**COMPITO di ARCHITETTURA DEI CALCOLATORI del 06-09-2019** **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 FOGLI PERSONALI, NO TELEFONI, SMARTPHONE/WATCH, ETC**

**COGNOME**_____

**NOME**_____

**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) [12/30] Scrivere in assembly MIPS l'implementazione della funzione char *itoa(int n) che trasforma il numero naturale n (ovvero un intero >=0) in una stringa contenente i caratteri ASCII, il cui puntatore e' restituito come parametro di ritorno che rappresenta in lo stesso numero (es. Il naturale 123 dovra' diventare "123"). La funzione dovra' anche provvedere all'allocazione della memoria dinamica per memorizzare la stringa di uscita, inoltre dovra' essere realizzato un semplice programma main che legga da tastiera l'intero di ingresso e stampi la stringa di uscita chiamando la funzione "itoa" (promemoria: i caratteri ASCII delle cifre sono "0"->0x30, "1"-> 0x31, ...), **usando solo e unicamente istruzioni della tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** riportate qua sotto per riferimento.

**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) |
| 00+04 | **shift left logical** | `sllv $1,$2,10` | $1 = $2 << $3 | Shift left by variable |
| 00+06/00+07 | **shift 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+,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 / $f0=(single)$f1 | 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 | Serv.No.($v0) | INPUT Arguments | OUTPUT Args |
|---|---|---|---|
| **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 |

| Service Name | Serv.No.($v0) | INPUT Arguments | OUTPUT Arguments |
|---|---|---|---|
| **read_float** | 6 | --- | $f0=float |
| **read_double** | 7 | --- | $f0-f1=double |
| **read_string** | 8 | $a0=address of input buffer, $a1=max chars to read | --- |
| **sbrk** | 9 | $a0=Number of bytes to be allocated | $v0=pointer to allocated memory |
| **exit** | 10 | --- | --- |

**COMPITO di ARCHITETTURA DEI CALCOLATORI del 06-09-2019**

2) [5/30] 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: 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) [5/30] ] In un processore a 64 bit con memoria virtuale supportata da paginazione a 3 livelli con dimensione della pagina di 4KiB, vengono riservati 12 bit per indirizzare il primo livello, 20 bit per indirizzare il secondo livello e altri 20 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.

7) [8/30] Descrivere e sintetizzare in Verilog una rete sequenziale basata sul modello di Moore con flip-flop D, con un ingresso X su un bit e una uscita Z su un bit che riconosca le sequenze interallacciate 1,0,1,0. Il modulo TopLevel e'dato con sequenza di ingresso 0,0,1,1,0,1,1,0,1,1,0,0,1,0,0,0,1,0,1,1,0,1,0,0. **Tracciare il diagramma di temporizzazione** 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.

```
Module Toplevel;
  reg reset_; initial begin reset_=0; #22 reset_=1; #300; $stop; end
  reg clock; initial clock=0; always #5 clock<=(!clock);
  reg   X;
  wire  z=Xxx.z;
  wire [2:0] STAR=Xxx.STAR;
  initial begin X=0;
    wait(reset_==1);
    @(posedge clock); X<=0;@(posedge clock); X<=0;@(posedge clock); X<=1;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=1;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=1;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    @(posedge clock); X<=0;@(posedge clock); X<=1;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=1;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
      $finish;
    end
  XXX Xxx(X,Z,clock,reset_);
endmodule
```