POLITECNICO DI TORINO

Computer architectures

Exam 03/02/2021

	MUHAMMAD TALHA FAROOQ 289537
Iniziato	mercoledì, 3 febbraio 2021, 14:45
Terminato	mercoledì, 3 febbraio 2021, 16:45
Tempo impiegato	2 ore

Domanda 1

Completo

Punteggio max.: 4

Let focus on the Reorder Buffer (ROB) existing in the architecture of some superscalar processors.

You are requested to

1. Explain how the ROB works (when an entry is allocated in it, when it is written, when it is read, when it is de-allocated)
2. Describe the ROB architecture, detailing the fields composing each entry
3. Summarize the advantages stemming from the adoption of the ROB.

1-
A Reorder buffer is used in a tomasulo algorithm for out of order instruction execution. It allows instructionto be committed in order.
There are three Stages of instruction:
'issue' 'Execute' 'Write result'
there is another stage "Commit". in this stage the resultof instructionwill be stored in the register. In the "write result" stage the result are just put in the re order buffer.
All content in this buffer then can be used when executing other instructions depending on these.
2-
There are additional field in every entry of the buffer:
-instruction type (jump, store to memory or register)
-Destination (either memory address or register no.)
-Result
-Validity (does the result already exist?)
3-
Advantages:
The benefits of the reorder buffer include precise EXCEPTIONS and easy ROLLBACK control of target adress misprediction. The ROB works by storing instructions in their original fetched order. It can also be accessed by the side since each reservation station has an additional parameters that points to instruction in the ROB. When jump prediction is not correct the ROB is cleared of all instruction and reservation station are re-intialized.

Domanda 2

Completo

Punteggio max.: 4

Let consider a MIPS64 architecture including the following functional units (for each unit the number of clock periods to complete one instruction is reported):

- Integer ALU: 1 clock period
- Data memory: 1 clock period
- FP arithmetic unit: 2 clock periods (pipelined)
- FP multiplier unit: 6 clock periods (pipelined)
- FP divider unit: 8 clock periods (unpipelined)

You should also assume that

- The branch delay slot corresponds to 1 clock cycle, and the branch delay slot is not enabled
- Data forwarding is enabled
- The EXE phase can be completed out-of-order.

You should consider the following code fragment and, filling the following tables, determine the pipeline behavior in each clock period, as well as the total number of clock periods required to execute the fragment. The value of the constant k is written in f10 before the beginning of the code fragment.

```
; ********** MIPS64 **********
;  for (i = 0; i < 10; i++) {
;      v4[i] = (v1[i]+v2[i])/(v3[i]*k);
;  }
```

Code	Comments	Clock cycles
.data		
v1: .double "10 values"		
v2: .double "10 values"		
v3: .double "10 values"		
.text		
main: daddui r1, r0, 0	r1 <- pointer	5
daddui r2,r0,10	r2 <= 10	1
loop: l.d f1,v1(r1)	f1 <= v1[i]	1
l.d f2,v2(r1)	f2 <= v2[i]	3
add.d f5, f1, f2	f5 <= v1[i] +v2[i]	1
l.d f3,v3(r1)	f3 <= v3[i]	1
mul.d f6, f3, f10	f6 <= f3*k	7
div.d f4, f5, f6	f4 <= (v1[i]+v2[i]) /(v3[i]*k)	8
s.d f4,v4(r1)	v4[i] <= f4	1
daddui r1,r1,8	r1 <= r1 + 8	1
daddi r2,r2,-1	r2 <= r2 - 1	1
bnez r2,loop		2
halt		1
total		6+(27*10)=276

main: daddui r1,r0,0	F	D	E	M	W																											
daddui r2,r0,10		F	D	E	M	W																										
loop: l.d f1,v1(r1)			F	D	E	M	W																									
l.d f2,v2(r1)				F	D	E	M	W																								
add.d f5, f1, f2					F	D	-	E	E	M	W																					
l.d f3,v3(r1)						F	-	D	E	-	M	W																				
mul.d f6, f3, f10								F	D	-	-	E	E	E	E	E	E	M	W													
div.d f4, f5, f6									F			D	-	-	-	-	-	E	E	E	E	E	E	E	E	M	W					
s.d f4,v4(r1)												F	-	-	-	-	-	D	-	-	-	-	-	-	-	E	M	W				
daddui r1,r1,8																		F	-	-	-	-	-	-	-	D	E	M	W			
daddi r2,r2,-1																										F	D	E	M	W		
bnez r2,loop																											F	-	D	E	M	W
halt																													F	D	E	M

Domanda 3

Completo

Punteggio max.: 6

Given a 4 x 4 matrix of WORD (i.e. 16 bits single data) SOURCE write a 8086 assembly program which rotates the rows of SOURCE from up to down by 1<=n<=3 positions and stores the result in the matrix DESTINATION, with n given by the user. The choice is yours about how to store the matrices in the memory. Please add significant comments to the code and instructions. If you have time, in order to get one additional point, provide the instructions to extend the program to consider n in the range -80<=n<=+120

Example:

Initial matrix SOURCE

A B C D

E F G H

I J K L

M N O P

if n=3 DESTINATION becomes

E F G H

I J K L

M N O P

A B C D

if n=1 DESTINATION becomes

M N O P

A B C D

E F G H

I J K L

```
.Model small
.Stack
.Data
MAT_A DB 16 DuP(ABCDEFGHIJKLMNOP)
MAT_B DB  16 DuP(?)
EXTRA DB 16 DuP(?)
.Code
.Stack up

MOV AH, o1H;      reading from keyboard
INT 21H;              n-->AL eg n=3
PUSH AX


CONT:
    MOV BX, 0

C1; MOV CX, MAT_A[BX]
    MOV MAT_B[BX], CX
    INC BX
    CMP BX,15
    JNE C1
    MOV BX,0
    MOV CX, 0

COPY;MOV DX, MAT_B
        MOV EXTRA[BX], DX
        INC BX
        DEC CX
        CMO CX,0
```

```
        JNZ COPY
        MOV BX,4
        MOV CX,12

C2; MOV DX, MAT_B[BX]
    MOV MAT_A[BX-4], DX
    INC BX
    DEC CX
    CMP CX,0
    JNZ C2
    MOV CX, 4
    MOV BX, 0

C3;MOV DX, EXTRA[BX]
   MOV MAT_A[12+BX], DX
   INC BX
   DEC CX
   CMP CX,0
   JNZ C3
   POP AX
   DEC AX
   CMPAX, 0
   JNZ CONT
.EXIT
```

Informazione

Click on the following links to open web pages with the ARM instruction set

http://www.keil.com/support/man/docs/armasm

https://developer.arm.com/documentation/ddi0337/e/introduction/instruction-set-summary?lang=en

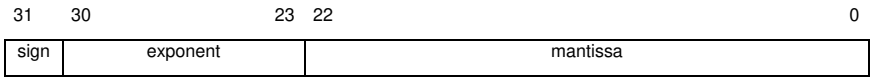
Note: Assembly subroutines must comply with the ARM Architecture Procedure Call Standard (AAPCS) standard (about parameter passing, returned value, callee-saved registers).

Domanda 4

Risposta non data

Punteggio max.: 9

The IEEE-754 SP standard expresses floating-point numbers in 32 bits:



Bit 31 is 0 if the number is positive, 1 if negative.

Write the addFPpositiveNumbers subroutine, which receives in input two 32-bit numbers, considers them as IEEE-754 SP floating point numbers, and returns their sum (in the same format). Bit 31 of the two input numbers is always 0 (i.e., the two numbers are positive).

In details, the subroutine implements the following steps:

1. take the mantissa of the two parameters
2. set the 23rd bit of the mantissa to 1
3. compare the two exponents. If they are equal, the exponent of the result is the same. If they are different:
   - the exponent of the result is the highest one
   - shift right the mantissa of the number with the lower exponent by as many position as the difference between the two exponents.
4. sum the two mantissas: this is the mantissa of the result. If the 24st bit of the mantissa of the result is 1:
   - shift right the mantissa of the result by one position
   - increment the exponent of the result by one.
5. set the 23rd bit of the mantissa of the result to 0.
6. combine the mantissa and the exponent to get the final result.

Example

parameter1 = 0100 0010 0100 1011 0000 0000 0000 0000

parameter2 = 0100 0001 1010 0100 0000 0000 0000 0000

1. mantissa1 = 0000 0000 0100 1011 0000 0000 0000 0000
   mantissa2 = 0000 0000 0010 0100 0000 0000 0000 0000
2. mantissa1 = 0000 0000 1100 1011 0000 0000 0000 0000
   mantissa2 = 0000 0000 1010 0100 0000 0000 0000 0000
3. exponent1 = 1000 0100
   exponent2 = 1000 0011
   - exponentResult = 1000 0100
   - mantissa2 = 0000 0000 0101 0010 0000 0000 0000 0000
4. mantissaResult = 0000 0001 0001 1101 0000 0000 0000 0000
   - mantissaResult = 0000 0000 1000 1110 1000 0000 0000 0000
   - exponentResult = 1000 0101
5. mantissaResult = 0000 0000 0000 1110 1000 0000 0000 0000
6. result = 0100 0010 1000 1110 1000 0000 0000 0000

Domanda 5

Risposta non data

Punteggio max.: 4

Initialize register r4 and r5 with two values expressing floating-point numbers according to the IEEE-754 SP standard.

Configure the SYSTICK timer in order to generate an interrupt every 2^{20} clock cycles.

Enter in an infinite loop. In the SYSTICK timer interrupt handler, sum the content of the r4 and r5 registers by calling the addFPpositiveNumbers subroutine and then store the result in r4.

The SYSTICK timer is configured by means of the following registers:

- Control and Status Register: size 32 bits, address 0xE000E010
- Reload Value Register: size 24 bits, address 0xE000E014
- Current Value Register: 24 bits, address 0xE000E018

The meaning of the bits in the Control and Status Register is as follows:

- Bit 16 (read-only): it is read as 1 if the counter reaches 0 since last time this register is read; it is cleared to 0 when read or when the current counter value is cleared
- Bit 2 (read/write): if 1, the processor free running clock is used; if 0, an external reference clock is used
- Bit 1 (read/write): if 1, an interrupt is generated when the timer reaches 0; if 0, the interrupt is not generated
- Bit 0 (read/write): if 1, SYSTICK timer is enabled; if 0, SYSTICK timer is disabled.

The Reload Value Register stores the value to reload when the timer reaches 0.

The Current Value Register stores the current value of the timer. Writing any number clears its content.

Domanda 6

Risposta non data

Non valutata

Here you can write:

- explanations on your answers, if you think that something is not clear
- your interpretation of the question, if you had any doubt about the formulation of the question
- any other comments that you want to let the professors know.

You can leave this space blank if you have no comments.
