COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-11-2017 (rev.1.2) MATRICOLA_____

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

COGNOME_____

NOME_____

SVOLGIMENTO DELLA PROVA (selezionare una delle seguenti 4 opzioni):
☐ PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16 e 16/17": es. N.1+2+3+7.
☐ PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1+2+3+4+6.
☐ PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo CALCOLATORI es. N.1+2+3+4+5.
☐ PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.6+7.

NOTA: per l'esercizio 7 dovranno essere consegnati DUE files: il file del programma VERILOG e il file relativo all'output (screenshot o copy/paste)

1) [20] Trovare il codice assembly MIPS corrispondente al seguente programma **(usando solo e unicamente istruzioni della tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** riportate qua sotto per riferimento).

```
float A[9]={10.0,1.0,2.0,1.0,20.0,3.0,2.0,3.0,30.0};

void printmat(char *name, float *X, int m) {
   int i;
   print_string(name); print_string("[");
   print_int(m); print_string("]=\n");
   for (i=0; i<m; ++i)
     { print_float(X[i]); print_string(" "); }
   print_string("\n");
}

float mysqrt(float x) {
   int vint = *(int*)&x; // stessi bit visti come int
   vint = (1 << 29) + (vint >> 1) - (1 << 22);
   return *(float*)&vint; // Interpreto di nuovo come float
}
```

```
float *cholesky(float *A, int n) {
   float *L, s; int i, j, k;

   L = (float*)sbrk(n * n * 4);
   for (j=0; j<n; j++) {
      for (s=0, k=0; k<j; k++) s += L[j*n+k] * L[j*n+k];
      L[j*n+j] = mysqrt(A[j*n+j] - s);
      for (i=j+1; i<n; i++) {
         for (s=0, k=0; k<j; k++) s += L[i*n+k] * L[j*n+k];
         L[i*n+j] = (1.0 / L[j*n+j] * (A[i*n+j] - s));
      }
   }
   return L;
}

int main() {
   float *R;
   printmat("X", A, 9);
   R = cholesky(A, 3);
   printmat("R", R, 9);
}
```

2) [8] Si consideri una cache di dimensione 96B e a 3 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: 623, 339, 327, 379, 778, 139, 333, 754, 725, 354, 322, 354, 739, 1, 26, 754, 324, 354, 729, 354, 328, 354. 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) [4] Assemblare il seguente programma MIPS, utilizzando la tabella sottostante e riportando il formato utilizzato e i valori in esadecimale di ciascun campo di quel formato (es. LW R2, 0(R1) → FORMATO I: 23 1 2 0):

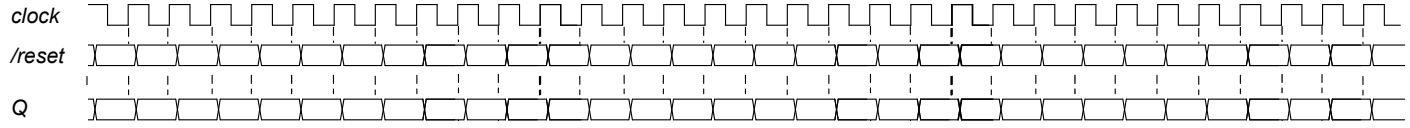
```
lab:    LW      R2, 0(R1)
        LW      R4, 0(R3)
        ADD     R5, R5, R2
        ADD     R5, R5, R4
        MUL     R5, R5
        SW      R5, 0(R1)
        SW      R5, 0(R3)
        ADDI    R1, R1, 4
        ADDI    R3, R3, 4
        ADDI    R7, R7, -1
        BNE     R7, R0, lab
```

4) [4] Spiegare la legge di Amdhal e tracciare il grafico di S (speedup) al variare di a (fattore di accelerazione della parte migliorabile) e con di p (porzione di parte migliorabile) pari a 0.1, 0.3, 0.7, 1.

5) [4] In un processore a 64 bit con memoria virtuale supportata da paginazione a 3 livelli con dimensione della pagina di 4KiB, vengono riservati 16 bit per indirizzare il primo livello, 18 bit per indirizzare il secondo livello e altri 18 bit per indirizzare il terzo livello. Rappresentare uno schema architetturale che implementi tale meccanismo di paginazione e istanziare numericamente il valore di un indirizzo virtuale, un indirizzo fisico corrispondente e valori realistici contenuti nelle celle delle pagine relative al percorso che collega l'indirizzo virtuale a quello fisico.

6) [7] Sintetizzare una rete sequenziale utilizzando il modello di Moore con un ingresso X su un bit e una uscita Z su un bit che riconosca le sequenze interallacciate 1,0,0,1. Rappresentare la macchina a stati finiti per tale rete logica, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.

7) [8] **Realizzare** in Verilog (per studenti 2014 e anni precedenti --> v.nota finale) sia un contatore ad anello a 4-bit che il relativo testbench: il clock ha un periodo di 10ns; il segnale _reset e' attivo basso: resta alto per 5ns, basso per 20ns, e poi ritorna alto per 600ns. Il contatore inizia il conteggio producendo sull'uscita Q il valore binario 1000 quando il segnale di /reset e'attivato, appena disattivato il /reset il conteggio prosegue. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità riportando i segnali clock, /reset, uscita Q per la durata complessiva (625ns). 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. (Per studenti 2014 e anni precedenti descrivere il comportamento di questa rete e disegnare líntero diagramma di temporizzazione, come sopra specificato).



COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-11-2017 (rev.1.2) MATRICOLA_____

SOLUZIONE **COGNOME**_____

(VERSIONE AGGIORNATA 04-11-20) **NOME**_____

Instructions

Opcode+Funct (hexadecimal)	Instruction	Example	Meaning	Comments
00+20/00+21	**add**	`add/addu $1,$2,$3`	$1 = $2 + $3	(signed/unsigned) 3 operands; exception possible
00+22/00+23	**subtract**	`sub/subu $1,$2,$3`	$1 = $2 - $3	(signed/unsigned) 3 operands; exception possible
08/09	**add immediate**	`addi/addiu $1,$2,100`	$1 = $2 + 100	(signed/unsigned) + constant ; exception possible
00+18/00+19	**multiplication**	`mult/multu $1, $2`	Hi,Lo= $1 x $2	(signed/unsigned) 64-bit Product ; result in Hi,Lo
00+1A/00+1B	**division**	`div/divu $1, $2`	Hi= $1 % $2, Lo = $1 / $2	(signed/unsigned) division
00+10/00+12	**move from Hi / move from Lo**	`mfhi/mflo $1`	$1 = Hi ($1 = Lo)	Create copy of Hi (Create a copy of Lo)
00+2A/00+2B	**set on less than**	`slt/sltu $1,$2,$3`	if ($2 < $3) $1 = 1; else $1 = 0	(signed/unsigned) compare $2 and $3 (less than)
0A/0B	**set on less than immediate**	`slti/sltiu $1,$2,100`	if ($2 < 100) $1 = 1; else $1 = 0	(signed/unsigned) compare $2 and constant (less than)
00+24/25/26/27	**and / or / xor / nor**	`and/or/xor/nor $1,$2,$3`	$1=$2&$3 / $2|$3 / $2^$3 / !($2|$3)	3 register operands; Logical AND/OR/XOR/NOR
0C/0D/0E	**and /or / xor immediate**	`andi/ori/xori $1,$2,100`	$1 = $2 & 100 / $2 | 100 / $2 ^100	Logical AND/OR/XOR register, constant
00+00	**shift left logical**	`sll $1,$2,10`	$1 = $2 << 10	Shift left by constant
00+02/00+03	**shift right (l=logical,a=arithmetic)**	`srl/sra $1,$2,10`	$1 = $2 >> 10	Shift right by constant (for arithmetic: sign is preserved)
23/20	**load word / load byte**	`lw/lb $1,100($2)`	$1 = Memory[$2+100]	Data from memory to register
24	**load byte unsigned**	`lbu $1,100($2)`	$1 = Memory[$2+100]	Data from mem. To reg.; no sign extension
2B/28	**store word / store byte**	`sw/sb $1,100($2)`	Memory[$2+100] = $1	Data from register to memory
0F	**load upper immediate**	`lui $1,0x1234`	$1=0x1234'0000	load most significant 16 bits
PSEUDOINSTRUCTION	**load address**	`la $1,var`	$1 = &var	Load address of var (lui $1,H16(&var);ori $1, L16(&var)) H16/L16=high/low 16 bits of &var
02	**jump**	`j 10000`	go to 10000	Jump to target address
00+08	**jump register**	`jr $31`	go to $31	For switch, procedure return
03	**jump and link**	`jal 10000`	$31 = PC + 4;go to 10000	For procedure call
04	**branch on equal**	`beq $1,$2,100`	if ($1 = = $2) go to PC+4+100	Equal test; PC relative branch
05	**branch on not equal**	`bne $1,$2,100`	if ($1 != $2) go to PC+4+100	Not equal test; PC relative
00+0C	**syscall**	`syscall`	call OS service $v0	See table of system calls below
10+10,rs=10	**rfe**	`rfe`	shift right (k,e) bits in STATUS reg	Exit Kernel Mode, Enable Interrupts
PSEUDOINSTRUCTION	**branch unconditional**	`b 100`	go to PC+4+100	PC relative branch (e.g., beq $0,$0,100)
PSEUDOINSTRUCTION	**no operation**	`nop`	do nothing	Do nothing (e.g. sll $0,$0,0)
30	**load-linked**	`ll $1,100($2)`	$1=Memory[$2+100]	Read and start to monitor the given memory location
38	**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)
11+00 fmt=10/11	**add.s / add.d**	`add.x $f0,f2,$f4`	$f0=$f2+$f4	Single and double precision add
11+01 fmt=10/11	**sub.s / sub.d**	`sub.x $f0,f2,$f4`	$f0=$f2-$f4	Single and double precision subtraction
11+02 fmt=10/11	**mul.s / mul.d**	`mul.x $f0,f2,$f4`	$f0=$f2*$f4	Single and double precision multiplication
11+03 fmt=10/11	**div.s / div.d**	`div.x $f0,f2,$f4`	$f0=$f2/$f4	Single and double precision division
11+05 fmt=10/11	**abs.s / abs.d**	`abs.x $f0,$f2`	$f0=ABS($f2)	Single and double precision absolute value
11+06 fmt=10/11	**mov.s / mov.d**	`mov.x $f0,$f2`	$f0←$f2	Single and double precision move
11+07 fmt=10/11	**neg.s / neg.d**	`neg.x $f0,$f2`	$f0= – ($f2)	Single and double precision opposite value
11+3C(31,32,3D,3E,3F) fmt=10/11	**c.lt.s / c.lt.d (ne,eq,gt,le,ge)**	`c.lt.x $f0,$f2`	Temp=($f0<$f2)	Single and double: compare $f0 and $f2 <,=,!=,>,<=>=
11+00 fmt=4/0	**move to/from coprocessor 1**	`mtc1/mfc1 $1,$f2`	$f2=$1 / $1=$f2	Move $1 to/from C1reg. $f2 (no conversion)
10+00 fmt=4/0	**move to/from coprocessor 0**	`mtc0/mfc0 $1,$2`	$c2=$1 / $1=$c2	Move $1 to/from C0 reg. $f2 (no conversion)
11+00 fmt=6/2	**move to/from control reg of cop.1**	`ctc1/cfc1 $1,$cf2`	$cf2=$1 / $1=$cf2	Move $1 to/from C1-CONTROL register
11 fmt=8,ft=1/0	**branch on true/false**	`bc1t/bc1f label`	If (Temp = = true/false) go to label	Temp is 'Condition-Code'
31/39	**load/store floating point (32bit)**	`lwc1/swc1 $f0,0($1)`	$f0←Memory[$1] / Memory[$1]←$f0	Data from FP (C1) register to memory
11+21,fmt=10/11+22,fmt=11	**convert from/to single to/from double**	`cvt.d.s/cvt s.d $f0,$f2`	$f0=(double)$f2/$f0=(single)$f2	Type conversion
11+24,fmt=11/11+20	**convert from/to single to/from integer**	`cvt.w.s/cvt s.w $f1,$f0`	$f1=(int)$f0 / $f0=(single)$f2	Type conversion

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	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-11-2017 (rev.1.2) **MATRICOLA**_____

SOLUZIONE **COGNOME**_____

(VERSIONE AGGIORNATA 04-11-20) **NOME**_____

ESERCIZIO 1)

```
.data
A:      .float 10.0,  1.0,   2.0
        .float  1.0, 20.0,   3.0
        .float  2.0,  3.0,  30.0
uno:    .float 1.0
paropen: .asciiz "["
parclose:.asciiz "]=\n"
newline: .asciiz "\n"
spc:    .asciiz " "
namer:  .asciiz "R"
namex:  .asciiz "X"

.text
.globl main

main:
       #________NO CALL FRAME___________
  # R e' una var. temporanea che coincide
  # col valore di ritorno della f. cholesky
        la   $a0, namex
        la   $a1, A
        addi $a2, $0, 9
        jal  printmat

        la   $a0, A
        addi $a1, $0, 3
        jal  cholesky

        la   $a0, namer
        add  $a1, $0, $v0
        addi $a2, $0, 9
        jal  printmat

        addi $v0, $0, 10
        syscall

printmat:
        #________NO CALL FRAME___________
        # i associato a t0
        addi $v0, $0, 4 # stampa 'name'=a0
        syscall
        la   $a0, paropen
        addi $v0, $0, 4
        syscall  # stampa parentesi aperta f2:
        add  $a0, $0, $a2
        addi $v0, $0, 1
        syscall          # stampa m
        la   $a0, parclose
        addi $v0, $0, 4
        syscall   # stampa parentesi chiusa
        add  $t0, $0, $0 # i=0
pmfor:
        slt  $t1, $t0, $a2       # i<?m
        beq  $t1, $0, pmfinefor
        sll  $t1, $t0, 2 # i*4
        add  $a0, $a1, $t1       # &X+i*4
        lwc1 $f12,0($a0)         # X[i]
        addi $v0, $0, 2
        syscall            # stampa X[i]
        la   $a0, spc
        addi $v0, $0, 4
        syscall        # stampa uno spazio
        addi $t0, $t0, 1 # ++i
        j pmfor
pmfinefor:
        la   $a0, newline
        addi $v0, $0, 4
        syscall   # stampa parentesi aperta
        jr $ra
```

```
mysqrt:
        #________NO CALL FRAME___________
        mfc1    $t0,  $f12
        sra     $t0,  $t0, 1 # vint>>1
        addi    $t1,  $0,  1
        sll     $t1,  $t1, 29
        add     $t0,  $t0, $t1
        addi    $t1,  $0,  1
        sll     $t1,  $t1, 22
        sub     $t0,  $t0, $t1
        mtc1    $t0,  $f12
        jr      $ra

cholesky:
        #________CALL FRAME___________
        # saved variables: s0   4B
        # saved variables: s1=j 4B
        # saved variables: s2=k 4B
        # saved variables: s3=i 4B
        # saved variables: s4   4B
        #                  ra   4B
        #____________________Totale 8B
        addi    $sp, $sp, -24
        sw      $ra, 0($sp)
        sw      $s0, 4($sp)
        sw      $s1, 8($sp)
        sw      $s2, 12($sp)
        sw      $s3, 16($sp)
        sw      $s4, 20($sp)
        add     s0, $0, $a0 #salvo a0 in s0

        mult    $a1, $a1    # n*n
        mflo    $a0
        sll     $a0, $a0, 2 # n*n*4
        addi    $v0, $0, 9  # sbrk, v0=&L
        syscall

        add     $s1, $0, $0 # j=0
f1:
        slt     $t9, $s1, $a1    # j<?n
        beq     $t9, $0, finef1
        mtc1    $0, $f0      # s=0.0
        add     $s2, $0, $0  # k=0
f2:
        slt     $t9, $s2, $s1    # k<?j
        beq     $t9, $0, finef2

        mult    $s1, $a1
        mflo    $t8
        add     $t8, $t8, $s2    # j*n+k
        sll     $t8, $t8, 2     # (j*n+k)*4
        add     $t8, $v0, $t8    # &L[j,k]
        lwc1    $f2, 0($t8)      # L[j,k]
        mul.s   $f1, $f2, $f2    # L*L
        add.s   $f0, $f0, $f1    # s+=L*L
        addi    $s2, $s2, 1      # ++k
        j       f2
finef2:
        mult    $s1, $a1
        mflo    $s4
        add     $s4, $s4, $s1    # j*n+j
        sll     $s4, $s4, 2 # (j*n+j)*4
        add     $t8, $s0, $s4    # &A[j,j]
        lwc1    $f1, 0($t8)      # A[j,j]
        sub.s   $f12, $f1, $f0   # A[]-s
        jal     mysqrt
        add     $t8, $v0, $s4    # &L[j,j]
        swc1   $f12,0($t8)#scrivo in L[j,j]
        addi    $s3, $s2, 1      # i=j+i
f3:
```

```
        slt     $t9, $s3, $a1    # i<?n
        beq     $t9, $0, finef3
        mtc1    $0, $f0          # s=0.0
        add     $s2, $0, $0      # k=0
f4:
        slt     $t9, $s2, $s1    # k<?j
        beq     $t9, $0, finef4

        mult    $s3, $a1
        mflo    $t8
        add     $t8, $t8, $s2    # i*n+k
        sll     $t8, $t8, 2  # (i*n+k)*4
        add     $t8, $v0, $t8    # &L[i,k]
        lwc1    $f3, 0($t8)      # L[i,k]

        mult    $s1, $a1
        mflo    $t8
        add     $t8, $t8, $s2    # j*n+k
        sll     $t8, $t8, 2    # (j*n+k)*4
        add     $t8, $v0, $t8    # &L[j,k]
        lwc1    $f4, 0($t8)      # L[j,k]

        mul.s   $f1,$f3,$f4 #L[i,k]*L[j,k]
        add.s   $f0, $f0, $f1 #s+=L[]*L[]

        addi    $s2, $s2, 1   # ++k
        j       f4
finef4:
        la      $t8, uno         # 1.0
        lwc1    $f10,0($t8)
        div.s   $f5,$f10,$f12 #1.0/L[j,j]

        mult    $s3, $a1
        mflo    $s4
        add     $s4, $s4, $s1    # i*n+j
        sll     $s4, $s4, 2   # (i*n+j)*4
        add     $t7, $s0, $s4    # &A[i,j]
        lwc1    $f1, 0($t7)      # A[i,j]
        sub.s   $f6, $f1, $f0    # A[]-s

        mult    $s1, $a1
        mflo    $t8
        add     $t8, $t8, $s2 # j*n+k
        sll     $t8, $t8, 2 # (j*n+k)*4
        add     $t8, $v0, $t8  # &L[j,k]
        lwc1    $f3, 0($t8)    # L[j,k]
        mul.s  $f7,$f5,$f6 # 1/L[]*(A[]-s)

        add     $t8, $v0, $s4 # &L[i,j]
        swc1    $f7, 0($t8) # scrivo in L[]

        addi    $s3, $s3, 1 # ++i
        j       f3

finef3:
        addi    $s1, $s1, 1      # ++j
        j       f1
finef1:
# v0 already contains &L
        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, 24
        jr $ra
```

ESERCIZIO 3)

```
lab:LW      R2, 0(R1)        FORMATO I (op,rs,rd,im):          23 1 2 0000
    LW      R4, 0(R3)        FORMATO I (op,rs,rd,im):          23 3 4 0000
    ADD     R5, R5, R2       FORMATO R (op,rs,rt,rd,sh,fu):    00 5 2 5 0 20
    ADD     R5, R5, R4       FORMATO R (op,rs,rt,rd,sh,fu):    00 5 4 5 0 20
    MUL     R5, R5           FORMATO R (op,rs,rt,rd,sh,fu):    00 5 5 0 0 18
    SW      R5, 0(R1)        FORMATO I (op,rs,rd,im):          2B 1 5 0000
    SW      R5, 0(R3)        FORMATO I (op,rs,rd,im):          2B 3 5 0000
    ADDI    R1, R1, 4        FORMATO I (op,rs,rd,im):          08 1 1 0004
    ADDI    R3, R3, 4        FORMATO I (op,rs,rd,im):          08 3 3 0004
    ADDI    R7, R7, -1       FORMATO I (op,rs,rd,im):          08 7 7 FFFF
    BNE     R7, R0, lab      FORMATO I (op,rs,rd,im):          05 7 0 FFF5
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-11-2017 (rev.1.2) MATRICOLA_____

SOLUZIONE **COGNOME**_____

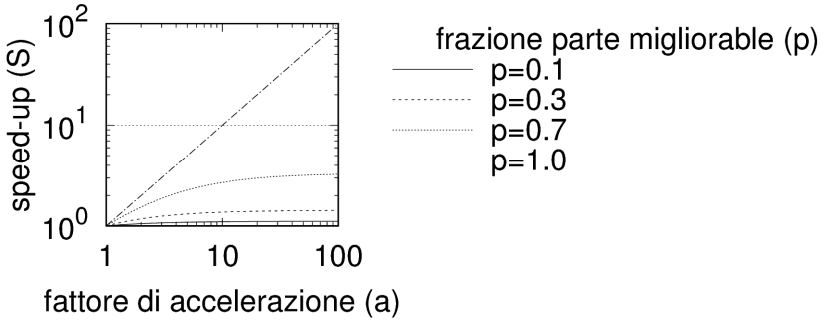
(VERSIONE AGGIORNATA 04-11-20) **NOME**_____

ESERCIZIO 2)

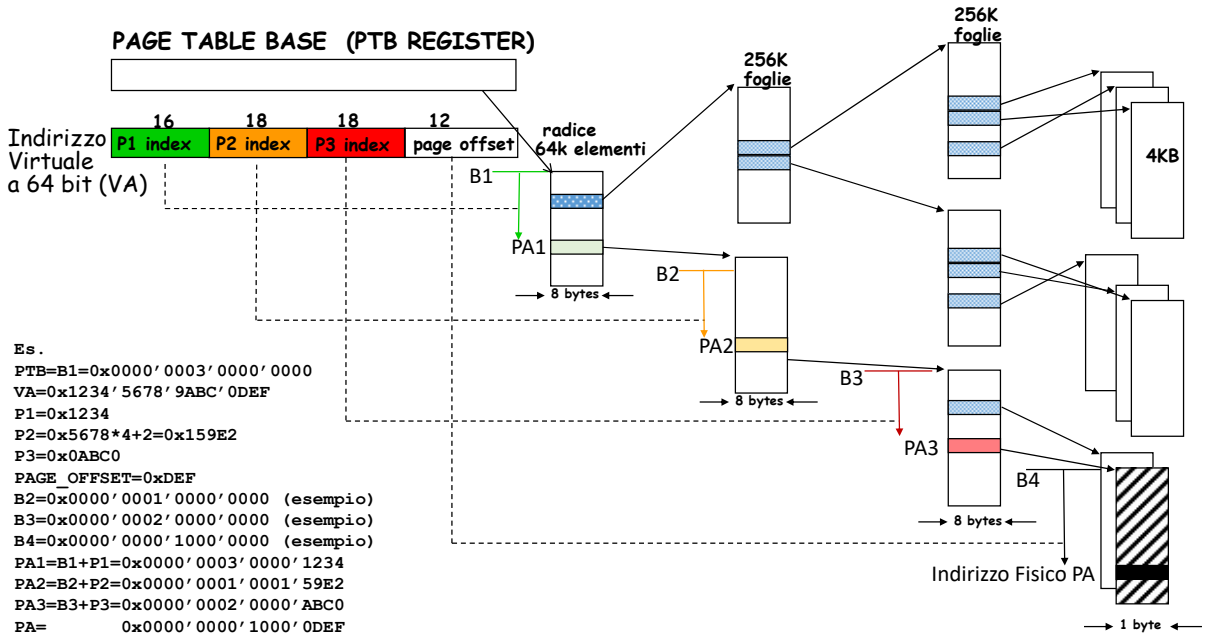
```
A = 3
B = 8
C = 96
RP = FIFO
Thit = 4
Tpen = 40
File: c1171114.sh_001000.din
Read 22 references.
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  623   77  19   1   7  0 [1]:2,0,0 [1]:0,0,0  [1]:19,-,-
=== W  339   42  10   2   3  0 [2]:2,0,0 [2]:0,0,0  [2]:10,-,-
=== R  327   40  10   0   7  0 [0]:2,0,0 [0]:0,0,0  [0]:10,-,-
=== W  379   47  11   3   3  0 [3]:2,0,0 [3]:0,0,0  [3]:11,-,-
=== R  778   97  24   1   2  0 [1]:1,2,0 [1]:0,0,0  [1]:19,24,-
=== W  139   17   4   1   3  0 [1]:0,1,2 [1]:0,0,0  [1]:19,24,4
=== R  333   41  10   1   5  0 [1]:2,0,1 [1]:0,0,0  [1]:10,24,4 (out: XM=77 XT=19 XS=1 )
=== W  754   94  23   2   2  0 [2]:1,2,0 [2]:0,0,0  [2]:10,23,-
=== R  725   90  22   2   5  0 [2]:0,1,2 [2]:0,0,0  [2]:10,23,22
=== W  354   44  11   0   2  0 [0]:1,2,0 [0]:0,0,0  [0]:10,11,-
=== R  322   40  10   0   2  1 [0]:1,2,0 [0]:0,0,0  [0]:10,11,-
=== W  354   44  11   0   2  1 [0]:1,2,0 [0]:0,1,0  [0]:10,11,-
=== R  739   92  23   0   3  0 [0]:0,1,2 [0]:0,1,0  [0]:10,11,23
=== W    1    0   0   0   1  0 [0]:2,0,1 [0]:0,1,0  [0]:0,11,23 (out: XM=40 XT=10 XS=0 )
=== R   26    3   0   3   2  0 [3]:1,2,0 [3]:0,0,0  [3]:11,0,-
=== W  754   94  23   2   2  1 [2]:0,1,2 [2]:0,1,0  [2]:10,23,22
=== R  324   40  10   0   4  0 [0]:1,2,0 [0]:0,0,0  [0]:0,10,23 (out: XM=44 XT=11 XS=0 )
=== W  354   44  11   0   2  0 [0]:0,1,2 [0]:0,0,0  [0]:0,10,11 (out: XM=92 XT=23 XS=0 )
=== R  729   91  22   3   1  0 [3]:0,1,2 [3]:0,0,0  [3]:11,0,22
=== W  354   44  11   0   2  1 [0]:0,1,2 [0]:0,0,1  [0]:0,10,11
=== R  328   41  10   1   0  1 [1]:2,0,1 [1]:0,0,0  [1]:10,24,4
=== W  354   44  11   0   2  1 [0]:0,1,2 [0]:0,0,1  [0]:0,10,11
-------------------------------------
P1 Nmiss=16   Nhit=6  Nref=22  mrate=0.727273   AMAT=33.0909
```

ESERCIZIO 4)

La legge di Amdhal mette in relazione lo speedup di una parte del sistema con lo speedup dell'intero sistema.
S(p,a)=1/((1-p)+p/a)



ESERCIZIO 5)



ESERCIZIO 6)

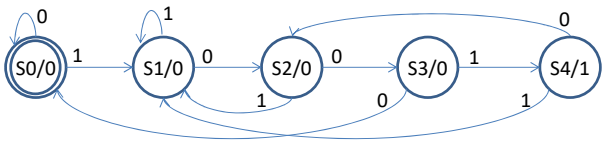
COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-11-2017 (rev.1.2) MATRICOLA_____

SOLUZIONE **COGNOME**_____

(VERSIONE AGGIORNATA 04-11-20) **NOME**_____

In corrispondenza del pattern X_{t-2},X_{t-1},X_t = 1,1,0 oppure 1,0,1 ottengo → Z_{t+1} = 1; (ricordare che è richiesto Moore).

Diagramma degli Stati



Schema di riferimento per Moore

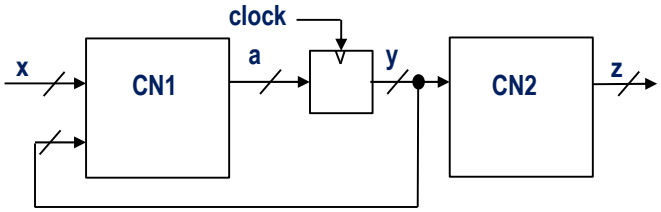


Tabella delle Transizioni (Tabella degli Stati)

TABELLA DELLE TRANSIZIONI

STATO ATTUALE \ x	0	1	z
S0	S0	S1	0
S1	S2	S1	0
S2	S3	S1	0
S3	S0	S4	0
S4	S2	S1	1
S5	X	X	X
S6	X	X	X
S7	X	X	X

STATO SUCCESSIVO/USCITA

Scelta della codifica degli Stati

STATO \ CODIFICA $y_2y_1y_0$	
S0	000
S1	001
S2	011
S3	010
S4	100
S5	xxx
S6	xxx
S7	xxx

(completare il n. di stati a una potenza di 2)

Rappresentazione della Tabella degli Stati in Formato più comodo per la sintesi

OVVERO

y_1y_0 \ y_2x	00	01	11	10
00	S0	S1	S1	S2
01	S2	S1	X/X	X/X
11	S3	S1	X/X	X/X
10	S0	S4	X/X	X/X

STATO SUCCESSIVO/USCITA

OVVERO

y_1y_0 \ y_2x	00	01	11	10
00	000	001	001	011
01	011	001	XXX	XXX
11	010	001	XXX	XXX
10	000	100	XXX	XXX

a2,a1,a0

Sintesi della variable 'a' (stato successivo o ingresso dello STATUS REGISTER -- STAR):

y_1y_0 \ y_2x	00	01	11	10
00	0	0	0	0
01	0	0	X	X
11	0	0	X	X
10	0	1	X	X

a_2

a2=y1/y0x

y_1y_0 \ y_2x	00	01	11	10
00	0	0	0	1
01	1	0	X	X
11	1	0	X	X
10	0	0	X	X

a_1

a1=y0/x

y_1y_0 \ y_2x	00	01	11	10
00	0	1	1	1
01	1	1	X	X
11	0	1	X	X
10	0	0	X	X

a_0

a0=/y1y0+y0x+y2+/y1x

Sintesi della variable 'z' (uscita):

y_2 \ y_1y_0	00	01	11	10
0	0	0	0	0
1	1	X	X	X

z

z=y2

ESERCIZIO 7)
(diagramma degli stati come nell'esercizio 6)

```
`timescale 1ns/1ps
module top;
  reg clock,_reset;
  wire[3:0] Q;
  contatore_ad_anello rc(Q,clock,_reset);
  always #10 clock = ~clock;
  initial begin
    $display("time,\t clock,\t _reset,\t QQQQ");
    $monitor("%g,\t %b,\t %b,\t
%b",$time,clock,_reset,Q);
     _reset=1'b1; clock=0;
    #5  _reset=1'b0;
    #20 _reset=1'b1;
    #600 $finish;
  end
endmodule
```

```
module contatore_ad_anello(q,clock,_reset);
  input clock,_reset;
  output[3:0] q;
  reg[3:0] q;
  always @(posedge clock)
    if(!_reset)
      q<=4'b1000;
    else begin
      q[3]<=q[0];
      q[2]<=q[3];
      q[1]<=q[2];
      q[0]<=q[1];
    end
endmodule
```