Computer architectures

EXAM 28/04/2020

	FEIHONG SHI 02lseov0_it_20_p1002_s274816
Iniziato	martedì, 28 aprile 2020, 14:50
Terminato	martedì, 28 aprile 2020, 17:00
Tempo impiegato	2 ore 10 min.

Informazione

before starting the exam, please click here to open the web page of the manual

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

Storico delle risposte

Passo	Data/Ora	Azione	Stato
1	28/04/2020 14:50	Iniziato	
2	28/04/2020 17:00	Visto	
3	**28/04/2020 17:00**	**Tentativo terminato**	

Domanda 1

Completo

Punteggio max.: 3,00

Let consider the Branch Prediction Mechanism based on the Branch Target Buffer (BTB).

You are requested to

1. Describe the architecture of a BTB
2. Describe in details the behavior of a BTB: when it is accessed, which input and output information are involved with each access

3. Assuming that the processor uses 32 bit addresses, each instruction is 4 byte wide, and the BTB is composed of 8 entries, clarify the content and size of the fields composing each BTB entry
4. Using the same assumptions of the previous point, identify the final content of the BTB if
    - The BTB is initially empty (i.e., full of 0s)
    - The following instructions are executed in sequence
        - l.d f1,v1(r1) located at the address 0x00A50050
        - bnez r2,l1 located at the address 0x00A50054; the branch is taken, and the branch target address is 0x00A60050
        - bez r4,l2 located at the address 0x00A60050; the branch is not taken.

1, There are three parts of the BTB, the first coloum is the address of branch instructions, the second is the next instruction address, the third is the Prediction of the branch instruction

2, when it is accessed, first find the entry of the branch, on the other words, to find which line matchs the branch, if can not find, just execute as normal. if find, check the prediction and put the next instruction address to the PC. If the prediction is wrong, change the cell value of the next instriction address, and change the cell value of prediction.

3, Because of the 8 entries, the first coloum is 3 bits wide. Because of the 32 bits address, the secod coloum is 32 bits. The third coloum is 1 bit. So, the size of each BTB entry is 36 bits.

4, 000   0x00A60054 0
    100   0x00A60050 1
     the other 6 entries are full of 0s

Storico delle risposte

Passo	Data/Ora	Azione	Stato
1	28/04/2020 14:50	Iniziato	Risposta non ancora data

Passo	Data/Ora	Azione	Stato
2	28/04/2020 17:00	Salvato: 1, There are three parts of the BTB, the first coloum is the address of branch instructions, the second is the next instruction address, the third is the Prediction of the branch instruction 2, when it is accessed, first find the entry of the branch, on the other words, to find which line matchs the branch, if can not find, just execute as normal. if find, check the prediction and put the next instruction address to the PC. If the prediction is wrong, change the cell value of the next instriction address, and change the cell value of prediction. 3, Because of the 8 entries, the first coloum is 3 bits wide. Because of the 32 bits address, the secod coloum is 32 bits. The third coloum is 1 bit. So, the size of each BTB entry is 36 bits. 4, 000 0x00A60054 0 100 0x00A60050 1 the other 6 entries are full of 0s	Risposta salvata
3	**28/04/2020 17:00**	**Tentativo terminato**	**Completo**

Domanda 2

Completo

Punteggio max.: 4,00

Let consider a superscalar MIPS64 architecture implementing dynamic scheduling, speculation and multiple issue and composed of the following units:

- An issue unit able to process 2 instructions per clock period; in the case of a branch instruction only one instruction is issued per clock period
- A commit unit able to process 1 instruction per clock period
- The following functional units (for each unit the number of clock periods to complete one instruction is reported):
  - 1 unit for memory access:1 clock period
  - 1 unit for integer arithmetic instructions: 1 clock period
  - 1 unit for branch instructions: 1 clock period
  - 1 unit for FP multiplication (pipelined): 6 clock periods
  - 1 unit for FP division (unpipelined): 8 clock periods
  - 1 unit for other FP instructions (pipelined): 2 clock periods
- 1 Common Data Bus.

Let also assume that

- Branch predictions are always correct
- All memory accesses never trigger a cache miss.

You should use the following table to describe the behavior of the processor during the execution of the first 2 iterations of a cycle composed of the specified instructions, computing the total number of required clock cycles. Registers f11 and f12 store two constants.

# iteration	instruction	Issue	EXE	MEM	CDB	COMMIT
1	l.d f1,v1(r1)	1	2	3	4	5
1	l.d f2,v2(r1)	1	3	4	5	6
1	l.d f3,v3(r1)	2	4	5	6	7
1	div.d f5, f3, f11	2	7		15	16
1	sub.d f4, f1, f2	3	6		8	17
1	add.d f6, f4, f5	3	16		18	19
1	div.d f7,f6,f12	4	19		27	28
1	s.d f7,v4(r1)	4	5			29
1	daddui r1,r1,8	5	6		7	30
1	daddi r2,r2,-1	5	7		8	31
1	bnez r2,loop	6	9			32
2	l.d f1,v1(r1)	7	8	9	10	33
2	l.d f2,v2(r1)	7	9	10	11	34
2	l.d f3,v3(r1)	8	10	11	12	35
2	div.d f5, f3, f11	8	27		35	36
2	sub.d f4, f1, f2	9	12		14	37
2	add.d f6, f4, f5	9	36		38	39

# iteration	instruction	Issue	EXE	MEM	CDB	COMMIT
2	div.d f7,f6,f12	10	39		47	48
2	s.d f7,v4(r1)	10	11			49
2	daddui r1,r1,8	11	12		13	50
2	daddi r2,r2,-1	11	13		14	51
2	bnez r2,loop	12	15			52

Storico delle risposte

Passo	Data/Ora	Azione	Stato
1	28/04/2020 14:50	Iniziato	Risposta non ancora data
2	28/04/2020 17:00	Salvato: # ITERATION INSTRUCTION ISSUE EXE MEM CDB COMMIT 1 l.d f1,v1(r1) 1 2 3 4 5 1 l.d f2,v2(r1) 1 3 4 5 6 1 l.d f3,v3(r1) 2 4 5 6 7 1 div.d f5, f3, f11 2 7 15 16 1 sub.d f4, f1, f2 3 6 8 17 1 add.d f6, f4, f5 3 16 18 19 1 div.d f7,f6,f12 4 19 27 28 1 s.d f7,v4(r1) 4 5 29 1 daddui r1,r1,8 5 6 7 30 1 daddi r2,r2,-1 5 7 8 31 1 bnez r2,loop 6 9 32 2 l.d f1,v1(r1) 7 8 9 10 33 2 l.d f2,v2(r1) 7 9 10 11 34 2 l.d f3,v3(r1) 8 10 11 12 35 2 div.d f5, f3, f11 8 27 35 36 2 sub.d f4, f1, f2 9 12 14 37 2 add.d f6, f4, f5 9 36 38 39 2 div.d f7,f6,f12 10 39 47 48 2 s.d f7,v4(r1) 10 11 49 2 daddui r1,r1,8 11 12 13 50 2 daddi r2,r2,-1 11 13 14 51 2 bnez r2,loop 12 15 52	Risposta salvata
3	**28/04/2020 17:00**	**Tentativo terminato**	**Completo**

Domanda 3

Completo

Punteggio max.: 4,00

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

In all subroutines, you can assume that there is no overflow with 32-bit operations.

The truncation of the Taylor series for the cosine function to a second-order approximation is:

$$cos\ y = 1 - \frac{y^2}{2} \qquad (1)$$

In equation (1), both the angle *y* and the value cos *y* are real numbers. They can be expressed as integer numbers to the detriment of precision. In particular, both angle *y* and value cos *y* are multiplied by 128 (i.e., each term in (1) is multiplied by 128 and the substitution $x = 128\ y$ is applied). We obtain:

$$128 \cdot cos\ x = 128 - \frac{x^2}{2 \cdot 128} \qquad (2)$$

Write the **cosine** subroutine that computes the value of 128 · cos x according to equation (2).

The subroutine receives in input the integer values x.

You can not use the division operation; instead, use a right shift to divide a number by power of 2.

The division (with right shift) must be computed after computing the numerator x^2 in (2).

```
        MOV R0, #X
        BL    cosine

cosine      PROC
  PUSH {R4-R8, R10, R11, LR}
        MOV R4, R0
        MUL R5, R4, R0
        LSR R6, R5, #8
        MOV R7, #128
        SUB R0, R7, R6
  POP  {R4-R8, R10, R11, LR}

            ENDP
```

Storico delle risposte

Passo	Data/Ora	Azione	Stato
1	28/04/2020 14:50	Iniziato	Risposta non ancora data
2	28/04/2020 17:00	Salvato: MOV R0, #X BL cosine cosine PROC PUSH {R4-R8, R10, R11, LR} MOV R4, R0 MUL R5, R4, R0 LSR R6, R5, #8 MOV R7, #128 SUB R0, R7, R6 POP {R4-R8, R10, R11, LR} ENDP	Risposta salvata
3	**28/04/2020 17:00**	**Tentativo terminato**	**Completo**

Domanda 4

Completo

Punteggio max.: 8,00

The truncation of the Taylor series for the sine function to a first-order approximation is sin y = y, i.e.

$$128 \cdot sin\ x = x \qquad (3)$$

by substituting the real value y with the integer number x as explained in previous exercice. However, this approximation is accurate only for small angles. In particular, we use the approximation if -0.5 rad < y < 0.5 rad, i.e., -64 < x < 64.

For bigger angles, we can use the double-angle formula:

$$sin\ y = 2 \cdot sin(\tfrac{y}{2}) \cdot cos(\tfrac{y}{2}) \qquad (4)$$

With the usual substitution, equation (4) becomes:

$$128 \cdot sin\ x = \tfrac{2}{128} \cdot [128 \cdot sin(\tfrac{x}{2})] \cdot [128 \cdot cos(\tfrac{x}{2})] \qquad (5)$$

Write the **doubleAngleSine** subroutine that computes the value of 128 · sin x as follows:

- according to equation (3) if -64 < x < 64
- according to equation (5) otherwise. In this case, **doubleAngleSine** calls the **cosine** subroutine passing x/2, and then it calls recursively **doubleAngleSine** passing x/2. The results returned by both functions are multiplied with each other and then multiplied by 2/128 (note: use a right shift instead of the division).

Note: the value returned by the cosine subroutine must be saved in a call-preserved register (according to AAPCS standard), in order to not be overwritten during the execution of the next **doubleAngleSine** call.

```
                MOV  R0, #X
                BL      doubleAngleSine


doubleAngleSine      PROC
        PUSH {R4-R8, R10, R11, LR}
                MOV  R4, R0
                CMP  R4, #-64
                BGT   checkPos1
                BLE   euqation51

checkPos1       CMP   R4, #64
                BLT    equation31
                BGE   equation51

equation31      B     return

equation51      LSR R4, #1
                MOV R0, R4
                BL   cosine
                MOV  R5, R0
                MOV  R0, R4
                BL      doubleAngleSine
                MUL R6, R0, R5
                LSR  R0, R6, #6
```

Storico delle risposte

Passo	Data/Ora	Azione	Stato
1	28/04/2020 14:50	Iniziato	Risposta non ancora data

Passo	Data/Ora	Azione	Stato
2	28/04/2020 17:00	Salvato: MOV R0, #X BL doubleAngleSine doubleAngleSine PROC PUSH {R4-R8, R10, R11, LR} MOV R4, R0 CMP R4, #-64 BGT checkPos1 BLE euqation51 checkPos1 CMP R4, #64 BLT equation31 BGE equation51 equation31 B return equation51 LSR R4, #1 MOV R0, R4 BL cosine MOV R5, R0 MOV R0, R4 BL doubleAngleSine MUL R6, R0, R5 LSR R0, R6, #6 return1 POP {R4-R8, R10, R11, PC} ENDP	Risposta salvata
3	**28/04/2020 17:00**	**Tentativo terminato**	**Completo**

Domanda 5

Completo

Punteggio max.: 6,00

The truncation of the Taylor series for the cosine function to a second-order approximation, as expressed in equation (1) and (2), is accurate only for small angles. In particular, we use the approximation if -1 rad < y < 1 rad, i.e., -128 < x < 128.

For bigger angles, we can use the double-angle formula:

$$cos\ y = 1 - [2 \cdot sin(\tfrac{y}{2})]^2 \qquad (6)$$

With the usual substitution, equation (6) becomes:

$$128 \cdot cos\ x = 128 - \tfrac{2}{128} \cdot [128 \cdot sin(\tfrac{x}{2})]^2 \qquad (7)$$

Write the **doubleAngleCosine** subroutine that computes the value of 128 · cos x as follows:

- according to equation (2) if -128 < x < 128. In this case, **doubleAngleCosine** calls the **cosine** subroutine passing x/2.
- according to equation (7) otherwise. In this case, **doubleAngleCosine** calls the **doubleAngleSine** subroutine passing x/2. Then, the result is squared, multiplied by 2/128 (note: use a right shift instead of the division) and subtracted from 128 as indicated in equation (7).

```
                MOV  R0, #X
                BL     doubleAngelCosine

doubleAngleCosine               PROC
        PUSH {R4-R8, R10, R11, LR}
                MOV  R4, R0
                CMP  R4, #-128
                BGT   checkPos2
                BLE     equation72

checkPos2    CMP  R4, #128
                BLT   divide2
                BGE  equation72

equation72    LSR   R0, #1
                BL      doubleAngleSine
                MOV  R5, R0
                MUL    R6, R5, R0
                LSR     R6, #6
                MOV    R7, #128
```

Storico delle risposte

Passo	Data/Ora	Azione	Stato
1	28/04/2020 14:50	Iniziato	Risposta non ancora data
2	28/04/2020 17:00	Salvato: MOV R0, #X BL doubleAngelCosine doubleAngleCosine PROC PUSH {R4-R8, R10, R11, LR} MOV R4, R0 CMP R4, #-128 BGT checkPos2 BLE equation72 checkPos2 CMP R4, #128 BLT divide2 BGE equation72 equation72 LSR R0, #1 BL doubleAngleSine MOV R5, R0 MUL R6, R5, R0 LSR R6, #6 MOV R7, #128 SUB R0, R7, R6 B return2 divider2 LSR R0, #1 BL cosine return2 POP {R4-R8, R10, R11, PC} ENDP	Risposta salvata
3	**28/04/2020 17:00**	**Tentativo terminato**	**Completo**

Domanda 6

Completo

Punteggio max.: 2,00

Write the **recursiveCosine** subroutine. The body of the subroutine is copied and pasted from the **doubleAngleCosine** subroutine. Then, apply the following changes:

- labels are renamed, to avoid duplication of labels
- call to **cosine** subroutine is replaced with the call to **recursiveCosine** subroutine.

Similarly, write the **recursiveSine** subroutine. The body of the subroutine is copied and pasted from the **doubleAngleSine** subroutine. Then, apply the following changes:

- labels are renamed, to avoid duplication of labels
- call to **cosine** subroutine is replaced with the call to **recursiveCosine** subroutine
- call to **doubleAngleSine** subroutine is replaced with the call to **recursiveSine** subroutine.

```
recursiveCosine             PROC
        PUSH {R4-R8, R10, R11, LR}
                MOV   R4, R0
                CMP   R4, #-128
                BGT   reCheckPos2
                BLE   reEquation72

reCheckPos2 CMP  R4, #128
                BLT   reDivid2
                BGE  reEquation72

reEquation72  LSR  R0, #1
                BL     doubleAngleSine
                MOV  R5, R0
                MUL  R6, R5, R0
                LSR   R6, #6
                MOV  R7, #128
                SUB   R0, R7, R6
                B       reReturn2

reDivid2        LSR   R0, #1
                BL      recursiveCosine
reReturn2
        POP   {R4-R8, R10,  R11, PC}
                                ENDP
```

Storico delle risposte

Passo	Data/Ora	Azione	Stato
1	28/04/2020 14:50	Iniziato	Risposta non ancora data

Passo	Data/Ora	Azione	Stato
2	28/04/2020 17:00	Salvato: recursiveCosine PROC PUSH {R4-R8, R10, R11, LR} MOV R4, R0 CMP R4, #-128 BGT reCheckPos2 BLE reEquation72 reCheckPos2 CMP R4, #128 BLT reDivid2 BGE reEquation72 reEquation72 LSR R0, #1 BL doubleAngleSine MOV R5, R0 MUL R6, R5, R0 LSR R6, #6 MOV R7, #128 SUB R0, R7, R6 B reReturn2 reDivid2 LSR R0, #1 BL recursiveCosine reReturn2 POP {R4-R8, R10, R11, PC} ENDP recusiveSine PROC PUSH {R4-R8, R10, R11, LR} MOV R4, R0 CMP R4, #-64 BGT reCheckPos1 BLE reEquation51 reCheckPos1 CMP R4, #64 BLT reEquation31 BGE reEquation51 reEquation31 B reReturn1 reEquation51 LSR R4, #1 MOV R0, R4 BL recursiveCosine MOV R5, R0 MOV R0, R4 BL recursiveSine MUL R6, R0, R5 LSR R0, R6, #6 reReturn1 POP {R4-R8, R10, R11, PC} ENDP	Risposta salvata
3	**28/04/2020 17:00**	**Tentativo terminato**	**Completo**

Domanda 7

Completo

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.

Thanks, professor. for your Patient and comfort or our exam!

Storico delle risposte

Passo	Data/Ora	Azione	Stato
1	28/04/2020 14:50	Iniziato	Risposta non ancora data
2	28/04/2020 17:00	Salvato: Thanks, professor. for your Patient and comfort or our exam!	Risposta salvata
3	**28/04/2020 17:00**	**Tentativo terminato**	**Completo**