**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. Explain why Loop Unrolling is not required in superscalar processors implementing dynamic scheduling with speculation.
5. ILP is Instruction Level Parallelism, from its name, we can know that it works on the instruction level. And at this level, instructions can be executed in parallel by dividing some stages into several parts, such as, we can divide IS stage into two parts.

Reason: Because the stages can divide into more parts, increase the parallel. That means we can execute more instructions in one clock in advantage.

1. Static: Archived by complier, re-scheduling the codes before it is executed.

Dynamic: Archived by hardware, re-scheduling the codes when the codes are running.

Static advantages:

1. It can complete before the codes are executed, so it can cost less time when running

Static disadvantages: lower precise.

Dynamic advantages: Higher precise.

Dynamic disadvantages: More cost when codes are running.

1. Loop Unrolling: By increasing the code size, to deduce the loop times

Advantages: deduce the loop times, then deduce the branch numbers, and also increase the parallel

Disadvantages: increase the code size

1. 1) Because it increases the size of codes, then it changes the address of below branch instructions. That lead to the below branch instructions can not find its precise cell in the precise table.

**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: 1 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 | 5 |
| loop: l.d f1,v1(r1) | f1 <= v1[i] | 5 |
| l.d f2,v2(r1) | f2 <= v2[i] | 5 |
| l.d f3,v3(r1) | f3 <= v3[i] | 5 |
| div.d f4, f1, f2 | f4 <= v1[i] / v2[i] | 10 |
| mul.d f5,f3,f3 | f5 <= v3[i]^2 | 8 |
| sub.d f6, f4, f5 | f7 <= v1[i] / v2[i] - v3[i]^2 | 6 |
| s.d f6,v4(r1) | v4[i] <= f6 | 6 |
| daddui r1,r1,8 | r1 <= r1 + 8 | 5 |
| daddi r2,r2,-1 | r2 <= r2 - 1 | 5 |
| bnez r2, loop |  | 5 |
| halt |  |  |
| total |  |  |

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