Computer architectures

Exam 22/02/2021

	MUHAMMAD TALHA FAROOQ 289537
Iniziato	lunedì, 22 febbraio 2021, 08:56
Terminato	lunedì, 22 febbraio 2021, 10:56
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<MAX;i++ )
  {
y[i] = x[i]+ 5;
  }.
```

1-loop unrolling is a compiler optimization applied to certain kinds of loops to reduce the frequency of branches and loop maintenance instructions.it is easily applied to sequential array processing loops where the number of iterations is known prior to execution of the loop.
The transformation can be undertaken manually by the programmer or by the optimizing compiler.

2-Advantages:
-can be implemented dynamically if the no. of array elements is unknownat compile time.
-branch penalty is minimized is this technique.
- increases the program's speed by eliminating loop control instruction and loop test instructions.
-increases the program efficiency.
-reduces the loop overhead
-if statements in the loop are not dependent on each other, they can be executed in parallel.

Disadvantages:
-increased the program code size which can be undesirable which possiblely increase usage of register in a single iteration to store temporary variables also which may reduce preformance.
-unless performed transparently by an optimizing compiler, the code may become less readable.
-increased program size can also cause increase in instruction cache misses, which also effects the performance.

3- there is a loop of fetching the content from x array From the starting position and then addup 5 to content and save result in y array for maximun time given in the code

Domanda 2

Completo

Punteggio max.: 4

Let consider a processor including a Branch Target Buffer (BTB).

Assuming that the processor uses 32 bit addresses, each instruction is 4 byte wide, and the BTB is composed of 4 entries, you are requested to

1. Describe the architecture of the BTB in the specific case described above
2. Describe in details the behavior of a BTB, explaining when it is accessed, and which input and output information are involved with each access
3. Identify the final content of the BTB if
   - The BTB is initially full of 0s
   - The following instructions are executed in sequence

add.d f1,f2,f3 located at the address 0x00A50050

bnez r4,l1   located at the address 0x00A50054; the branch is taken, and the branch target address is 0x00A60050

mul.d f5,f6,f7 located at the address 0x00A60050

daddi r2,r2,-1 located at the address 0x00A60054

bez r2,l2   located at the address 0x00A60058; the branch is not taken

b l3   located at the address 0x00A6005B; the branch target address is 0x00A60AA0

1-The BTB is a Table composed of n entries,(in our case there are 6 entries) each composed of 2-m bit field:address and target.
2-the BTB is accessed each time an instruction is fetched.
using the least significant log n bits of instruction address an entry is selected. the address field of the entry is compared with the address of the fetched instruction , if they matched , the target is uploaded in the pc.
when the instruction is completed the BTB is possibly updated.
3-0X00A60058 is the final contect of the BTB

Domanda 3

Completo

Punteggio max.: 6

Given a 3 x 3 matrix of bytes SOURCE representing unsigned numbers, write a 8086 assembly program which computes (in circular buffer mode) the addition of each row element with the corresponding same column element in the row immediately below and stores the result on 16 bits in the same position of a matrix DESTINATION. The last row elements do add up with the corresponding first row elements (i.e. circular buffer mode). Please add significant comments to the code and instructions.

Example:

Initial matrix SOURCE

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

the following matrix DESTINATION is computed

```
5    7    9
11    13    15
8    10    12
```

```
.MODEL SMALL
.STACK
.DATA
MAT DB 1,2,3,4,5,6,7,8,9
ARR DB 8 DUP(?)
OUTPUT DW 8 DUP(?)
.CODE
.STARTUP

TAB1:MOV BX,0
        MOV CX,9
        MOV DX, MAT[BX]
        MOV ARR[BX],DX
        INC BX
        DEC CX
        CMPCX,0
        JNZ TAB1

TAB2:MOV BX,0
MOVCX,6
MOV SI,3
MOVDI,0
MOV AX,MAT[BX]
```

```
MOVDX,ARR[SI]
ADD AX, DX
MOV OUTPUT[DI],AX
INC BX
INC SI
ADD DI,2
DEC CX
CMP CX,0
JNZ TAB2

MOV BX,6
MOV SI,0
MOV CX,3
MOV DI,6

TAB3:MOV AX, ARR[BX]
MOV DX, MAT[SI]
ADD AX,DX
MOV OUTPUT[DI],AX
INC BX
INC SI
ADD DI,2
DEC CX
CMP CX,0
JNZ TAB3

.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.: 9

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

31	30 23	22 0
sign	exponent	mantissa

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

Some notable values are:

- all bits of exponent and mantissa are 0: zero
- exponent = 1111 1111, mantissa > 0: NaN (not a number).

Write the divideFPnumbers subroutine, which receives in input two 32-bit numbers (dividend and divisor, in this order), considers them as IEEE-754 SP floating point numbers, and returns their quotient (in the same format). As a simplification, it is assumed that the lowest 16 bits of the mantissa of the divisor are 0.

In details, the subroutine implements the following steps:

1. the sign of the result is 0 (positive) if dividend and divisor have the same sign, 1 (negative) otherwise
2. the exponent of the result is: exponent of dividend – exponent of divisor + 126
3. if the dividend is zero (i.e. all bits of exponent and mantissa are 0), then the mantissa of the dividend is 0. Otherwhise:
   a) take the mantissa of the dividend

   b) shift left the mantissa of the dividend by 8 positions

   c) set bit 31 of the mantissa of the dividend to 1.
4. if the divisor is zero (i.e. all bits of exponent and mantissa are 0), then the mantissa of the divisor is 0. Otherwhise:
   a) take the mantissa of the divisor

   b) shift right the mantissa of the divisor by 16 positions

   c) set bit 7 of the mantissa of the divisor to 1.
5. the mantissa of the result is: mantissa of dividend / mantissa of divisor. / is the unsigned integer division
6. if bit 24 of the mantissa of the result is set:
   a) shift right the mantissa of the result by 1 position
   b) add 1 to the exponent of the result computed at step 2
7. set bit 23 of the mantissa of the result to 0
8. combine sign, exponent, and mantissa to get the final result.

Example: dividend = 0100 0100 0101 0010 0010 0001 0000 0100 divisor = 1100 0000 1001 0101 0000 0000 0000 0000

1. sign of dividend = 0
   sign of divisor = 1
   sign of result = 1
2. exponent of dividend = 1000 1000
   exponent of divisor = 1000 0001
   exponent of result = 1000 1000 - 1000 0001 + 0111 1110 = 1000 0101
3. dividend is not zero
   a) mantissa of dividend = 0000 0000 0101 0010 0010 0001 0000 0100
   b) mantissa of dividend = 0101 0010 0010 0001 0000 0100 0000 0000
   c) mantissa of dividend = 1101 0010 0010 0001 0000 0100 0000 0000
4. divisor is not zero
   a) mantissa of divisor = 0000 0000 0001 0101 0000 0000 0000 0000
   b) mantissa of divisor = 0000 0000 0000 0000 0000 0000 0001 0101
   c) mantissa of divisor = 0000 0000 0000 0000 0000 0000 1001 0101
5. mantissa of result = 0000 0001 0110 1001 0000 0110 1110 0110
6. bit 24 of the mantissa of the result is set
   a) mantissa of result = 0000 0000 1011 0100 1000 0011 0111 0011
   b) exponent of result = 1000 0101 + 1 = 1000 0110
7. mantissa of result = 0000 0000 0011 0100 1000 0011 0111 0011
8. result = 1100 0011 0011 0100 1000 0011 0111 0011

```
DivideFPNumbers PROC
    PUSH{r4-r11,LR}

LDR r7,=SIGN
LDR r7,[r7]
AND r4,r0,r7     ;r0 is a dividend
AND r5,r1,r7
CMPr4,0
BEQ DividendPos
LDR R3,=MANTI
LDR r3,[r3]
AND r4,r0,r3
LSL r4,#8
AND r4,r4,r9     ;r9 IS SIGN which is 31 bit of MANTI to1
```

Domanda 5

Completo

Punteggio max.: 4

Write an exception handler that returns a NaN value when a division by zero occurs at step 5 of previous algorithm.

The division by zero exception is managed by means of the following registers:

- System Handler Control and State Register: size 32 bits, address 0xE000ED24
- Configuration Control Register: size 32 bits, address 0xE000ED14
- Usage Fault Status register: 16 bits, address 0xE000ED2A.

The meaning of the bits in the System Handler Control and State Register is as follows:

- Bit 18: enable usage fault handler
- Bit 17: enable bus fault handler
- Bit 16: enable memory management fault handler
- Bit 15: SVC pended
- Bit 14: Bus fault pended
- Bit 13: Memory management fault pended
- Bit 12: Usage fault pended
- Bit 11: Read as 1 if SYSTICK exception is active
- Bit 10: Read as 1 if PendSV exception is active
- Bit 8: Read as 1 if debug monitor exception is active
- Bit 7: Read as 1 if SVC exception is active
- Bit 3: Read as 1 if usage fault exception is active
- Bit 1: Read as 1 if bus fault exception is active
- Bit 0: Read as 1 if memory management fault is active.

The System Handler Control and State Register enables the following actions if the corresponding bit is set:

- Bit 9: Force exception stacking start in double word aligned address.
- Bit 8: Ignore data bus fault during hard fault and NMI.
- Bit 4: Trap on divide by 0
- Bit 3: Trap on unaligned accesses
- Bit 1: Allow user code to write to Software Trigger Interrupt register
- Bit 0: Allow exception handler to return to thread state at any level by controlling return value.

The bits in the Usage Fault Status register explains the cause of the usage fault:

- Bit 9: Division by zero and DIV_0_TRP is set.
- Bit 8: Unaligned memory access attempted and UNALIGN_TRP is set
- Bit 3: Attempt to execute a coprocessor instruction
- Bit 2: Invalid EXC_RETURN during exception return. Invalid exception active status. Invalid value of stacked IPSR (stack corruption). Invalid ICI/IT bit for current instruction.
- Bit 1: Branch target address to PC with LSB equals 0.
- Bit 0: Use of not supported (undefined) instruction.

Add a label to the last instruction of the divideFPnumbers subroutine. For example

```
lastInstruction     POP {r4-r7, PC}
```

The exception handler must check if the current usage fault exception is caused by a division by zero. If so, it moves 0x7FFFFFFF (corresponding to NaN) to the value or r0 stored in the stack and changes the value of PC stored in the stack such that the next instruction after the exception handler will be at label lastInstruction.

For simplicity, you can assume that in thread mode the same stack is used as in handler mode (i.e., the main stack pointer). When the exception is entered, register r0 is automatically saved at the top of the stack, and PC is saved in the stack with offset 24.

```
AREA data,DATA,READONLY

   CCR EQU
   USR EQU
   SCR EQU

AREA |.TEXT|, CODE, READONLY
DivideFPnUMBER PROC

  LDR r1,=TALaddress
  LDR {size} r2,[r1]
```

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.
