Computer Architectures
02LSEOV

Laboratory 3

Delivery date:
October 22nd 2024, 11.59 PM

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 labwillexplore 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 programyou 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];
}
```

a. Manually detect the different data, structural, and control hazards that cause a pipeline stall.
   Hazards:
   1. Structural hazard between mtc1 r8, f8 and cvt.d.l f8, f8
   2. RAW/WAW with ddiv r7,r2,r6 and dmul r7,r7,r6
   3. RAW with both ddiv r7,r6,r6,dmul r7,r7,r6 and beq r7,r2,multiple_3
   4. RAW between dmul r4, r3, r2 and mtc1 r4, f4
   5. Structural hazard between dmul r4, r3, r2 and mtc1 r4, f4
   6. RAW between mul.d f5, f2, f4 and cvt.l.d f6, f5

7. Structural hazard between mul.d f5, f2, f4 and cvt.l.d f6, f5
8. RAW/WAW between sub.d f6, f6, f2 and mul.d f6, f5, f1
9. RAW between s.d f6, v4(r1) and sub.d f6, f6, f2
10. RAW between div.d f7, f6, f3 and sub.d f6, f6, f2
11. RAW between sub.d f7, f7, f8 and s.d f7, v5(r1)
12. RAW/WAW between sub.d f7, f7, f8 and div.d f7, f6, f3
13. RAW between s.d f7, v5(r1) and sub.d f7, f7, f8
14. Structural hazard between sub.d f7, f7, f8 and s.d f7, v5(r1)
15. RAW between mul.d f9, f9, f1 and s.d f9, v6(r1)
16. Structural hazard between mul.d f9, f9, f1 and s.d f9, v6(r1)
17. RAW between slt r10, r2, 0 and beq r10, 0, cycle
18. RAW between div.d f5, f1, f4 and cvt.l.d f6, f5
19. Structural hazard between div.d f5, f1, f4 and cvt.l.d f6, f5
20. RAW between dmul r4, r3, r2 and r4, f4
21. Structural hazard between dmul r4, r3, r2 and r4, f4
22. RAW between mul.d f5, f2, f4 and cvt.l.d f6, f5
23. Structural hazard between mul.d f5, f2, f4 and cvt.l.d f6, f5

b. 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.

c. 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.

d. 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	3043	2953	2930	2827
By simulation	2922	2730	2677	2497

Computation by hand:

Program_1.s:

Fixed part outside the cycle:

daddui r1, r0, 8	F	D	E	M	W																									
daddui r2, r0,31		F	D	E	M	W																								
daddui r14, r0,3			F	D	E	M	W																							
dmul r1, r1, r2				F	D	E_M	E_M	E_M	E_M	E_M	E_M	M	W																	
ld r3, m(r0)					F	D	E	M	W																					
daddui r8, r0, 1						F	D	E	M	W																				
mtc1 r8, f8							F	D	E	M	W																			
daddui r7, r0, 2								F	D	E	M	W																		
cvt.d.l f8, f8									F	D	D	E	M	W																

Total CC $_{fixed}$ = 14

First part in cycle (before the if):

l.d f1, v1(r1)	F	D	D	E	M	W																								
l.d f2, v2(r1)		F	F	D	E	M	W																							
l.d f3, v3(r1)				F	D	E	M	W																						
beq r7, r0, multiple_3					F	D	E	M	W																					

Total CC $_{cycle\_first\_part}$ = 4 + 6 * 31 = 190 (because after the first iteration is terminated, for the others the program will “waste” 2 additional CC)

Not multiple of 3 part in the cycle:

j not_multiple_3	F	D	E	M	W																										
dmul r4, r3, r2			F	D	E_M	E_M	E_M	E_M	E_M	M	W																				
mtc1 r4, f4				F	D	D	D	D	D	E	M	W																			
cvt.d.l f4, f4					F	F	F	F	F	D	E	M	W																		
mul.d f5, f2, f4									F	D	E_M	E_M	E_M	E_M	E_M	M	W														
cvt.l.d f6, f5										F	D	D	D	D	D	E	M	W													
mfc1 r3, f6											F	F	F	F	F	D	E	M	W												

Total CC $_{not\_multiple}$ = 2/3 * 32 * (16 + 1 wasted CC) = 362.67 = 363

Multiple of 3 part in the cycle:

dsllv r3, r3, r2	F	D	E	M	W																										
mtc1 r3, f4		F	D	E	M	W																									
cvt.d.l f4, f4			F	D	E	M	W																								
div.d f5, f1, f4				F	D	E_D * 30 times						M	W																		
cvt.l.d f6, f5					F	D * 30 times						E	M	W																	
mfc1 r3, f6						F	F * 30 times					D	E	M	W																
j end												F	D	E	M	W															

Total CC $_{multiple}$ = 1/3 * 32 * 37 = 394.33 = 395

Last part of the cycle:

mul.d f6, f5, f1	F	D	E_M	E_M	E_M	E_M	E_M	M	W																						
sub.d f6, f6, f2		F	D	D	D	D	D	D	E_A	E_A	E_A	E_A	M	W																	
s.d f6, v4(r1)			F	F	F	F	F	F	D	D	D	D	E	M	W																
div.d f7, f6, f3									F	F	F	F	D	E_D * 30			M	W													
sub.d f7, f6, f3													F	D * 30			E_A	E_A	E_A	E_A	M	W									
s.d f7, v5(r1)														F * 30			D	D	D	D	E	M	W								
sub.d f9, f6, f1																	F	F	F	F	D	E_A	E_A	E_A	E_A	M	W				
mul.d f9, f9, f7																					F	D	D	D	D	E_M	E_M	E_M	E_M	M	W
s.d f9, v6(r1)																						F	F	F	F	D	D	D	D	E	M

s.d f9, v6(r1)	D	D	D	D	E	M	W																								
daddui r7, r7, 1		F	F	F	D	E	M	W																							
daddi r1, r1, -8					F	D	E	M	W																						
daddi r2, r2, -1						F	D	E	M	W																					
bne r7, r14, end2							F	D	E	M	W																				
slt r10, r2, r0									F	D	E	M	W																		
beq r10, r0, cycle										F	D	E	M	W																	

Total CC $_{\text{last\_part\_cycle}}$ = 32 * (64 + 1) = 2080 (the 1 is because of end2 if)

Total CC = 14 + 190 + 363 + 395 + 2080 + 1 = 3043

Program_1_a.s:
Total CC = 13 + 190 + 341 + 1824 + 384 + 1= 2953

Program_1_b.s:
Total CC = 2930

Program_1_c.s:
Total CC = 2827

2) 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	3.423	3.200	3.179	3.608

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

Eventual explanation:
The clock cycles **improve** enhancement after enhancement and this is a good thing, because it means the changes written in the code are real and effective. I also want to point out that instead of using Barrett Reduction to check the multiple of 3 condition, I used a counter that goes from 0 to 2 and then resets. This technique drastically reduce the clock cycles in every version of the program, by removing a ddiv and a dmul at every iteration of the cycle.

The difference in the computation of clock cycles by hand and then by simulation is because when calculating by hand I estimated that the multiple of 3 condition would have been reached just 1/3 of all the cases, so I used this form:

Total $CC_{\text{by hand}} = CC_{\text{fixed part}} + 1/3 * CC_{\text{multiple\_3}} + 2/3 * CC_{\text{not\_multiple\_3}}$

Also, the computation by hand differs from the simulator result because of the the behavior of WinMIPS, different from what we've seen during the lessons: the simulator, when it can, lets the instructions go through the EX stage before stalling them, instead of stalling them at the D stage (this was already seen in the previous labs).