POLITECNICO DI TORINO

Computer architectures

Exam 23/06/2021

	RAO MUHAMMAD WAJDAN 289627
Iniziato	mercoledì, 23 giugno 2021, 14:19
Terminato	mercoledì, 23 giugno 2021, 16:19
Tempo impiegato	2 ore

Domanda 1

Completo

Punteggio max.: 4

The Tomasulo architecture for superscalar processors with dynamic scheduling and speculation uses one or more Common Data Busses (CDBs).

You are requested to

1. Explain what the CDB is and where it is placed in the Tomasulo architecture, listing the modules able to write on it, and those reading from it
2. Detail when data are written in the CDB
3. List the advantages and disadvantages possibly stemming from the introduction of multiple instances of the CDB.

CDB is also known as common data bus . The main functionality of this CDB is to interface Reorder buffer (ROBs) and unified schedular also known as reservation center . This bus does not all approve any kind of discord and dissension in order to avoid any disturbance so that all corresponding outcomes will be transmitted to other reservation center .
There are some consequences which causes interupption or delay one or more chain due to reason that operand value are anticipated only when value are recorded on CDB or it has accessibililty only when its attainable in reorder buffer .
In first period the main unit perform its function to write all the possible outcome to comman data bus and afterward in the second period reorder buffer transmit these outcome to reservation station

The advantages of CDB are that it connects our reservation station to our functional units of system. Tamasulo allow out of order execution of instructions for the main purpose of our dynamic scheduling. For the reservation station, CDB do the computation of the values broadcast which may be needed. This type of advancements allow more better parallel processing of instructions that will in the other case can cause stalls. Without requisite of using FPU, main functional system have accessibility to all important operations.
Exceptions, Register renaming,instruction order are some key points for it.
Disadvantage is that its design is not limited to very pipeline structure

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: 4 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 values of the constants k1 and k2 are written in f10 and f11, respectively, before the beginning of the code fragment.

```
; ********************* MIPS64 ***********************
;  for (i = 0; i < 100; i++) {
;
        v4[i] = ((v1[i]/k1) + (v2[i]/k2))*v3[i];
;  }
```

Code	Comments	Clock cycles
.data		
v1: .double "100 values"		
v2: .double "100 values"		
v3: .double "100 values"		
v4: .double "100 values"		
.text		
main: daddui r1,r0,0	r1← pointer	5
daddui r2,r0,100	r2 <= 100	1
loop: l.d f1,v1(r1)	f1 <= v1[i]	1
div.d f2, f1, f10	f2 <= v1[i]/k1	9
l.d f3,v2(r1)	f3 <= v2[i]	0
div.d f4, f3, f11	f4 <= v2[i]/k2	8
add.d f5,f2,f4	f6 <= v1[i]/k1+v2[i]/k2	2
l.d f6,v3(r1)	f5 <= v3[i]	1
mul.d f7,f5,f6	f7 <= (v1[i]/k1+v2[i]/k2)*v3[i]	5
s.d f7,v4(r1)	v4[i] <= f7	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+(32*100)=3206

main: daddui r1,r0,0	F	D	E	M	W																											
daddui r2,r0,100		F	D	E	M	W																										
loop: l.d f1,v1(r1)			F	D	E	M	W																									
div.d f2, f1, f10				F	D	-	E	E	E	E	E	E	E	E	M	W																
l.d f3,v2(r1)					F	-	D	E	M	W																						
div.d f4, f3, f11							F	D	-	-	-	-	-	-	E	E	E	E	E	E	E	E	M	W								
add.d f5,f2,f4								F	-	-	-	-	-	-	D	-	-	-	-	-	-	-	E	E	M	W						
l.d f6,v3(r1)															F	-	-	-	-	-	-	-	D	E	-	M	W					
mul.d f7,f5,f6																							F	D	-	-	E	E	E	E	M	W	
s.d f7,v4(r1)																								F	-	-	D	E	-	-	-	M	W
daddui r1,r1,8																											F	D	-	-	-	E	M
daddi r2,r2,-1																												F	-	-	-	D	E
bnez r2,loop																																F	-

halt

Domanda 3

Completo

Punteggio max.: 6

Given a 5 x 5 matrix of bytes SOURCE representing unsigned numbers, write a 8086 assembly program which computes on 16 bits the sum of all cells excluding these on the main diagonal, i.e. upper left-to-lower-right diagonal, minus the sum of all the cells of the same main diagonal.

Please add significant comments to the code and instructions.

Friendly advice: before starting to write down the code, think at a possible (very) simple algorithm! The choice of the algorithm highly influences the complexity and length of the code.

Example:

matrix SOURCE

1 2 3 4 5
6 7 8 9 0
9 8 7 6 5
4 3 2 1 0
7 7 7 7 7

all cells excluding the main diagonal:
2+3+4+5+
6+8+9+0+
9+8+6+5+
4+3+2+0+
7+7+7+7= 102

all cells on the main diagonal
1+
7+
7+
1+
7= 23

Result (on 16 bits in two's complement) = 102-23 = 79

```
.MODEL
.STACK
.DATA
MySOURCE DB 1,2,3,4,5,6,7,8,9,7,6,5,4,3,2,1,0,7,7,7,7,7
. CODE
.STARTUP
MOV SI,0   ;;
MOV CX,2
MOV BX,4
MOV BP,0
LABEL2: ADD SI,1  ;;
LABEL1: MOV AL,MYSOURCE[SI]
    ADD SI,1  ;;;
MOV DL,MYSOURCE[SI]
ADD AX,DX
ADD BP,AX
ADD SI,1  ;;
DEC CX
CMP CX,0
```

```
JNZ LABEL1
MOV AL,MYSOURCE[SI]
ADD BP,AX
DEC BX
ADD SI,1  ; ;
CMP BX,0
JNZ LABEL2

MOV SI,0  ;;
MOV CX,2
MOV AL,0
MOV DL,0
MOV DI,0

LABEL3: MOV AL, MYSOURCE[SI] ;;
ADD SI,6  ;;
MOV DL,SOURCE[SI]  ;;
ADD AX,DX
ADD DI,AX
ADD SI,6  ;;
DEC CX
CMP CX,0
JNZ LAB3
ADD SI,6 ;;
MOV AL,SOURCE[SI]
ADD DI,AX
SUB BP,DI
.EXIT
END
```

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

Completo

Punteggio max.: 4

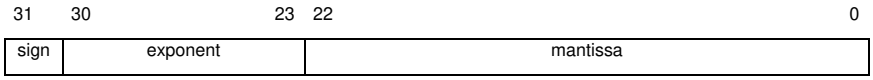
Write the getMaxAbsoluteValue function in C language, having the following prototype:

int getMaxAbsoluteValue(float parameter1, float parameter2)

The function returns 1 if the absolute value of parameter1 is higher than or equal to the absolute value of parameter2, 0 otherwise.

Then, write the Reset_Handler procedure in an assembly file that calls the C function passing two parameters.

Note 1: the two parameters in the Reset_Handler can be initialized to any value; anyway, it should be noted that their value is considered according to the IEEE-754 SP standard. This standard expresses floating-point numbers in 32 bits:



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

Note 2: it is important to add proper directives in the C and/or assembly file in order to guarantee the visibility of the getMaxAbsoluteValue function.

Content of the C file:

```
include<stdint.h>
include"LPC17xx.H"

getMaxAbsoluteValue(float para1,float para2)
{
int result;
paraa1= abs(para1);
paraa2=abs(para2);
if (paraa1>paraa2)
{
myresult=1;
}
else {
myresult=0;}
}
```

Content of the assembly file:

```
ResetHandler PROC
        EXPORT Reset_Handler
        IMPORT SystemInit
         IMPORT _main
         Para1 RN 0
         Para2 RN 1
         myresult RN 3
MOV32  Para1, 0x7FF45F00
MOV32 Para2, 0x7AC45120
B .
ENDP
```

Domanda 5

Completo

Punteggio max.: 9

Write the getAbsoluteDifference subroutine in ARM assembly, which receives in input two 32-bit numbers, considers them as IEEE-754 SP floating point numbers, and returns their absolute difference (in the same format).

In details, the subroutine implements the following steps:

1. pass the two parameters to the getMaxAbsoluteValue function. If the result of the function is 0, swap the two parameters
2. the exponent of the result is the same as the exponent of the first parameter; the sign of the result is 0
3. take the mantissa of the two parameters
4. set bit 23 of both mantissas to 1
5. if the exponent of the second parameter is lower than the exponent of the first parameter, shift right the mantissa of the second parameter by as many positions as the difference between the two exponents
6. check the sign of the two parameters.
   If the sign is the same:
   a) the mantissa of the result is the difference between the mantissa of the first parameter and the mantissa of the second parameter
   b) As long as bit 23 of the mantissa of the result is 0:
   - shift left the mantissa of the result by one position
   - decrement the exponent of the result by one

   Instead, if the two parameters have different sign:
   a) sum the two mantissas: this is the mantissa of the result
   b) If bit 24 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
7. set bit 23 of the mantissa of the result to 0
8. combine sign, mantissa and exponent to get the final result

Example:

parameter1 = 0100 0000 0100 1001 0000 1111 1101 1011

parameter2 = 1100 0001 1111 0110 1100 1011 1110 0100

1. the getMaxAbsoluteValue function returns 0, so:
   parameter1 = 1100 0001 1111 0110 1100 1011 1110 0100
   parameter2 = 0100 0000 0100 1001 0000 1111 1101 1011
2. exponentResult = 1000 0011
   signResult = 0
3. mantissa1 = 0000 0000 0111 0110 1100 1011 1110 0100
   mantissa2 = 0000 0000 0100 1001 0000 1111 1101 1011
4. mantissa1 = 0000 0000 1111 0110 1100 1011 1110 0100
   mantissa2 = 0000 0000 1100 1001 0000 1111 1101 1011
5. exponent1 = 1000 0011
   exponent2 = 1000 0000
   mantissa2 = 0000 0000 0001 1001 0010 0001 1111 1011
6. the parameters have different sign
   a) mantissaResult = 0000 0001 0000 1111 1110 1101 1101 1111
   b) mantissaResult = 0000 0000 1000 0111 1111 0110 1110 1111
   exponentResult = 1000 0100
7. mantissaResult = 0000 0000 0000 0111 1111 0110 1110 1111
8. result = 0100 0010 0000 0111 1111 0110 1110 1111

```
Resest Handler PROC
 Para1 RN 0
 Para2 RN 1
 Exponent1 RN 2
Exponent2 RN 3
 Exponentresult RN 4
 Subresult RN 5
Mantissa1 RN 6
Mantissa2 RN 7

MOV32   Para1,0x4049FDB
MOV32 Para2,0xC1F6CBE4



stop B stop
ENDP

Get absolute difference PROC

getmaxabsolutevaluedifference     SUB subresult,para1,para2
CMP subresult,0
BEQ Swap


Swap  MOV32 r3, para1
```

```
        MOV32 r4, para2
        MOV32 para1,r4
        MOV32 Para2,r3
   AND exponent1,para1,0xFFFC0000
  AND exponent2,para2,0xFFFC0000

   Mov32 exponentresult,exponent1
   MOv32 Signresult,0
   AND mantissa1,para1,0x003FFFFF
  AND matissa2, para2,0x003FFFFF
  ORR mantissa1, mantissa1,0x00400000
 ORR mantissa2,mantissa2,0x00400000
 Sub Subresult,exponent1,exponent2
  CMP exponent1,exponent2
  BHI Greatexp1
  BLO Lowexp2

 Greatexp1
   LSR Mantissa2,mantissa2,subresult

 AND Sign1,0x80000000
 AND Sign2,0x80000000
   CMP Sign1,sign2
   BEQ Signsame
   BNE Skip
 Signsame  Sub r10,mantissa1,mantissa2
                MOv32 mantissaresult,r10
               AND r10,mantissaresult,0x00400000
               CMP r10,0
              LSL mantissaresult,mantissaresult,1
               Sub exponentresult,exponentresult,1



    AND mantissaresult,mantissaresult,0xFF4FFFFF
    ORR r0,Exponentresult,mantissaresult
    ORR r0, r0,SignResult

   Skip
 MOV32  r0,

   ENDP
```

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.