ECE 4750 Computer Architecture

Section 6: Problem-Based Learning on Processors

http://www.csl.cornell.edu/courses/ece4750
School of Electrical and Computer Engineering
Cornell University

revision: 2022-09-29-23-30

List of Problems

Problem 1. Single-Cycle vs. Iterative Multipliers in a TinyRV1 Processor

In this problem, we will consider a pipelined TinyRV1 processor that uses stalling to resolve data hazards. The baseline design includes a single-cycle integer multiplier and the alternative design includes a 4-cycle unpipelined iterative integer multiplier.

Part 1.A Multiplication Microbenchmark

We will evaluate the performance of both the baseline and alternative design by considering the following assembly sequence:

```
loop:
 lw   x5,  0(x11)
 mul  x6,  x5, x12
 sw   x6,  0(x11)
 addi x11, x11, 4
 addi x13, x13, -1
 bne  x13, x0, loop
```

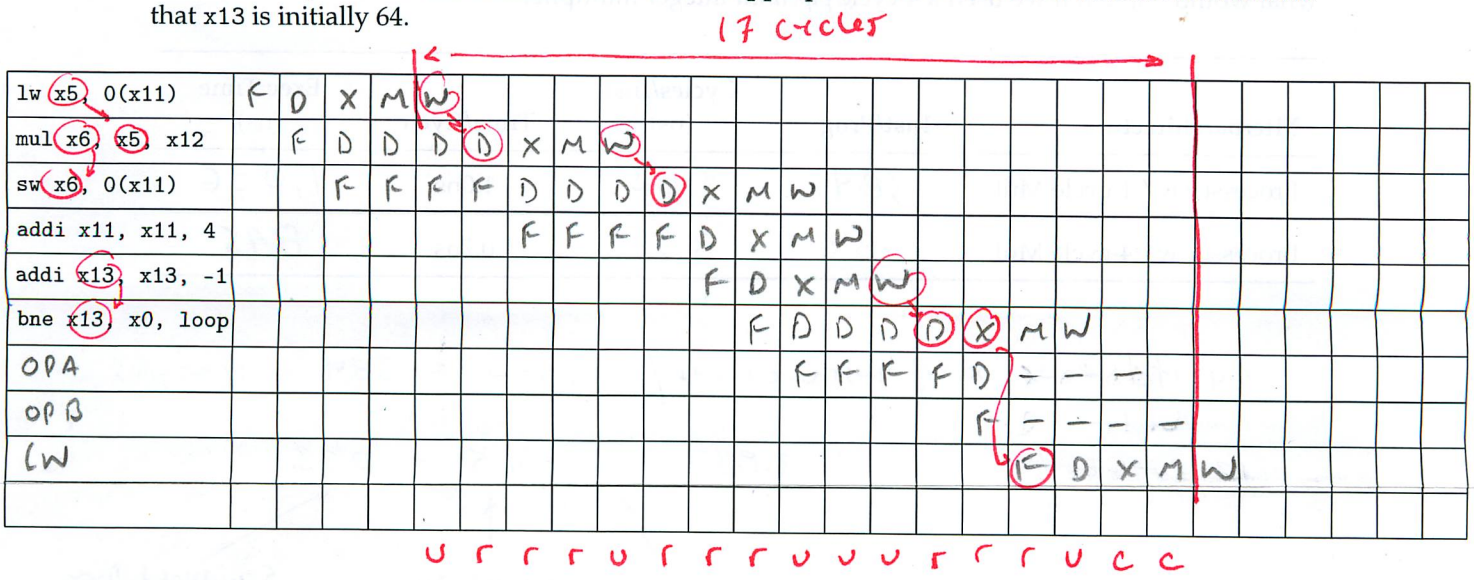
the original version of this problem had the arguments beginning at x11 but the one presented in class started at x10.

Write a C function that clearly represents the functionality of this assembly sequence. Recall that arguments are passed in registers x11–x17, the return value is stored to x10, the return address is stored in x1, and x5–x7, x28–x31 are used as temporary registers.

```
void vsmul( int* x, int a, int n )
{
  for (int i=0; i<n; i++)
    x[i] = x[i] * a;
}
```

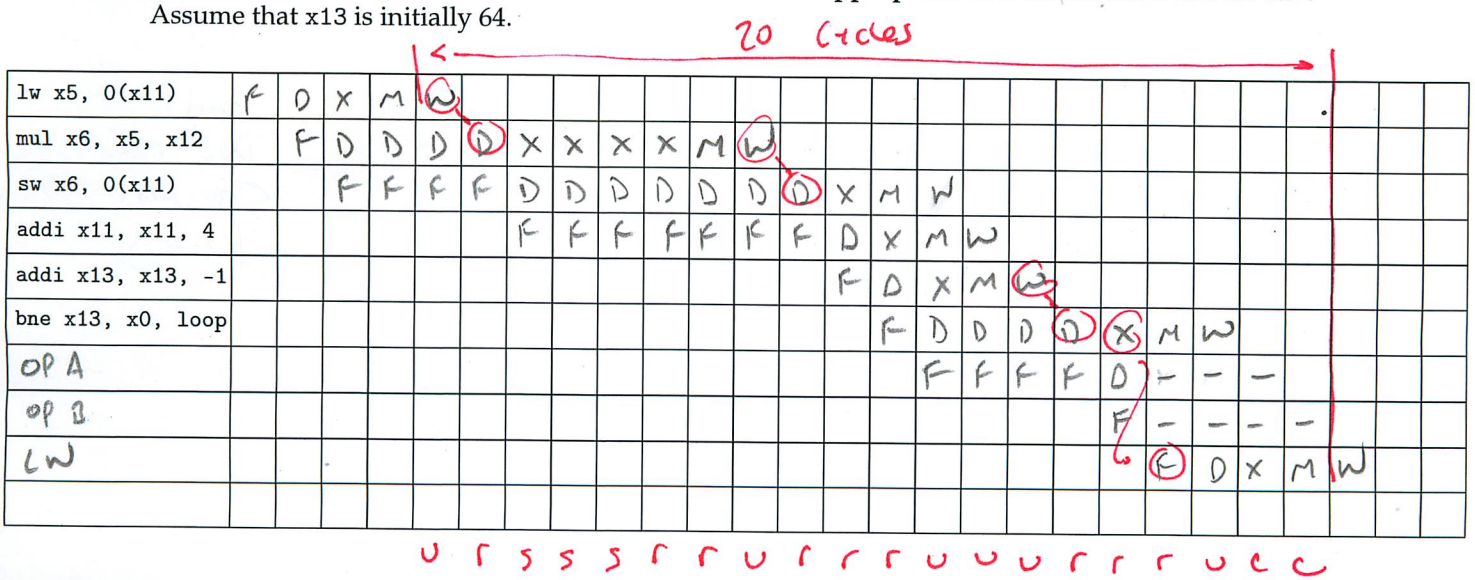
Part 1.B TinyRV1 Processor with Single-Cycle Integer Multiplier

The baseline design uses a single-cycle integer multiplier. Assume that this processor has a cycle time of 1ns, and that the critical path is through the single-cycle multiplier in the X stage. **Draw a pipeline diagram illustrating the first iteration of the loop.** Draw arrows in your pipeline diagram to indicate any data or control hazards. **Fill in the appropriate row in the table in Part 1.D.** Assume that x13 is initially 64.



Part 1.C TinyRV1 Processor with 4-Cycle Unpipelined Iterative Integer Multiplier

The alternative design uses a 4-cycle unpipelined iterative integer multiplier. Assume that multiplier is no longer on the critical path, and the cycle time for the processor decreases to 0.75ns. **Draw a pipeline diagram illustrating the first iteration of the loop.** Draw arrows in your pipeline diagram to indicate any data or control hazards. **Fill in the appropriate row in the table in Part 1.D.** Assume that x13 is initially 64.



Part 1.D Comparison of Processor Microarchitectures

Consider the results in the following table. Which microarchitecture has the highest performance on this microbenchmark? How would these results generalize to other workloads? Discuss some of the trade-offs in terms of area and energy between the processor microarchitectures. Consider what would happen if we used a 4-cycle *pipelined* integer multiplier.

Microarchitecture	Inst/Prog	Cycles/Inst	Time/Cycle	Exec Time (ns)
Processor w/ 1-cycle Mul	384	2.83	1.0ns	1,086
Processor w/ 4-cycle Mul	384	3.33	0.7ns	895

64 iterations x 6 instructions/iteration = 384 instructions

17 cycles / 6 instr = 2.83 = 1 + 1.5 + 0.33

1 ← useful work; 1.5 ← stalls due to RAW hazard; 0.33 ← squashes due to ctrl hazard

20 cycles / 6 instr = 3.33 = 1 + 1.5 + 0.33 + 0.5

0.5 ← stalls due to structural hazard

When professor Bracy discussed this in class, we did not discuss this being a structural hazard but rather a more expensive RAW hazard (which went from 3 cycles to 6 cycles).

The last thing we did was to discuss the AMAL of the memory accesses which Loads from an address A and then Stores back to the same address A. with a 1 cycle hit latency and a 10 cycle miss penalty, the following size cache lines will have the following AMALs:
1-word $ line: 1 + 50% miss rate x 10 cycle penalty = 6 cycles/access
4-word $ line: 1 + 12.5% miss rate x 10 cycle penalty = 2.25 cycles/access
16-word $ line: 1 + 3.125% miss rate x 10 cycle penalty = 1.3125 cycles/access