**Question #1**

Let consider the Instruction Level Parallelism (ILP).

You are requested to

1. Explain what ILP is and why it is important in pipelined architectures
2. Define what static and dynamic instruction re-scheduling are, listing the advantages / disadvantages they involve
3. Summarize what Loop Unrolling is and list the advantages / disadvantages it involves
4. Provide an example of loop unrolling, writing a piece of code before and after its application.
5. Explain why Loop Unrolling is not required in superscalar processors implementing dynamic scheduling with speculation.

ILP is the amount of parallelism between instructions. To achieve a better CPI we need to increase the ILP. Is is a characteristic of the code.

Rescheduling is the activity of changing the order of instruction executed by processor

Static rescheduling is a rescheduling made by compiler through an analysis of the code.

In this case the Compiler is more complex, and the code is optimized for a given processor architecture, then some dependencies are unknown a compile time, hence the performance are lower than dynamic rescheduling

While, dynamic rescheduling is made by processor during the execution of the program. The hardware is more complex, but the code is more portable and the performance is higher.

Loop unrolling is a static technique to increase the ILP between instruction through unrolling of loops, in this way the basic block within the loop can be replicated during a single cycle and so increase the chance to parallelize the block. Another advantage is the reduction of branches. The disadvantage of loop unrolling is the increase of the code size.

Loop unrolling isn’t required because superscalar processor with dynamic scheduling can execute instructions in out of order, hence instruction of different cycles is executed. This thanks also to Branch prediction and speculation.

**Question #2**

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: 4 clock periods (pipelined)
* FP divider unit: 6 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, using the table in the following page (where each column corresponds to a clock period), and determine the pipeline behavior in each clock period, as well as the total number of clock periods required to execute the fragment, reporting the result in the right column in the table below. The value of the constant k is written in f5 before the beginning of the code fragment.

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MIPS64 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; for (i = 0; i < 10; i++) {

; v4[i] = v1[i]/v2[i] - v3[i]^2;

; }

|  |  |  |
| --- | --- | --- |
| .data | comments | Clock cycles |
| 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 <= 20 | 1 |
| loop: l.d f1,v1(r1) | f1 <= v1[i] | 1 |
| l.d f2,v2(r1) | f2 <= v2[i] | 1 |
| l.d f3,v3(r1) | f3 <= v3[i] | 1 |
| div.d f4, f1, f2 | f4 <= v1[i] / v2[i] | 6 |
| mul.d f5,f3,f3 | f5 <= v3[i]^2 | 0 |
| sub.d f6, f4, f5 | f7 <= v1[i] / v2[i] - v3[i]^2 | 2 |
| s.d f6,v4(r1) | v4[i] <= f6 | 1 |
| daddui r1,r1,8 | r1 <= r1 + 8 | 1 |
| daddi r2,r2,-1 | r2 <= r2 - 1 | 1 |
| bnez r2, loop |  | 2 |
| halt |  | 1 |
| total |  | 176 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| daddui r1,r0,0 | F | D | X | M | w | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r2,r0,10 |  | F | D | X | M | W | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f1,v1(r1) |  |  | F | d | X | M | W | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f2,v2(r1) |  |  |  |  | F | D | X | M | W | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f3,v3(r1) |  |  |  |  |  | F | D | X | M | W | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| div.d f4, f1, f2 |  |  |  |  |  |  | F | D | X | X | x | X | X | X | M | W | 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| mul.d f5,f3,f3 |  |  |  |  |  |  |  | F | D | X | X | X | X | M | W |  | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sub.d f6, f4, f5 |  |  |  |  |  |  |  |  | F | D |  |  |  |  | X | X | M | W | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| s.d f6,v4(r1) |  |  |  |  |  |  |  |  |  | F |  |  |  |  | D | X |  | M | W | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D |  | X | M | W | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D | X | M | W | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| bnez r2, loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F |  | D | X | M | W | 2 |  |  |  |  |  |  |  |  |  |  |
| halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | f |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |

6+17\*10=176