/x1/x0

Per CN2:
z=y2/y1/y0

Circuito sequenziale sincronizzato basato su FF-D: