COEN 122: Computer Architecture
Homework #4

Tamir Enkhjargal

February 24, 2020

Question #1

Loop "unfolding" is a technique used by compilers to improve pipeline performance. Below is an example of a C loop and then the same loop after unfolding.

```
for(i = 0; i < 100; i++)
    a[i] = b[i];
```

```
for(i = 0; i < 100; i = i+2) {
    a[i] = b[i];
    a[i+1] = b[i+1];
}
```

The LEGV8 implementation of both loop versions would be (assumed initial value of X9 is 100)

```
top:    CBZ     X9, exit
        LDUR    X12, [X4, #0]
        STUR    X12, [X5, #0]
        ADDI    X4, X4, #8
        ADDI    X5, X5, #8
        SUBI    X9, #1
        B       top
exit:
```

```
top:    CBZ     X9, exit
        LDUR    X12, [X4, #0]
        STUR    X12, [X5, #0]
        LDUR    X12, [X4, #8]
        STUR    X12, [X5, #8]
        ADDI    X4, X4, #16
        ADDI    X5, X5, #16
        SUBI    X9, #2
exit:
```

a) How many cycles does a typical iteration of the original loop take?

   Seven cycles

b) Assuming forwarding hardware is used, what is the number of data stall cycles in the original loop?

   One stall between the LDUR and STUR.

c) Repeat for the unfolded loop

   In a single-cycle datapath, it will take Eight cycles for the unfolded loop. Then there will be three stalls between each LDUR/STUR/LDUR/STUR.

d) What is the advantage of unfolding you observe so far?

   The original loop: No forwarding is 7*100 = 700 cycles.
   The unfolded loop: No forwarding is 8*50 = 400 cycles.
   Unfolding leads to less cycles.

e) A smart compiler will replace X12 used in the second LDUR/STUR pair by another pair (e.g. X13), and then schedule the code to reduce or eliminate the load stall cycles. Show the resulting schedule code that achieves that purpose.

```
top:    CBZ     X9, exit
        LDUR    X12, [X4, #0]
        LDUR    X13, [X4, #8]
        SUBI    X9, #2
        STUR    X12, [X5, #0]
        STUR    X13, [X5, #8]
        ADDI    X4, X4, #16
        ADDI    X5, X5, #16
        B       top
exit:
```

Question #2

Solve Problem 4.27 in the textbook. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
ADD     X5, X2, X1
LDUR    X3, [X5, #4]
LDUR    X2, [X2, #0]
ORR     X3, X5, X3
STUR    X3, [X5, #0]
```

1. If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

```
ADD     X5, X2, X1
NOP
NOP
LDUR    X3, [X5, #4]
LDUR    X2, [X2, #0]
NOP
ORR     X3, X5, X3
NOP
NOP
STUR    X3, [X5, #0]
```

2. Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register X7 can be used to hold temporary values in your modified code.

   We can't change around the ordering of the instructions because these should be done in atomic order. First, LDUR depends on X5 from ADD. Second, ORR needs X3 from first LDUR. Third, STUR depends on X3 from ORR to be stored. STUR also can't come before ORR because it would overwrite the X5 value which ORR needs.

3. If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

   There would be a lack of ID/EX.MemRead and IF/IDWrite signals, which would have existed with a hazard detection unit. Since these don't exist, the program would run through all of the processes, but will work with stale data, if we didn't hard-code in the no operation instructions.

4. If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.59.

```
ADD     X5, X2, X1
LDUR    X3, [X5, #4]
LDUR    X2, [X2, #0]
ORR     X3, X5, X3
STUR    X3, [X5, #0]
```

   Between ADD/LDUR, the value of X5 is forwarded. From the LDUR to ORR, the value of X3 and X5 are forwarded. And then value of X3 and X5 are forwarded from the ORR to STUR stages.

5. If there is no forwarding, what new input and output signals do we need for the hazard detection unit in Figure 4.59. Using this instruction sequence as an example, explain why each signal is needed.

   We can make the hazard unit put out a stall signal, or an instruction sequence reorder signal.

6. For the new hazard detection unit from problem 5, specify which output signals it asserts in each of the first five cycles during the execution of this code.

   Between ADD/LDUR, LDUR to ORR, and ORR to STUR, it would need to send out a stall signal.

Question #3

The pipeline registers hold data and control between pipeline stages. The first register (IF/ID) stores the instruction and the program counter (PC). List the contents about each of the other registers. Do not list the control signals.

- IF/ID: Holds the next PC, and fetched instructions.
- ID/EX: Holds the next PC, and read data from registers 1-3, and immediate values.
- EX/MEM: Holds the next PC/Branch location, and ALU output.
- MEM/WB: Holds address of read memory and write location.

Question #4

Write the program for the last question in Test 1

```
main:   SUBI    SP, SP, #-40
        ADD     X0, X10, XZR
        BL      init_elements
        LDUR    X0, [X10, #0]
        LDUR    X1, [X10, #4]
        LDUR    X2, [X10, #8]
        LDUR    X3, [X10, #12]
        BL      calc_shape
        ADDI    X19, XZR, #40
        LDUR    X20, [X10, #20]
        STUR    X0, [X20]
        BL      print_number
        ADDI    SP, SP, #40
```