|  |  |
| --- | --- |
| **Computer Architectures**  **02LSEOV** | Delivery date:  October 23nd 2024, 11.59 PM |
| **Laboratory**  **3** | Expected delivery of lab\_03.zip must include:   * program\_1\_a.s, program\_1\_b.s, and program\_1\_c.s * This file, filled with information and possibly compiled in a pdf format. |

This lab will explore some of the concepts seen during the lessons, such as hazards, rescheduling, and loop unrolling. The first thing to do is to configure the WinMIPS64 simulator with the *Initial Configuration* provided below:

* *Integer ALU: 1 clock cycle*
* *Data memory: 1 clock cycle*
* Code address bus: 12
* Data address bus: 12
* FP arithmetic unit: pipelined, 4 clock cycles
* FP multiplier unit: pipelined, 6 clock cycles
* FP divider unit: not pipelined, 30 clock cycles
* Forwarding is enabled
* Branch prediction is disabled
* Branch delay slot is disabled

1. Enhance the assembly program you created in the previous lab called **program\_1.s**:

int m=1 /\* 64 bit \*/

double a, b

for (i = 31; i >= 0; i--){

if (i is a multiple of 3) {

a = v1[i] / ((double) m<< i) /\*logic shift \*/

m = (int) a

} else {

a = v1[i] \* ((double) m\* i))

m = (int) a

}

v4[i] = a\*v1[i] – v2[i];

v5[i] = v4[i]/v3[i] – b;

v6[i] = (v4[i]-v1[i])\*v5[i];

}

The variable b must be initialized with a desired value.

If you want to use the modulo operation to evaluate whether a number is a multiple of another, you can refer to the Barrett reduction algorithm:

*a mod n = a – [a/n]n*

* + 1. Manually detect the different data, structural, and control hazards that cause a pipeline stall.

Code inside the if statement is executed 11 times, the else statement 20 times.

.text

daddui r1, r0, 31 |FDEMW

daddui r2, r0, 248 | FDEMW

daddui r3, r0, 1 | FDEMW

l.d f21, fcs(r3) | FDEMW

daddui r9, r9, 1 | FDEMW

l.d f10, fcs(r0) | FDEMW

-- 10 cycles

cycle:

l.d f1, v1(r2) |FDEMW

l.d f2, v2(r2) | FDEMW

l.d f3, v3(r2) | FDEMW

bne r9, r0, notmul3 | FDEMW

| - possible delay slot

-- 4 cycles

dsllv r15, r15, r1 | FDEMW

mtc1 r3, f15 | FDEMW

cvt.d.l f15, f15 | FDEMW

-- 3 cycles

div.d f20, f1, f15 | FDEEEEEMW

| - 30\*E

cvt.l.d f16, f20 | FDSSSSEMW

mtc1 r16, f16 | FSSSSDEMW

daddui r9, r0, 2 | FDEMW

-- 33 cycles

j end\_if | FDEMW

-- 34 cycles

notmul3:

-- 1 cycle wasted by branch prediction failed

mtc1 r3, f15 | FDEMW

cvt.d.l f15, f15 | FDEMW

mtc1 r1, f16 | FDEMW

cvt.d.l f16, f16 | FDEMW

-- 4 cycles

mul.d f15, f15, f16 | FDEEEMW

| - 6\*E

mul.d f20, f15, f1 | FDSSEEEMW

| - 6\*E

cvt.l.d f16, f20 | FSSDSSEMW

mtc1 r16, f16 | FSSDEMW

daddui r9, r9, -1 | FDEMW

-- 15 cycles

end\_if: |

mul.d f4, f20, f1 | FDEEEMW

| - 6\*E

sub.d f4, f4, f2 | FDSSSEEMW

| - 4\*E

s.d f4, v4(r2) | FSSSSDEMW

-- 11 cycles

div.d f5, f4, f3 | FDEEEEMW

| - 30\*E

sub.d f5, f5, f21 | FDSSSEEEMW

| - 4\*E

s.d f5, v5(r2) | FSSSDSSEMW

-- 35 cycles

sub.d f6, f4, f1 | FSSDEEEMW

| - 4\*E

mul.d f6, f6, f5 | FDSSEEEMW

| - 6\*E

s.d f6, v6(r2) | FSSDSSEMW

daddi r2, r2, -8 | FSSDEMW

daddi r1, r1, -1 | FDEMW

slt r3, r2, r0 | FDEMW

-- 14 cycles

beq r3, r0, cycle | FDEMW

| - possible delay slot

halt |

-- 2 cycles

* + 1. Optimize the program by re-scheduling the program instructions to eliminate as many hazards as possible. Manually calculate the number of clock cycles for the new program (**program\_1\_a.s**) to execute and compare the results with those obtained by the simulator.

The instructions using integers have been moved between floating point operations, to properly fill the pipeline during RAW hazards

* + 1. Starting from **program\_1\_a.s**, enable the *branch delay slot* and re-schedule some instructions to improve the previous program execution time. Manually calculate the number of clock cycles needed by the new program (**program\_1\_b.s**) to execute and compare the results obtained with those obtained by the simulator.
    2. Unroll the program (**program\_1\_b.s**) 3 times; if necessary, re-schedule some instructions and increase the number of registers used. Manually calculate the number of clock cycles to execute the new program (**program\_1\_c.s**) and compare the results obtained with those obtained by the simulator.

Complete the following table with the obtained results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program**  **Clock cycle computation** | **program\_1.s** | **program\_1\_a.s** | **program\_1\_b.s** | **program\_1\_c.s** |
| **By hand** | 3940 | 3602 | 3320 | 3260 |
| **By simulation** | 2864 | 2640 | 2556 | 2510 |

1. Collect the Cycles Per Instruction (CPI) from the simulator for different programs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **program\_1.s** | **program\_1\_a.s** | **program\_1\_b.s** | **program\_1\_c.s** |
| **CPI** | 3.459 | 3.359 | 3.207 | 3.476 |

Compare the results obtained in 1) and provide some explanation if the results are different.

Eventual explanation:

I accidentally haven’t executed some instructions in parallel: considering that the loop executes 31 times, the clock cycles quickly exploded above the winmips64 simulation.  
  
  
In any case, the two results would have been different because the model architecture we studied can’t stall inside the EX stage, while winmips64 stalls in EX.