COMPITO di ARCHITETTURA DEI CALCOLATORI del 20-11-2018

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": 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) [19/38] 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). Nota: la funzione "fabs" puo' essere mappata direttamente sull'istruzione "abs.s".

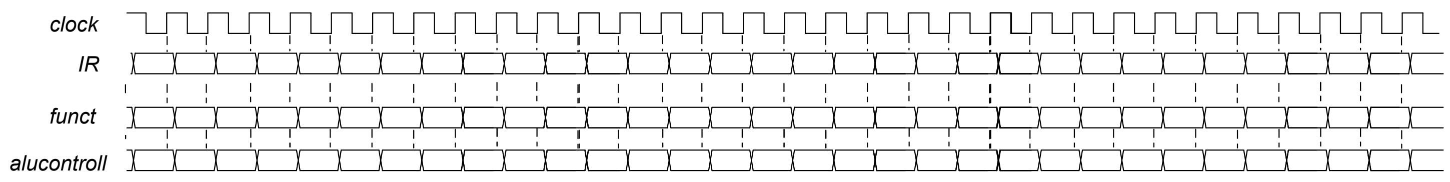
```
typedef struct header {
   struct header *ptr; unsigned size;
} Header;
static Header base = {NULL,0 };
static Header *freep = NULL;

void myfree(void *ap) {
   Header *bp, *p;
   bp = (Header *)ap - 1;
   for (p = freep; !(bp > p && bp < p->ptr); p = p->ptr) {
       if (p >= p->ptr && (bp > p || bp < p->ptr)) break;
   }
   if (bp + bp->size == p->ptr) {
      bp->size += p->ptr->size;
      bp->ptr = p->ptr->ptr;
   } else bp->ptr = p->ptr;
   if (p + p->size == bp) {
      p->size += bp->size;
      p->ptr = bp->ptr;
   } else p->ptr = bp;
   freep = p;
}
```

```
void *alloc_and_print_pun(int sz) {
   void *p = sbrk(sz);
   print_string("p=");
   print_int(p);
   print_string("\n");
   return (p);
}

int main() {
   void *p0, *p1, *p2, *p3;
   base.ptr = &base; freep=&base;
   p0 = alloc_and_print_pun(0);
   p1 = alloc_and_print_pun(256);
   p2 = alloc_and_print_pun(256);
   p3 = alloc_and_print_pun(256);
   myfree(p1); myfree(p2); myfree(p3);
   p0 = alloc_and_print_pun(0);
}
```

2) [7/38] 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' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 755, 773, 715, 719, 722, 747, 718, 649, 734, 748, 777, 719, 683, 643, 791, 744, 770, 745, 61, 794. 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) [4/38] Spiegare la differenze e i vantaggi/svantaggi delle quattro categorie di benchmark: "Workload", "Benchmark-suite", "Small-kernel", "Micro-benchmark".

7) [8/38] **Realizzare** in Verilog il modulo "aludec" che implementa la rete combinatoria relativa al decoder dei codici operativi della ALU di un semplice processore MIPS, che supporti le operazioni add/addi/sub/and/or/slt/lw/sw/beq. E' gia' fornito il modulo testbench e il campo funct puo' essere derivato dalla tabella delle istruzioni sottostante. Il campo aluop vale 0 per le istruzioni di formato I, vale 1 per beq, mentre vale 2 per le altre istruzioni. Il campo alucontrol vale rispettivamente 2 per le istruzioni di formato I, vale 6 per beq, mentre vale 2/6/0/1/7 rispettivamente per add/sub/and/or/slt. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità riportando i segnali clock, IR, funct, uscita alucontrol. 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 aludec_testbench;
   reg reset_; initial begin reset_=0; #22 reset_=1; #300; $stop; end
   reg clock;  initial clock=0; always #5 clock<=(!clock);
   wire[5:0] funct; reg[1:0] aluop;
   wire[2:0] alucontrol; reg[31:0] IR;
   initial begin
      wait(reset_==1); aluop<=0; IR<=32'bx;
      @(posedge clock); IR<=32'h20020005; aluop<=2'b00;
      @(posedge clock); IR<=32'h2003000c; aluop<=2'b00;
      @(posedge clock); IR<=32'h2067fff7; aluop<=2'b00;
      @(posedge clock); IR<=32'h00e22025; aluop<=2'b10;
      @(posedge clock); IR<=32'h00642824; aluop<=2'b10;
      @(posedge clock); IR<=32'h00a42820; aluop<=2'b10;
      @(posedge clock); IR<=32'h10a70007; aluop<=2'b01;
      @(posedge clock); IR<=32'h0064202a; aluop<=2'b10;
      @(posedge clock); IR<=32'h10800001; aluop<=2'b01;
      @(posedge clock); IR<=32'h20050000; aluop<=2'b00;
      @(posedge clock); IR<=32'h00e2202a; aluop<=2'b10;
      @(posedge clock); IR<=32'h00853820; aluop<=2'b10;
      @(posedge clock); IR<=32'h00e23822; aluop<=2'b10;
      @(posedge clock); IR<=32'hac670044; aluop<=2'b00;
      @(posedge clock); IR<=32'h8c020050; aluop<=2'b00;
      #10 $finish;
   end
   assign funct = IR[5:0];
   aludec ALUdec(funct,aluop,alucontrol);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 20-11-2018

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 20-11-2018

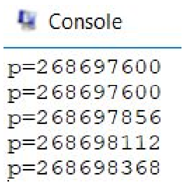
SOLUZIONE AGGIORNATA il 07-11-2019

ESERCIZIO 1

```
.data
base:    .word 0
       .word 0
freep:   .word 0
RTN:     .asciiz "\n"
puneq:   .asciiz "p="

.globl main
.text
myfree:
#---------------------------------------
# a0=ap,bp   t0=p, sizeof(Header)=8
   addi $a0,$a0,-8   # bp=ap-8
   la   $t1,freep    # &freep
   lw   $t0,0($t1)   # t1=p=freep
MF_INIFOR1: #scan list of free blocks
   slt  $t9,$t0,$a0# bp>?p
   lw   $t2,0($t0)   # t2=p->ptr
   slt  $t8,$a0,$t2  # bp<?p->ptr
   and  $t7,$t8,$t9  #
   bne  $t7,$0,MF_FINEFOR1
   slt  $t7,$t0,$t2# p>=?p->ptr
                   # => !(p<?p->ptr)
   nor  $t7,$t7,$0 #not(.)=> p<?p->ptr
   or   $t8,$t8,$t9  # (... || ...)
   and  $t7,$t7,$t8  # (... && (.))
   beq  $t7,$0,MF_FINEFOR1
   add  $t0,$t2,$0   # p=p->ptr
   j MF_INIFOR1
MF_FINEFOR1:

   lw   $t3,4($a0)   # bp->size
   add  $t4,$t3,$a0# bp+bp->size
   bne  $t4,$t0,MF_E1# (.)!=p->ptr
   lw   $t5,4($t0)   # p->ptr->size
   add  $t4,$t3,$t5  # bp->size+(.)
   sw   $t4,4($a0)   # bp->size=(.)
   lw   $t5,0($t0)   # p->ptr->ptr
   sw   $t5,0($a0)   # bp->ptr=(.)
   j MF_F1
MF_E1:
   sw   $t0,0($a0)   #bp->ptr=p->ptr
MF_F1:

   lw   $t6,4($t0)   # p->size
```

```
   add  $t7,$t6,$t0  # p+p->size
   bne  $t7,$a0,MF_E2# (.)!=bp
   lw   $t8,4($a0) # bp->size
   add  $t6,$t6,$t8# p->size+(.)
   sw   $t6,4($t0) # p->size=(.)
   lw   $t6,0($a0) # bp->ptr
   sw   $t6,0($t0) # p->ptr=(.)
   j MF_F2
MF_E2:
   sw   $a0,0($t0) # p->ptr=bp
MF_F2:
   sw   $t0,0($t1) # freep=p
   jr    $ra
#---------------------------------------
# a0=sz,  v1=p
alloc_and_print_pun:
   addi $v0,$0,9   # serv.9
   syscall #sbrk
   add $v1,$0,$v0  # salvo in v1

   la   $a0,puneq  # stampa msg
   addi $v0,$0,4   # serv.4
   syscall #print_str

   add  $a0,$0,$v1 # p
   addi $v0,$0,1   # serv.1
    syscall #print_int

   la   $a0,RTN    # stampa RTN
   addi $v0,$0,4   # serv.4
   syscall # print_str

   add  $v0,$v1,$0 # return(p)
   jr $ra

#---------------------------------------
main: #s0=p0, s1=p1, s2=p2, s3=p3
#---------------------------------------
# PROLOGO
   addi $sp,$sp,-24 # alloco frame
   sw   $ra,20($sp) # salvo OLD-ra
   sw   $fp,16($sp) # salvo OLD-fp
   add  $fp,$0,$sp  # NUOVO fp
   sw   $s3,12($fp) # salvo s3
   sw   $s2, 8($fp) # salvo s2
```

```
   sw   $s1, 4($fp) # salvo s1
   sw   $s0, 0($fp) # salvo s0

   la   $t0,base   # &base
   la   $t1,freep  # &freep
   sw   $t0,0($t0) # base.ptr=&base
   sw   $t0,0($t1) # freep=&base

   add  $a0,$0,$0  # sz=0 (HEAPtop)
   jal alloc_and_print_pun
   add  $s0,$v0,0  # salva p0

   addi $a0,$0,256 # sz=256
   jal alloc_and_print_pun
   add  $s1,$v0,0  # salva p1

   addi $a0,$0,256# sz=256
   jal alloc_and_print_pun
   add  $s2,$v0,0  # salva p2

   addi $a0,$0,256# sz=256
   jal alloc_and_print_pun
   add  $s3,$v0,0  # salva p3

   add  $a0,$s1,$0 # p1
   jal myfree
   add  $a0,$s2,$0 # p2
   jal myfree
   add  $a0,$s3,$0 # p3
   jal myfree
   add  $a0,$0,$0  # sz=0 (HEAPtop)
   jal alloc_and_print_pun
   add  $s0,$v0,$0 # salva p0

#---------------------------------------
# EPILOGO
   lw   $s0, 0($fp)  # ripristina s0
   lw   $s1, 4($fp)  # ripristina s1
   lw   $s2, 8($fp)  # ripristina s2
   lw   $s3,12($fp)  # ripristina s3
   lw   $ra,20($fp)  # RIPRISTINA ra
   lw   $fp,16($fp)  # RIPRISTINA fp
   addi $sp,$sp,24   # DEALLOCO FRAME
   addi $v0,$0,10  #serv.10
   syscall #exit
```



```
p=268697600
p=268697600
p=268697856
p=268698112
p=268698368
```

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=96/8/3=4,   XM=X/B,   XS=XM%S,   XT=XM/S:

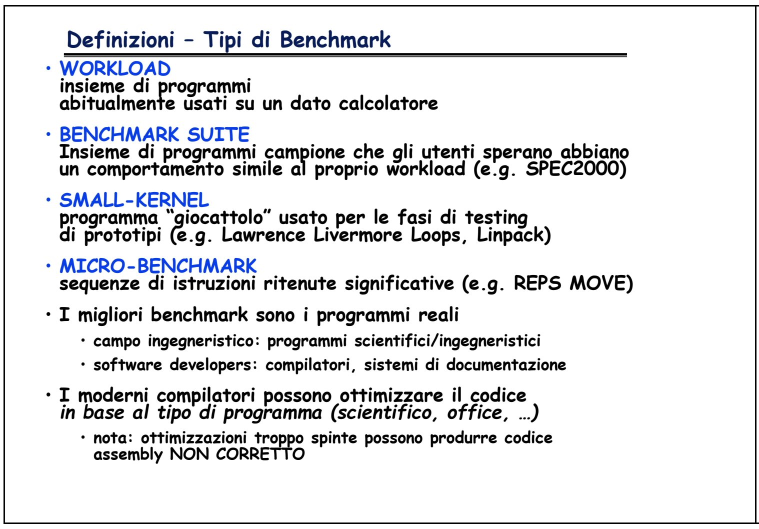
```
A=3, B=8, C=96, RP=LRU, Thit=4, Tpen=40, 20 references:
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  755   94  23   2   3  0 [2]:2,0,0 [2]:0,0,0  [2]:23,-,-
=== W  773   96  24   0   5  0 [0]:2,0,0 [0]:0,0,0  [0]:24,-,-
=== R  715   89  22   1   3  0 [1]:2,0,0 [1]:0,0,0  [1]:22,-,-
=== W  719   89  22   1   7  1 [1]:2,0,0 [1]:1,0,0  [1]:22,-,-
=== R  722   90  22   2   2  0 [2]:1,2,0 [2]:0,0,0  [2]:23,22,-
=== W  747   93  23   1   3  0 [1]:1,2,0 [1]:1,0,0  [1]:22,23,-
=== R  718   89  22   1   6  1 [1]:2,1,0 [1]:1,0,0  [1]:22,23,-
=== W  649   81  20   1   1  0 [1]:1,0,2 [1]:1,0,0  [1]:22,23,20
=== R  734   91  22   3   6  0 [3]:2,0,0 [3]:0,0,0  [3]:22,-,-
=== W  748   93  23   1   4  1 [1]:0,2,1 [1]:1,1,0  [1]:22,23,20
=== R  777   97  24   1   1  0 [1]:2,1,0 [1]:0,1,0  [1]:24,23,20 (out:  XM=89 XT=22 XS=1 )
=== W  719   89  22   1   7  0 [1]:1,0,2 [1]:0,1,0  [1]:24,23,22 (out:  XM=81 XT=20 XS=1 )
=== R  683   85  21   1   3  0 [1]:0,2,1 [1]:0,0,0  [1]:24,21,22 (out:  XM=93 XT=23 XS=1 )
=== W  643   80  20   0   3  0 [0]:1,2,0 [0]:0,0,0  [0]:24,20,-
=== R  791   98  24   2   7  0 [2]:0,1,2 [2]:0,0,0  [2]:23,22,24
=== W  744   93  23   1   0  0 [1]:2,1,0 [1]:0,0,0  [1]:23,21,22 (out:  XM=97 XT=24 XS=1 )
=== R  770   96  24   0   2  1 [0]:2,1,0 [0]:0,0,0  [0]:24,20,-
=== W  745   93  23   1   1  1 [1]:2,1,0 [1]:1,0,0  [1]:23,21,22
=== R   61    7   1   3   5  0 [3]:1,2,0 [3]:0,0,0  [3]:22,1,-
=== W  794   99  24   3   2  0 [3]:0,1,2 [3]:0,0,0  [3]:22,1,24
-------------------------------------
P1 Nmiss=15   Nhit=5   Nref=20   mrate=0.750000   AMAT=34
```

LISTA BLOCCHI USCENTI:

CONTENTUTI dei 4 SET al termine:

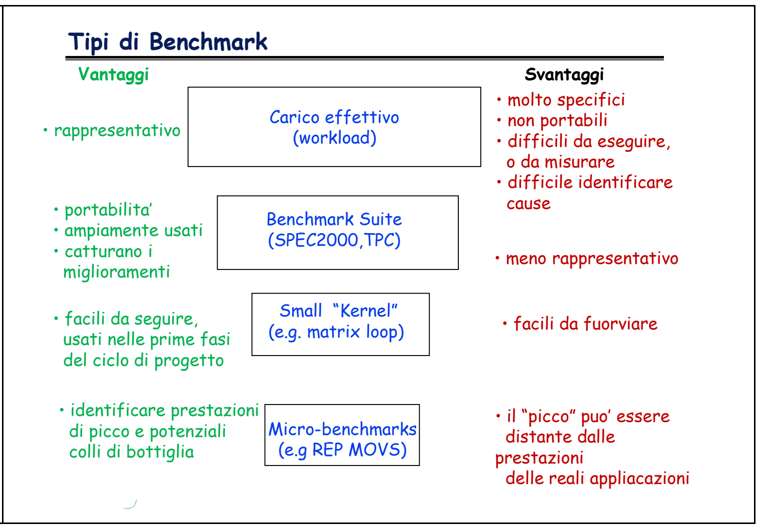
```
COMPITO di ARCHITETTURA DEI CALCOLATORI del 20-11-2018
SOLUZIONE  AGGIORNATA il 07-11-2019
```

ESERCIZIO 3



Definizioni – Tipi di Benchmark

- WORKLOAD
  insieme di programmi
  abitualmente usati su un dato calcolatore
- BENCHMARK SUITE
  Insieme di programmi campione che gli utenti sperano abbiano un comportamento simile al proprio workload (e.g. SPEC2000)
- SMALL-KERNEL
  programma "giocattolo" usato per le fasi di testing di prototipi (e.g. Lawrence Livermore Loops, Linpack)
- MICRO-BENCHMARK
  sequenze di istruzioni ritenute significative (e.g. REPS MOVE)
- I migliori benchmark sono i programmi reali
  - campo ingegneristico: programmi scientifici/ingegneristici
  - software developers: compilatori, sistemi di documentazione
- I moderni compilatori possono ottimizzare il codice *in base al tipo di programma (scientifico, office, …)*
  - nota: ottimizzazioni troppo spinte possono produrre codice assembly NON CORRETTO



Tipi di Benchmark

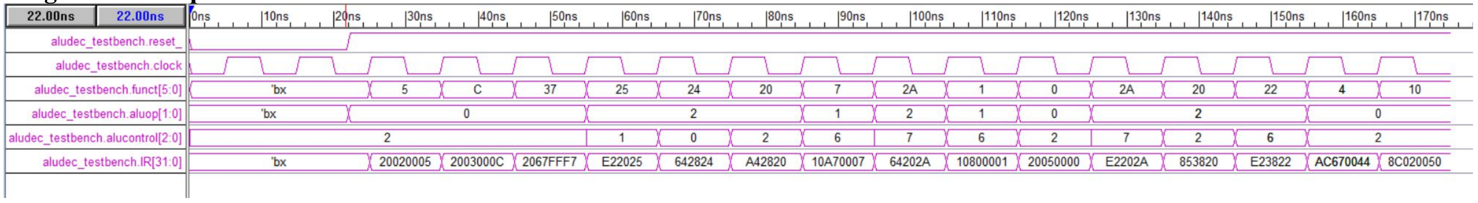
Vantaggi		Svantaggi
• rappresentativo	Carico effettivo (workload)	• molto specifici • non portabili • difficili da eseguire, o da misurare • difficile identificare cause
• portabilita' • ampiamente usati • catturano i miglioramenti	Benchmark Suite (SPEC2000,TPC)	• meno rappresentativo
• facili da seguire, usati nelle prime fasi del ciclo di progetto	Small "Kernel" (e.g. matrix loop)	• facili da fuorviare
• identificare prestazioni di picco e potenziali colli di bottiglia	Micro-benchmarks (e.g REP MOVS)	• il "picco" puo' essere distante dalle prestazioni delle reali appliacazioni

ESERCIZIO 7

Codice VERILOG:

```
module aludec(funct, aluop, alucontrol);
   input [5:0] funct; input [1:0] aluop;
   output [2:0] alucontrol;
   reg [2:0] alucontrol;
   always @(aluop or funct)
   casex(aluop)
      2'b00: alucontrol <= 3'b010; // add (for lw/sw/addi)
      2'b01: alucontrol <= 3'b110; // sub (for beq)
      default: casex(funct) // R-type instructions
         6'b100000: alucontrol <= 3'b010; // add
         6'b100010: alucontrol <= 3'b110; // sub
         6'b100100: alucontrol <= 3'b000; // and
         6'b100101: alucontrol <= 3'b001; // or
         6'b101010: alucontrol <= 3'b111; // slt
         default: alucontrol <= 3'bxxx; // ???
      endcase
   endcase
endmodule
```

Diagramma temporale:


22.00ns
22.00ns
0ns 10ns 20ns 30ns 40ns 50ns 60ns 70ns 80ns 90ns 100ns 110ns 120ns 130ns 140ns 150ns 160ns 170ns
aludec_testbench.reset_
aludec_testbench.clock
aludec_testbench.funct[5:0] 'bx 5 C 37 25 24 20 7 2A 1 0 2A 20 22 4 10
aludec_testbench.aluop[1:0] 'bx 0 2 1 2 1 0 2 0
aludec_testbench.alucontrol[2:0] 2 1 0 2 6 7 6 2 7 2 6 2
aludec_testbench.IR[31:0] 'bx 20020005 2003000C 2067FFF7 E22025 642824 A42820 10A70007 64202A 10800001 20050000 E2202A 853820 E23822 AC670044 8C020050