COMPITO di ARCHITETTURA DEI CALCOLATORI 1 del 04-02-2013

MATRICOLA_____

COGNOME_____

NOME_____

MODULO RETI LOGICHE:

MODULO CALCOLATORI ELETTRONICI

1) [25/80] Scrivere in assembly MIPS un programma (utilizzando solo e unicamente istruzioni dalla tabella sul retro di questo foglio), rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS (riportate di seguito a detta tabella) che calcoli l'espressione 1+ x + x^2/2 + x^3/6 effettuando tutti i calcoli (compresa la "x") con numeri floating point in doppia precisione IEEE-754 e lasci il risultato all'indirizzo di memoria 0x1000.

2) [18/80] Il seguente programma C viene fatto girare (senza alcuna ottimizzazione) su un processore con una cache che ha blocchi da 64 Byte ed e' grande 512 Byte di dati:

```
int i, j, c, stride, array[1024];
...
for (i = 0; i < 5000; i++)
   for (j = 0; j < 1024; j = j + stride)
      c = array[j]+17;
```

   Se consideriamo solo l'attivita' della cache generata dai riferimenti al vettore 'array' e assumiamo che gli interi siano 4 Byte, quale sara' il miss rate avendo una cache ad accesso diretto e supponendo che 'stride' valga 256?
   Cosa accade se 'stride' vale 255? Se la cache fosse invece associativa a due vie, i due valori precedentemente trovati per il miss rate verrebbero influenzati?

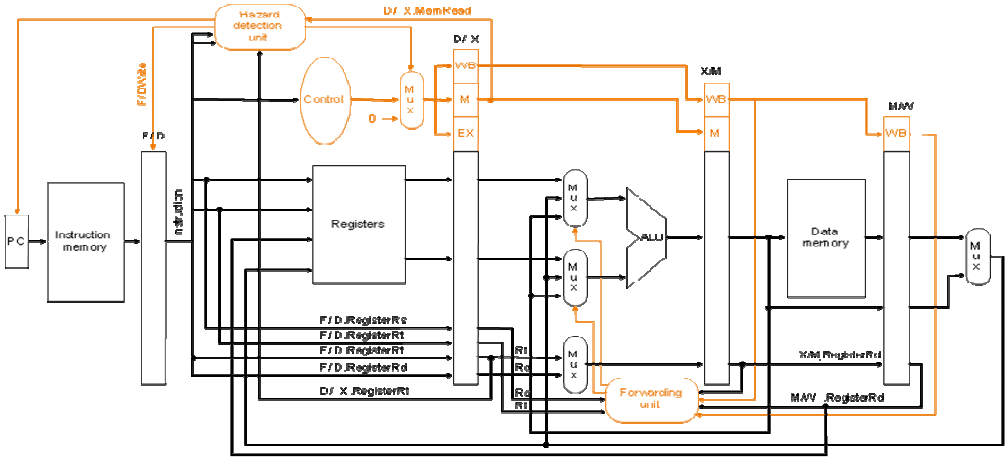
3) [8/80] Spiegare in dettaglio come avviene la paginazione inversa.

4) [5/80] Assumendo che l'istruzione di moltiplicazione impieghi 12 cicli e occupi il 15% delle istruzioni di un tipico programma, mentre il restante 85% delle istruzioni richiede in media 4 cicli per istruzione. Quale percentuale del tempo viene spesa dalla CPU per effettuare le moltiplicazioni?

5) [24/80] Nella seguente figura e' rappresentata una pipeline di un processore. Supponendo che su di essa si esegua il seguente codice:

```
sw   $5, 100($2)
sub  $6, $3, $2
and  $7, $2, $1
add  $8, $4, $3
or   $9, $2, $1
```

   Al ciclo 1, subito prima che si inizi ad eseguire queste istruzioni, lo stato del processore e': a) il PC vale 100 che e'l'indirizzo dell'istruzione sw ; b) ogni registro ha il valore iniziale di 20 piu' il numero del corrispondente registro (es. il registro $8 contiene 28); c) ogni parola in memoria contiene il valore iniziale 2000 piu' il valore dell'indirizzo di tale parola (es. la locazione Memory[8] ha il valore inizale 2008). Determinare il valore sui fili in uscita da ogni stadio al ciclo 5.



MIPS 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 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
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 absolute 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 (mfc1)	`mtc1 $1,$f2`	$f2=$1	Data from gen.reg. to C1 reg. (no conversion) (and viceversa)
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]	
store floating point (32bit)	`swc1 $f0,0($1)`	Memory[$1]←$f0	
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	Register 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	Register 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

Name	Usage
$f0, $f1, …, $f31	Single precision floating point registers
$f0, $f2, …, $f30	Double precision floating point 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_string	4	$a0=address of ASCIIZ string to print	---
Sbrk	9	$a0=Number of bytes to be allocated	$v0=pointer to the allocated memory