EXERCISES/QUESTIONS on VLIW_STATIC_SCHEDULING_V2023

Question 1 (format open text)

Let's consider the following loop code to be ported on a simple **2-issue VLIW machine** with 2 fully pipelined functional units:

- 1 LD/ST Unit with 2 cycle latency
- 1 Integer ALU/BR UNIT with 1 cycle latency to next Int/LD/ST and with 2 cycle latency to next Branch

The Branch completes with 1 cycle delay slot (branch solved in ID stage).
Let's assume **static branch prediction BT/FNT with Branch Target Buffer**, so the branch delay slot can be used for next iteration
In the Register File, it is possible to read and write at the same address at the same clock cycle:

C code:

```
for(int i=0; i<N; i++)
        C[i] = A[i] + B[i] +INC1 +INC2;
```

Assembly code:

```
L1:     lw    $2, BASEA ($4)
        addi  $2, $2, INC1     // raw $2, waw $2
        lw    $3, BASEB ($4)
        addi  $3, $3, INC2     // raw $3, waw $3
        add   $5, $2, $3       // raw $2, raw $3
        sw    $5, BASEC ($4)   // raw $5
        addi  $4, $4, 4        // war $4
        bne   $4, $7, L1       // raw $4
```

Considering **one iteration** of the loop and **schedule** it on the 2-issue VLIW in the following table by using the **list-based scheduling.**

	SLOT1: LOAD/STORE ops	SLOT2: ALU/BRANCH ops
C1		
C2		
C3		
C4		
C5		
C6		
C7		
C8		
C9		

How long is the critical path?
What performance did you achieve in CPIas?
What performance did you achieve in VLIW clock cycles per iteration?
What code efficiency did you achieve?
What loop overhead did you achieve?

Open text answer

Feedback

	SLOT1: LOAD/STORE ops	SLOT2: ALU/BRANCH ops
C1	`lw $2,BASEA($4)`	`NOP`
C2	`lw $3,BASEB($4)`	`NOP`
C3	`NOP`	`addi $2, $2, INC1`
C4	`NOP`	`addi $3, $3, INC2`
C5	`NOP`	`add $5, $2, $3`
C6	`sw $5,BASEC($4)`	`addi $4, $4, 4`
C7	`NOP`	`NOP`
C8	`NOP`	`bne $4, $7, L1`
C9	`NOP`	`(br. delay slot)`

How long is the critical path?
8 cycles
(There is branch prediction, so the branch delay slot can be used for next iteration)

What performance did you achieve in cycles per iteration?
(# cycles) / iteration = 8 / 1 = 8

What performance did you achieve in CPIas?
CPIas =(# cycles) / IC = 8 / 8 = 1

What code efficiency did you achieve?
Code_eff = IC/ (# cycles * # issues) = 8 / (8 * 2) = 0.5
(there are 8 instructions and 8 NOPs in the VLIW code)

What loop overhead did you achieve? **2 cycles per iteration (cycles C7 and C8).**

Question 2 (format open text)

Let's consider the following loop code to be ported on a simple **3-issue VLIW machine** with 3 fully pipelined functional units:

- 1 Integer ALU with 1 cycle latency to next Integer/FP and with 2 cycle latency to next Branch
- 1 Memory Unit with 3 cycle latency
- 1 Floating Point Unit with 3 cycle latency (it can complete one add or one multiply per clock cycle)

The branch completes with 1 cycle delay slot (branch solved in ID stage)
In the Register File, it is possible to read and write at the same address at the same clock cycle

C Code:

```
for(int i=0; i<N; i++)
        C[i] = A[i]*A[i] + B[i];
```

Assembly Code:

```
loop:ld   f1, 0(r1)
     ld   f2, 0(r2)
     fmul f3, f1, f1
     fadd f4, f3, f2
     st   f4, 0(r3)
     addi r1, r1, 4
     addi r2, r2, 4
     addi r3, r3, 4
     bne  r3, r4, loop
```

Considering **one iteration** of the loop and **schedule** it on the 3-issue VLIW in the following table by using the **list-based scheduling (**please do not use any software pipelining or loop unrolling).
You do not need to write in NOPs (You can leave them as blank slot).

	Integer ALU	Memory Unit	FPU
C0			
C1			
C2			
C3			
C4			
C5			
C6			
C7			
C8			
C9			
C10			
C11			
C12			

How long is the critical path?
What performance did you achieve in cycles per loop iteration?
What performance did you achieve in CPI for one iteration?
What performance did you achieve in FP ops per cycle?
What code efficiency did you achieve?
What loop overhead did you achieve?

Open text answer

Feedback

	Integer ALU	Memory Unit	FPU
C0	`addi r1, r1, 4`	`ld f1, 0(r1)`	
C1	`addi r2, r2, 4`	`ld f2, 0(r2)`	
C2			
C3			`fmul f3, f1, f1`
C4			
C5			
C6			`fadd f4, f3, f2`
C7			
C8			
C9	`addi r3, r3, 4`	`st f4, 0(r3)`	
C10			
C11	`bne r3, r4, loop`		
C12	`(br delay slot)`		

How long is the critical path? **13 cycles (no branch prediction**

What performance did you achieve in CPI for one iteration?
CPI =(# cycles) / IC = 13 / 9 = 1.44

What performance did you achieve in cycles per loop iteration? **13/1**

What performance did you achieve in FP ops per cycle? **2/13**

What code efficiency did you achieve?
Code_eff = IC/ (# cycles * # issues) = 9 / (13 * 3) = 9 / 39 = 0.23
(there are 9 instructions and 30 NOPs in the VLIW code)

What loop overhead did you achieve? **3 cycles per iteration (cycles C10, C11, C12)**

Question 3 (format open text)

Please consider the following assembly code:

```
Loop:   LD F0, 0(R1)
        FADD F3, F0, F1
        FMULT F5, F0, F1
        FADD F7, F3, F5
        SD F7, 0(R1)
        LD F2, 4(R1)
        FADD F4, F2, F1
        FMULT F6, F2, F1
        FADD F8, F4, F6
        SD F8, 4(R1)
        ADD R1, R1,8
        BNE R1 R2 Loop
```

Details about the **4-issue VLIW** machine with 4 fully pipelined functional units:

- 1 Integer ALU with 1 cycle latency to next Integer/FP and with 2 cycle latency to next Branch
- 1 Memory Unit with 2 cycle latency
- 1 FP ADDER with 3 cycle latency
- 1 FP MULTIPLIER with 3 cycle latency

The branch is completed with 1 cycle delay slot (branch solved in ID stage).
In the Register File, it is possible to read and write at the same address at the same clock cycle

Please consider the following **schedule** on the 4-issue VLIW machine **(NOPs are not written).**

Please explain if/where there are any **error** in the schedule and explain how to solve them.

	Integer ALU	Mem Unit	FP ADDER	FP MULTIPLIER
C1		LD F0, 0(R1)		
C2		LD F2, 4(R1)		
C3			FADD F3, F0, F1	FMULT F5, F0, F1
C4			FADD F4, F2, F1	FMULT F6, F2, F1
C5			FADD F7, F3, F5	
C6			FADD F8, F4, F6	
C7				
C8	ADD R1,R1,8	SD F7, 0(R1)		
C9		SD F8, 4(R1)		
C10	BNE R1 R2 Loop			
C11	Br. delay slot			

Open text answer

Feedback:

There are **two errors** in the proposed schedule:

1) We need to introduce a full NOPs bundle (row) after cycle 4: The reason is that instruction **`FADD F7, F3, F5`** has a RAW dependence on F3 and F5 and the FP ADD/MUL Units require 3 cycle latency, therefore the FADD F7 instruction must be issued one clock later. Same reasoning applies to **`FADD F8, F4, F6`**: due to the RAW dependence on F4 and F6, it must be issued one clock later.
2) The instruction **`ADD R1, R1,8`** must be postponed by one cycle and therefore also the instruction **`BNE R1 R2 Loop`** must be postponed by one cycle.

The correct schedule (including the branch delay slot) requires 13 cycles instead of 11 as follows:

	Integer ALU	Mem Unit	FP ADDER	FP MULTIPLIER
C1		`LD F0, 0(R1)`		
C2		`LD F2, 4(R1)`		
C3			`FADD F3, F0, F1`	`FMULT F5, F0, F1`
C4			`FADD F4, F2, F1`	`FMULT F6, F2, F1`
C5				
C6			`FADD F7, F3, F5`	
C7			`FADD F8, F4, F6`	
C8				
C9		`SD F7, 0(R1)`		
C10	`ADD R1,R1,8`	`SD F8, 4(R1)`		
C11				
C12	`BNE R1 R2 Loop`			
C13	`Br. delay slot`			

Question 6 (format open text) from PART2_16/07/2020

Let's consider the following assembly code:

```
LOOP: LD $F0, 0($R1)
      ADDD $F4, $F0, $F2
      SD $F4, 0($R1)
      ADDUI $R1, $R1, 8
      BNE $R1, $R2, LOOP
```

*Show a software-pipelined version of this loop by **omitting** the start-up and finish up code*

Open text answer (max 100 words)

Solution:

```
LOOP: SD $F4, 0 ($R1)     /*store from iteration [i] corr. to index 0($R1) */
      ADDD $F4, $F0, $F2 /*add from iteration [i+1] corr. to index 8($R1) */
      LD $F0, 16 ($R1)    /*load from iteration [i+2] corr. to index 16($R1)*/
      ADDUI $R1, $R1, 8
      BNE $R1, $R2, LOOP
```

Feedback:

LD $F0, 0($R1)			
ADDD $F4, $F0, $F2	LD $F0, 8($R1)		
SD $F4, 0($R1)	ADDD $F4, $F0, $F2	LD $F0, 16($R1)	
	SD $F4, 8($R1)	ADDD $F4, $F0, $F2	....
		SD $F4, 16($R1)	....
			....

Question 5 (format open text) from PART2 of 25 Aug. 2021

*Let's consider the following assembly code where the registers **$R1** and **$R2** have been respectively initialized to **0** and **40:***

```
LOOP: LD $F0, 0 ($R1)
      ADD.D $F2, $F0, $F2
      SD $F2, 0 ($R1)
      ADDUI $R1, $R1, 8
      BNE $R1, $R2, LOOP
```

1) How many iterations of the LOOP are executed?
2) Write a software-pipelined version of this loop by ALSO INCLUDING THE START UP AND THE FINISH UP CODE.

Open text answer

Feedback:

1) The given LOOP is executed ***5 times.***
2) Software-pipelined version of the given loop including start-up & finish-up code:

```
           ADDI $R2, $R0, 24       /* to initialize $R2 to 24 */
START-UP:  LD $F0, 0 ($R1)
           ADDD $F2, $F0, $F2
           LD $F0, 8 ($R1)
SW-LOOP:   SD $F2, 0 ($R1)          /* store from iteration [i] */
           ADDD $F2, $F0, $F2       /* add from iteration   [i+1] */
           LD $F0, 16 ($R1)         /* load from iteration  [i+2] */
           ADDUI $R1, $R1, 8
           BNE $R1, $R2, LOOP       /* SW-LOOP is executed 3 times */
FINISH-UP: SD $F2, 0 ($R1)         /* here the value of $R1 is 24 */
           ADDD $F2, $F0, $F2
           SD $F2, 8 ($R1)
```

Question 2 (format open text) from PART2_31/08/2020

Let's consider the following loop code:

```
for (i=1; i<=100, i++) {
      X[i] = X[i-1] + Y[i-1];   /*S1*/
      Z[i] = Z[i-1] + X[i]      /*S2*/
}
```

1. *Are there any loop-carried dependence in the loop?*
2. *Could the loop iterations be made parallel?*

Open text answer (max 100 words)

Feedback

By definition, a loop-carried dependence occurs when data accesses in later iterations are dependent on data values produced in earlier iterations.

1) There are **2 loop-carried dependencies:**
   1. the value X[i] in S1 depends on X[i-1]
   2. the value Z[i] in S2 depends on Z[i-1].

2) Because of these two loop-carried dependencies, **the loop iterations cannot be made parallel.**

Please notice that the dependence of X[i] in S1 on Y[i-1] is not a loop-carried dependence because the vector Y[ ] is never modified in the loop. Also, the dependence of Z[i] in S2 on X[i] is not a loop-carried dependence.

Question 8 (format Multiple Choice – Multiple answers) from PART1 of 25 Aug 2021

Let's consider the following loop code:

```
for (i=1; i<=100, i++) {
      X[i] = X[i-1] + Y[i-1];     /*S1*/
      Z[i] = X[i-1] + Y[i-1]      /*S2*/
}
```

Are there any loop-carried dependence in the code?

(MULTIPLE ANSWERS)

Answer 1: One in S1 because X[i] depends on X[i-1];

Answer 2: One in S1 because X[i] depends on Y[i-1];

Answer 3: One in S2 because Z[i] depends on X[i-1];

Answer 4: One in S2 because Z[i] depends on Y[i-1];

Feedback:

Answer 1: One in S1 because X[i] depends on X[i-1]; (**TRUE**)

Answer 3: One in S2 because Z[i] depends on X[i-1]; (**TRUE**)

Please note that the dependence of Z[i] on Y[i-1] and the dependence of X[i] on Y[i-1] are not loop-carried dependences because the vector Y[ ] is never modified in the loop.

Question 13 (Format Multiple Choice – Multiple answers) 19 Jul. 2021 PART1

To obtain loop unrolling code we had to make the following transformations:

(MULTIPLE ANSWERS)

1 point

Answer 1: Reschedule the loop body by selecting instructions from different iterations of the original loop

Answer 2: Use register renaming to avoid name dependences

Answer 3: Replicate the loop body multiple times and adjust loop control code depending on the unrolling factor

Answer 4: Check if loop iterations are independent to each other

Answer 5: Apply global scheduling techniques operating across different basic blocks

Feedback:

Answer 2: Use register renaming to avoid name dependences **(TRUE)**

Answer 3: Replicate the loop body multiple times and adjust loop control code depending on the unrolling factor **(TRUE)**

Answer 4: Check if loop iterations are independent to each other **(TRUE)**