COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-02-2019

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, ETC

SVOLGIMENTO DELLA PROVA:

☐ PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16, 16/17, 17/18, 18/19": es. N.1+2+3+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) [13/30] 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).

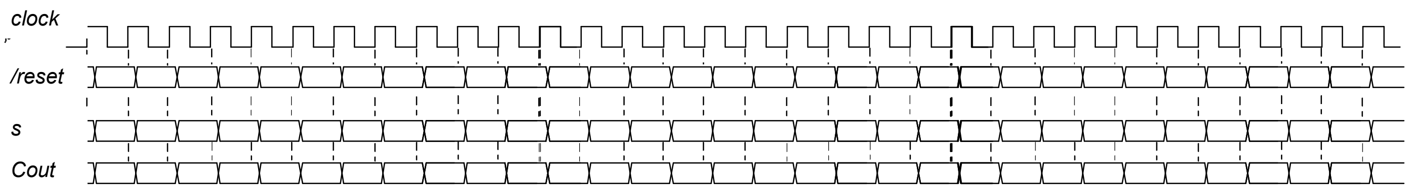
```
int a[12] = { 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1 };
int c[32];

int bitstuff(int *a, int n) {
  int i=0, count=1, j=0, k;
  while(i<n) {
    if(a[i]==1) {
      c[j]=a[i];
      for(k=i+1; a[k]==1 && k<n && count<5; k++) {
        j++; c[j]=a[k]; count++;
        if(count==5) { j++; c[j]=0; }
        i=k;
      }
    } else c[j]=a[i];
    i++; j++;
  }
  return j;
}
```

```
void main()
{
  int m = bitstuff(a, 12);
  for(int i=0; i<m; i++) print_int(c[i]);
}
```

2) [5/30] Si consideri una cache di dimensione 192B e a 3 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' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 1, 105, 240, 378, 492, 597, 678, 712, 850, 976, 1025, 1123, 1233, 1377, 1456, 1512, 1613, 1714, 1844, 1911, 2012. 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) [3/30] Spiegare il funzionamento dei cinque modi di indirizzamento del processore MIPS.

7) [9/30] **Realizzare** in Verilog un sommatore a 5-bit di tipo carry-look-ahead. Il testbench e' dato. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità riportando i segnali clock, /reset, uscita S e Cout (carry in uscita al sommatore) per la durata complessiva (45ns). 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



Testbench:

```
`timescale 1ns/1ps
module cla5_testbench;
   reg reset_; initial begin reset_=0; #2 reset_=1; #600; $stop; end
   reg clock;  initial clock<=0;   always #5 clock<=(!clock);
   reg[4:0] a, b; reg cin; wire cout; wire [4:0] s;
   initial begin cin <=0;
      @(posedge clock); a<=14; b<=7; @(posedge clock); a<=5; b<=19;
      @(posedge clock); a<=16; b<=2; @(posedge clock); a<=9; b<=13;
      #10 $finish;
   end
   cla_adder_5bit CLA5(a,b,cin,s,cout);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-02-2019

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	**ift right (l=logical,a=arithmetic)**	`srl/sra     $1,$2,10`	$1 = $2 >> 10	Shift right by constant (for arithmetic: sign is preserved)
00+04	**shift left logical**	`sllv        $1,$2,10`	$1 = $2 << $3	Shift left by variable
00+06/00+07	**ift right (l=logical,a=arithmetic)**	`srlv/srav   $1,$2,10`	$1 = $2 >> $3	Shift right by variable (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

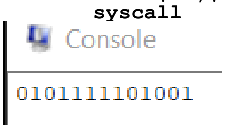
Service Name	Service Num. ($v0)	INPUT Arguments	OUTPUT Arguments
print_int	1	$a0=integer to print	---
print_float	2	$f12=float to print	---
print_double	3	($f12,$f13)=double to print	---
print_string	4	$a0=address of ASCIIZ string to print	---
read_int	5	---	$v0=integer
read_float	6	---	$f0=float
read_double	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 22-02-2019

SOLUZIONE

ESERCIZIO 1

```
.data
a: .word 0, 1, 0, 1, 1, 1, 1, 1, 1,
0, 0, 1
c: .space 128

.globl main
.text

#----------------------------------
--
# a0=a,a1=n
t0=i,t1=count,t2=j,t3=k
bitstuff:
        addi $t0,$0,0   #i=0
        addi $t1,$0,1   #count=1
        addi $t2,$0,0   #j=0
bs_wh_ini:
    slt  $t9,$t0,$a1# i<?n
        beq  $t9,$0,bs_wh_end

        sll  $t4,$t0,2  #i*4
        add  $t4,$t4,$a0#&a[i]
        lw   $t4,0($t4) #a[i]
        addi $t5,$0,1   #1
        bne  $t4,$t5,bs_if1_else
        sll  $t9,$t2,2  #j*4
        la   $t8,c      #&c
        add  $t8,$t8,$t9#&c[j]
        sw   $t4,0($t8) #c[j]=a[i]

        addi $t3,$t0,1  #k=i+1
bs_for_ini:
        sll  $t6,$t3,2  #k*4
        add  $t6,$t6,$a0#&a[k]
        lw   $t6,0($t6) #a[k]
        bne  $t6,$t5,bs_for_end
        slt  $t9,$t3,$a1#k<?n
        beq  $t9,$0,bs_for_end
        slti $t9,$t1,5  #count<?5
        beq  $t9,$0,bs_for_end

        addi $t2,$t2,1  #j++
        sll  $t9,$t2,2  #j*4
        la   $t8,c      #&c
        add  $t8,$t8,$t9#&c[j]
        sw   $t6,0($t8) #c[j]=a[k]
        addi $t1,$t1,1  #count++
        addi $t7,$0,5   #5
        bne  $t1,$t7,bs_if2_end
#count==?5

        addi $t2,$t2,1  #j++
        sll  $t9,$t2,2  #j*4
        la   $t8,c      #&c
        add  $t8,$t8,$t9#&c[j]
        sw   $0,0($t8)  #c[j]=0

bs_if2_end:
        add  $t0,$t3,$0 #i=k
        addi $t3,$t3,1  #k++
        j bs_for_ini
bs_for_end:
        j bs_if1_end

bs_if1_else:
        sll  $t9,$t2,2  #j*4
        la   $t8,c      #&c
        add  $t8,$t8,$t9#&c[j]
        sw   $t4,0($t8) #c[j]=a[i]
bs_if1_end:
        addi $t0,$t0,1  #i++
        addi $t2,$t2,1  #j++
        j bs_wh_ini
bs_wh_end:

        add  $v0,$t2,$0
        jr $ra


#-----------------------------------
--
main:
        la   $a0,a      # &a
        addi $a1,$0,12  # n
        jal  bitstuff
        add  $t1,$v0,$0 # m
        add  $t0,$0,$0  # i=0
main_forini:
    slt  $t9,$t0,$t1# i<?m
        beq  $t9,$0,main_forend
        sll  $t2,$t0,2  # i*4
        la   $t3,c      # &c
        add  $t2,$t2,$t3# &c[i]
        lw   $a0,0($t2) # c[i]
        addi $v0,$0,1   # serv.1
        syscall
        addi $t0,$t0,1  #i++
        j    main_forini
main_forend:
        addi $v0,$0,10  # serv.10
        syscall
```



```
010111101001
```

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=192/16/3=4,   XM=X/B,   XS=XM%S,   XT=XM/S:

```
A=3, B=16, C=192, RP=FIFO, Thit=4, Tpen=40, 21 references:
=== T     X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R     1    0   0   0   1  0 [0]:2,0,0  [0]:0,0,0   [0]:0,-,-
=== W   105    6   1   2   9  0 [2]:2,0,0  [2]:0,0,0   [2]:1,-,-
=== R   240   15   3   3   0  0 [3]:2,0,0  [3]:0,0,0   [3]:3,-,-
=== W   378   23   5   3  10  0 [3]:1,2,0  [3]:0,0,0   [3]:3,5,-
=== R   492   30   7   2  12  0 [2]:1,2,0  [2]:0,0,0   [2]:1,7,-
=== W   597   37   9   1   5  0 [1]:2,0,0  [1]:0,0,0   [1]:9,-,-
=== R   678   42  10   2   6  0 [2]:0,1,2  [2]:0,0,0   [2]:1,7,10
=== W   712   44  11   0   8  0 [0]:1,2,0  [0]:0,0,0   [0]:0,11,-
=== R   850   53  13   1   2  0 [1]:1,2,0  [1]:0,0,0   [1]:9,13,-
=== W   976   61  15   1   0  0 [1]:0,1,2  [1]:0,0,0   [1]:9,13,15
=== R  1025   64  16   0   1  0 [0]:0,1,2  [0]:0,0,0   [0]:0,11,16
=== W  1123   70  17   2   3  0 [2]:2,0,1  [2]:0,0,0   [2]:17,7,10
=== R  1233   77  19   1   1  0 [1]:2,0,1  [1]:0,0,0   [1]:19,13,15
=== W  1377   86  21   2   1  0 [2]:1,2,0  [2]:0,0,0   [2]:17,21,10
=== R  1456   91  22   3   0  0 [3]:0,1,2  [3]:0,0,0   [3]:3,5,22
=== W  1512   94  23   2   8  0 [2]:0,1,2  [2]:0,0,0   [2]:17,21,23
=== R  1613  100  25   0  13  0 [0]:2,0,1  [0]:0,0,0   [0]:25,11,16
=== W  1714  107  26   3   2  0 [3]:2,0,1  [3]:0,0,0   [3]:26,5,22
=== R  1844  115  28   3   4  0 [3]:1,2,0  [3]:0,0,0   [3]:26,28,22
=== W  1911  119  29   3   7  0 [3]:0,1,2  [3]:0,0,0   [3]:26,28,29
=== R  2012  125  31   1  12  0 [1]:1,2,0  [1]:0,0,0   [1]:19,31,15
-------------------------------------
P1 Nmiss=21   Nhit=0  Nref=21  mrate=1.000000   AMAT=th+mrate*tpen=44
```

CONTENTUTI dei 3 SET al termine:

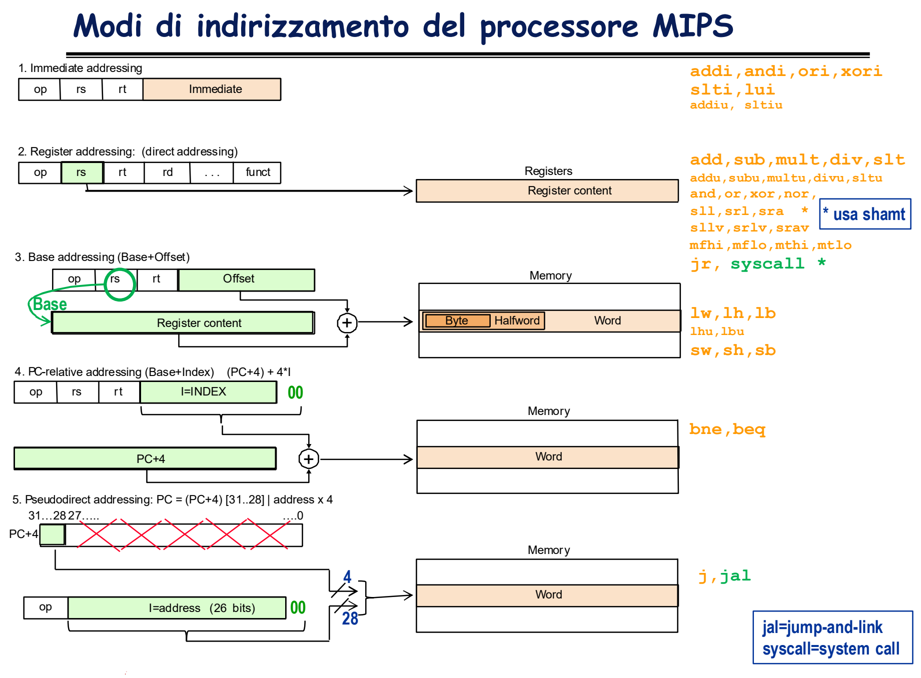
LISTA BLOCCHI USCENTI:

```
(out:  XM=6   XT=1   XS=2 )
(out:  XM=37  XT=9   XS=1 )
(out:  XM=30  XT=7   XS=2 )

(out:  XM=42  XT=10  XS=2 )
(out:  XM=0   XT=0   XS=0 )
(out:  XM=15  XT=3   XS=3 )
(out:  XM=23  XT=5   XS=3 )
(out:  XM=91  XT=22  XS=3 )
(out:  XM=53  XT=13  XS=1 )
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-02-2019

SOLUZIONE

ESERCIZIO 3

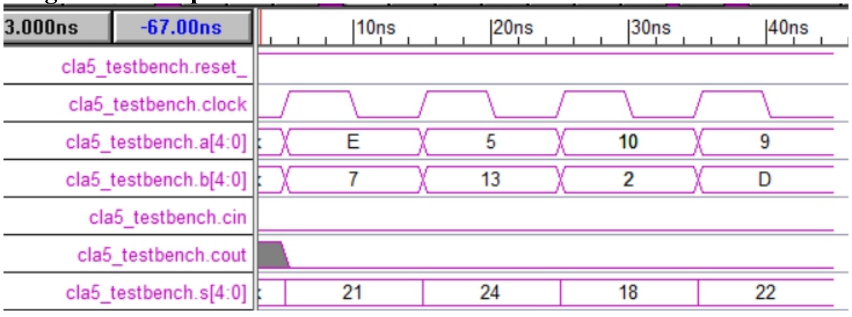


ESERCIZIO 7

Codice VERILOG:

```
module cla_full_adder(a,b,c,  g,p,s);
  input  a, b, c;
  output g, p, s;
  assign g = a & b;
  assign p = a ^ b;
  assign s = a ^ (b ^ c);
endmodule

module cla_adder_5bit(a,b,cin,s,cout);
  input [4:0]a,b; input  cin;
  output [4:0]s;  output cout;
  wire [5:0] c;
  wire [4:0] g, p;
  assign c[0] = cin;
  assign cout = c[5];
  cla_full_adder add0(a[0],b[0],c[0],  g[0],p[0],s[0]);
  assign c[1] = g[0] | (p[0] & c[0]);
  cla_full_adder add1(a[1],b[1],c[1],  g[1],p[1],s[1]);
  // assign c[2] = g[1] | (p[1] & c[1]);
  assign c[2] = g[1] | (p[1] & g[0]) | (p[1] & p[0] & c[0]);
  cla_full_adder add2(a[2],b[2],c[2],g[2],p[2],s[2]);
  // assign c[3] = g[2] | (p[2] & c[2]);
  assign c[3] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0])
  | (p[2] & p[1] & p[0] & c[0]);
  cla_full_adder add3(a[3],b[3],c[3],g[3],p[3],s[3]);
  // assign c[4] = g[3] | (p[3] & c[3]);
  assign c[4] = g[3]|(p[3]&g[2])|(p[3]& p[2]&g[1])|(p[3]&p[2]&p[1]&g[0])
  |(p[3]&p[2]&p[1]&p[0]&c[0]);
  cla_full_adder add4(a[4],b[4],c[4],g[4],p[4],s[4]);
  // assign c[5] = g[4] | (p[4] & c[4]);
  assign c[5] = g[4]|(p[4]&g[3])|(p[4]&p[3]&g[2])|(p[4]&p[3]&p[2]&g[1])
  |(p[4]&p[3]&p[2]&p[1]&g[0])|(p[4]&p[3]&p[2]&p[1]&p[0]&c[0]);
endmodule
```

Diagramma temporale:


3.000ns
-67.00ns
10ns
20ns
30ns
40ns
cla5_testbench.reset_
cla5_testbench.clock
cla5_testbench.a[4:0]
E
cla5_testbench.b[4:0]
D
cla5_testbench.cin
cla5_testbench.cout
cla5_testbench.s[4:0]