Computer architectures

Exam 12/05/2022

ABDULLAH HASSAN
251425

Iniziato	giovedì, 12 maggio 2022, 13:20
Terminato	giovedì, 12 maggio 2022, 15:20
Tempo impiegato	2 ore

Domanda 1

Completo

Punteggio max.: 4

You are requested to

1. Explain what Loop Unrolling is, stating who is in charge of applying it
2. Describe the advantages and disadvantages it introduces
3. Report the code resulting from the application of Loop Unrolling to the following code:

```
for (i=0;i<10;i++ )
{
        y[i] = x[i]+ 5;
}
```

---

1. loop unrolling is a static technique to exploits the instruction level paralalism . it is done by the compiler in which it replicates the body of loop to analyse the code .
2. advantage is it reduce the number of iterations in the loop . code is optimized
disadvantage is it increase the size of code . not all the dependencies are known it compile time . , smart compiler is required .
3 . for(i= 0 ; i<5; i++)

```
    {
        y[i] = x[i] +5;
        y[i] = x[i]+5
        y[i] = x[i]+5
         y[i]= x[i]+5
          y[i]=x[i]+5

}
```

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 values 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]*k)/v3[i];
;  }
```

	Comments	Clock cycles
.data		
v1: .double “10 values”		
v2: .double “10 values”		
v3: .double “10 values”		
v4: .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]	1
mul.d f6, f2, f10	f6 <= v2[i]*k	7
l.d f3,v3(r1)	f3 <= v3[i]	0
add.d f7, f1, f6	f7 <= v1[i]+v2[i]*k	2
div.d f8, f7, f3	f8 <= (v1[i]+v2[i]*k) / v3[i]	8
s.d f8,v4(r1)	v4[i] <= f8	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+(25*10) =256

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									
mul.d f6, f2, f10	7				F	D		E	E	E	E	E	E	M	W		
l.d f3,v3(r1)	0					F		D	E	M	W						
add.d f7, f1, f6	2							F	D					E	E	M	W

div.d f8, f7, f3	8	F	D	E	E	E	E	E	E	E	E	M	W			
s.d f8,v4(r1)	1		F	D								E	M	W		
daddui r1,r1,8	1			F								D	E	M	W	
daddi r2,r2,-1	1											F	D	E	M	W
bnez r2,loop	2												F		D	E M W
halt	1														F	DEMW

Informazione

Click on the following link to open a web page with the 8086 instruction set:

http://www.jegerlehner.ch/intel/IntelCodeTable.pdf

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 3

Completo

Punteggio max.: 6

Given a 12 x 12 matrix of bytes SOURCE representing positive integers on 8 bits each one, write an 8086 assembly program which scans the main diagonal (upper left to bottom right corner) and computes & adds up to a final result variable named RESULT (the student has to determine on how many bits to prevent/minimize overflows, and has to explain the reasons about the choice taken) the single contributions i*j*c, where i and j are the row and column indexes of a diagonal element and c is its corresponding content. It is requested that the matrix is cut by columns. In other words RESULT = sum_ for i=j from 0 to 11 of i*j*SOURCE[i,j].

Remind that in a 12 x 12 matrix the indexes ranges is from 0 to 11. BRUTE FORCE APPROACH (i.e., without even a single "loop") IS NOT ACCEPTABLE. Comments are mandatory.

```
.MODEL SMALL
.STACK
.DATA
SOURCE DB 1,5,6,4,3,2,1, 0 , 9 , 10 , 11 ,12 ;CUT BY COLUMN                    ;
             DB 2,6,5,3,4,1,0,9,10,11,12,3 ;
             DB  6,5,12,1,0,1,2,3,4,5,6,7,9;
             DB  ;SIMILARYU

RESULT DB ?
.CODE
.STARTUP
;SOURCE MATRIX AND RESULT IS DEFINED AS DATA BYTE
MOV AX , 0
MOV BX , 0
MOV CX , 0  ; FOR DATABYE 2^8 IS THE RANGE IF IT EXCEED THAN RANGE THEN THERE IS OVERFLOW


LOOP1:
MOV AL , SOURCE [BX] ; FIRST ELEMENT MOV TO AL
ADD RESULT , AL  ; ADD VALUE IN AL TO RESULT
MOV AL , 0
ADD BX , 13 ; JMP TO NEXT DIAGNOL ELEMENT
INC CX ; INCREMETN IN COUTER EACH TIME
CMP CX , 12  ; COUNTER UP T0 12
JNE LOOP1 ; JMP IF NOT EQUAL TO LOOP1







.EXIT
END
```

Domanda 4

Completo

Punteggio max.: 8

The instruction set of ARM v7-M contains an instruction for the unsigned division with 32-bit operands: **UDIV Rd, Rn, Rm** where **Rd** is the quotient, **Rn** is the dividend, and **Rm** is the divisor. You have to write the **UDIV64** subroutine in ARM assembly language, which computes the unsigned division between a 64-bit dividend and a 32-bit divisor, and returns a 32-bit quotient. The subroutine receives in input:

- U: the upper half of the dividend
- L: the lower half of the dividend
- D: the divisor

The following algorithm is used to compute the quotient Q:

```
1. for i = 0 to 31:
2.       double the dividend: U = 2U, L = 2L
```

```
3.      if U >= D:
4.          U = U – D
5.          set i-th bit of Q to 1
6.      else:
7.          set i-th bit of Q to 0
```

At point 2, you have to use the left shift (not multiplication) to double the dividend. The most significant bit of L becomes the least significant bit of U.

In general, when you double a 64-bit value, you obtain a 65-bit value. In particular, the upper half of the dividend at points 3 and 4 can be a 33-bit value. You have to manage that case (hint: use the carry flag).

At points 5 and 7, the bits are counted from left to right. For example, bit 0 is the most significant one.

An example is given here considering 4-bit registers. You can easily extend it to the 32-bit case. The dividend is expressed with 8 bits; when you double it, the new value can have 9 bits (as in the first iteration of the example).

Example: U = 1001, L = 1010, D = 1100

Iteration i = 0

U = 10011, L = 0100

U >= D -> U = 10011 – 1100 = 0111 bit 0 of Q is 1

Iteration i = 1

U = 1110, L = 1000

U >= D -> U = 1110 – 1100 = 0010 bit 1 of Q is 1

Iteration i = 2

U = 0101, L = 0000

U < D -> bit 2 of Q is 0

Iteration i = 3

U = 1010, L = 0000

U < D -> bit 3 of Q is 0

The quotient is Q = 1100

```
U DCD 2_10011
L DCD 2_1010
D DCD 2_1100

RESET HANDLER
       LDR R0 , U
       LDR R1 , L
        LDR R2 , D
       BL UDIV64
STOP B STOP
ENDP
UDIV64
PROC{R4-R11 , LR}
LDR R4 , RO ;U
LDR R5 , R1 ; L
MOV R7 , #0

LOOP1:
LSL R4  , #1 ; 2U
LSL R5 , #1  ; 2L
 CMP R4 , R5 ; CMP U>=D
BHI UHIGH
BLO DHIGH
UHIGH  SUB R6 R4 , R5 ; U=U-D
LDR R10 , =0X1
LSL R10 , #32
ORR R6 , R6 , R10  ; SET Q T0 BIT 0 1
B SKIP
DGIHIH        LDR R10 , =0X1
               LSL R10 , 32
                AND R6 , R6 , R10
SKIP
 ADD  R7 , 1 ; ADD IN I VALUE
CMP R7 , #32 ; NUMBER OF TIME
```

```
BNE LOOP1
PROC{R4-R11 , PC}
ENDP
```

Domanda 5

Completo

Punteggio max.: 5

Write the UDIV64S subroutine, which extends the subroutine of the previous exercise by setting the flags in the Program Status Register (PSR). The bits in PSR are:

- Bit 31 (most significant bit): negative flag
- Bit 30: zero flag
- Bit 29: carry flag
- Bit 28: overflow flag
- Bit 27: sticky saturation flag
- Bit 26, 25, 15-10: interrupt-continuable instruction (ICI) bits, IF-THEN instruction status bit
- Bit 24: Thumb state
- Bit 8-0: exception number.

At the end of the procedure, the flags have to be:

- negative flag = first bit of the quotient
- zero flag = 1 if the quotient is zero; zero flag = 0 if the quotient is different from 0
- carry flag = 1 if the quotient can not be stored in 32 bits; carry flag = 0 otherwise.
- overflow flag = 0

The procedure does not change the other bits in PSR.

The value of the carry flag can be easily determined at the beginning of the subroutine by comparing the upper half of the dividend U with the divisor D. If U >= D, then the carry flag is 1, otherwise is 0.

Note: you do not have to copy duplicated code from the previous exercise. Write new instructions; you can write comments to indicate duplicated section of code.

```
LDR R8 , 0X100000 ; IF NEGATIVE
AND R8 , R8 , R7
CMP R8 , 0X10000
BEQ NEGATIVE
B SKIP

NEGATIVE:
LDR R8 , 0X100000
ORR R10 , R10 , R8 ; SET BIT 31 TO 1
SKIP
LDR R10 , 0X00000
CMP R7 , R10
QUOTIENTZERO ; SET ZERO BIT T0 1
B SKIP2
QUOTIENTZERO
LDR R8 , =0X020000000
ORR R10 , R8 , R7
SKIP2
LDR R8 , =0XFFFFFFFF  ; CARRY FLAG
CMP R7 , R8
BHI CARRYFLAG
B SKIP3
```

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.