POLITECNICO DI TORINO

Computer architectures

Exam 22/02/2021

	COSIMO MICHELAGNOLI 291262
Iniziato	lunedì, 22 febbraio 2021, 08:55
Terminato	lunedì, 22 febbraio 2021, 10:55
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;
}.
```

The Loop Unrolling is a technique that try to maximise the ILP(instructions level parallelism) of a program. This technique can be done statically(by the compiler) or dynamically(by the processor). The main advantages of Loop Unrolling is the reduction of the number of branch instruction to reduce the possible control hazards(that reduce the CPI in a pipelined architecture) and the increasing number of basic block that can be easily managed for ILP. The main disadvantage is the increasing of the code size.
example:

```
        daddui r4, r0, HALFMAX    ; put MAX/2 into r4
loop    l.d r1, V1(r2)
        daddui r3, r1, 5
        s.d r3, V2(r2)
        daddui r2, r2, 4
        l.d r1, V1(r2)
        daddui r3, r1, 5
        s.d r3, V2(r2)
        daddui r2, r2, 4
        daddui r4, r4, -1
        bnez r4, loop
```

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

The BTB is a table compose of 2^n entry, so in this specific case the table is comose of 4 entries, where each entry is compose of 2M(8bytes) bit(with M equal to the size of the processor's address) address(4bytes) and target (4 bytes). Whenever an instruction is fetched, the BTB is accessed. If the instruction is a branch, then we consider the last n LSB(excluding the first two zero), and we access the corrisponding entry in the table. If the addess of the instruction match with the address into it's corresponding entry in te BTB, then we choose as a target for the branch the address in the target side of the entry. At the end of the branch execution the btb will be updated knowing if the brach was taken or not.

final content of BTB:
entry 1 0x00A50054, 0x00A60050
all the other antry will remain full of 0s

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

```
.data
SOURCE db 1,4,7 ,2,5,8 ,3,6,9
DESTINATION 9 DUP(?)
.code
.startup

xor si, si
mov cl, 2
xor di, di
xor bx, bx

nextColoumn:
cmp di, 18
jxx stop            ;if di is pointing beyond the destination matrix, then stop

finishColoumn:
mov ax, SOURCE[si]      ;get element in SI position of SOURCE
inc si
mov dx, SOURCE[si]   ;get next in SOURCE
```

```
add ax, dx
mov DESTINATION[di], ax   ; put solution
add di, 2                 ; next destination
dec cl
cmp cl, 0
Jxx finishColoumn         ; if cl is not 0 loop
inc SI                    ; inc SI for point the first element of next coloumn

;when we exit we have write the first two element of each coloum in destination and DI point to the third one BX pointing on the first of the coloumn ;and DX still has the last element of the coloumn so i just nee to use BX to grab the first element ad after the sum store in DI position

mov ax, SOURCE[bx]      ;  grab the starting value of the coloumn
add ax, dx              ; dx has the last value of coloumn
add bx, 3               ;next first element of the next coloumn
mov DESTINATION[di], ax
add di, 2               ; point next coloumn first element
jmp nextColoumn

stop:
        end

.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

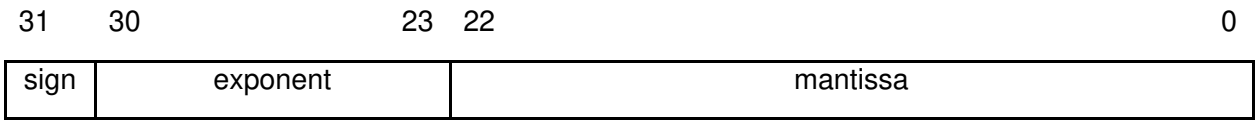
Note: Assembly subroutines must comply with the ARM Architecture Procedure Call Standard (AAPCS) standard (about parameter passing, returned value, callee-saved registers).

Domanda 4

Completo

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.

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
                    ; r0 has the first parameter DIVIDENT
                    ; r1 has the second parameter  DIVISOR
;point1
                    push{r4, lr}
                    ldr r12, =0x80000000          ; mask for the sign
                    and r2, r0, r12
                    eor r2, r1                    ; if is the same sign, then 0
                    and r2, r12
;point 2
                    ldr r12, =0x7F800000          ; 0111 1111 1 000 00..0
                    and r3, r12, r0               ; exponent divident in r3
                    and r12, r12, r1               ; exponent of divisor in r12
                    sub r3, r3, r12               ; divident - divisor in r3
                    add r3, #126

                    mov r4, r3                ; now exponent saved in r4
;point 3
                    ldr r12, =0x007FFFFF          ; mantissa mask
                    and r3, r12, r0               ; mantissa in r3 of divident
                    beq dividentIsZero
                    lsl r3, #8
                    orr r3, #0x80000000           ; set one 31 bit
```

```
;point 4
dividentisZero      and r12, r12, r1              ;mantissa in r12 of divisor
                    beq divisorIsZero
                    lsr r12, #16
                    orr r12, #0x00000080            , set 7th bit to 1
;point 5
divisorIsZero       div r3, r3, r2
;point 6
                    tst r3, #0x01000000           ;24th
                    beq theBitIsSet
                    b next
theBitIsSet         lsr r3, #1
                    add r4, #1
;point 7
next                and r3, #0xFF7FFFFF                ; 23th to 0
 ;last point
                    lsl r4, #23
                    orr r2, r4
                    orr r2, r3                    ;result in r2
lastInstruction     pop{r4, pc}
                    ENDP
```

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 myDATA, DATA, READONLY
SYSTEMHANDLER EQU 0XE00ED24
CONFIGUARTIONCONTROL EQU 0XE00ED14
USAGEFAULTSTATUSREG EQU OXE00ED2A

AREA |.text|, CODE, READONLY

reset_handler PROC
ldr r0, =SYSTEMHANDLER
```

```
ldr r1, [r0]
orr r1, #4
str r1, [r0]    ; enable usage fault
ldr r0, =CONFIGURATIONCONTROL
ldr r1, [r0]
orr r1, #0x00000010                ,enable trap on divide by 0
str r1, [r0]
;load two parameters on r0 and r1
Bl divideFPnumbers
B .
ENDP

usageFaultHandler PROC

ldr r0, =USAGEFAULTSTATUSREG
ldr r1, [r0]
tst r2, #0x00000200 ; check if ther was a division by 0
bneq  dummyImplementation       ;if is not zero we have other problem

mrs r0, MSP      ; mainstack pointer
ldr r1, #0x7FFFFFFF
str r1, [r0]      ;where should be r0
ldr r1, =lastInstruction
str r1, [r0, #24]
Bx lr

dummyImplementation
B .
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.