Full Name:____________________________________

15-740/18-740, Fall 2018

Exam 1

September 26, 2018, 3:00pm-4:20pm

Instructions:

- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer. A few pages of scratch paper are provided at the end of the text booklet, but your final answer should be written in the space provided.
- Show your work and discuss your answer. You will be graded more on your explanation than on your final answer.
- The exam has a maximum score of 80 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is CLOSED BOOK, CLOSED NOTES. You may use a calculator, but no networked devices (e.g., phones, laptops, etc.).
- The exam introduces techniques and key ideas from research papers and asks you analyze them. *Confine your answers to the ideas that are presented in the exam.* We are not looking for answers that involve other techniques in these papers which we do not present.

Do not write below this line

Problem	Your Score	Possible Points
1		50
2		30
Total		80

WaveScalar

Problem 1. (50 points total):

This problem asks you to explore the design of data-flow machines and answer questions about the performance characteristics of data-flow designs.
Data-flow architecture is an alternative to the well-known von Neumann architecture in which a program counter represents the current point of execution and drives the rest of the architecture. In data-flow machines, however, no program counter is present. Instructions are "fired" whenever their operands are ready. The binary executable of a data-flow machine stores a directed graph. Nodes in the graph are instructions, and arcs between nodes are data dependencies between instructions.
In the following questions we simplify some aspects of a real data-flow machine. All questions are based on what we have covered during the lectures, and no prior background or expertise of working on data-flow projects is assumed.

A. What is the potential advantage and disadvantage of data-flow machines compared with von Neumann architecture? Give one for each.

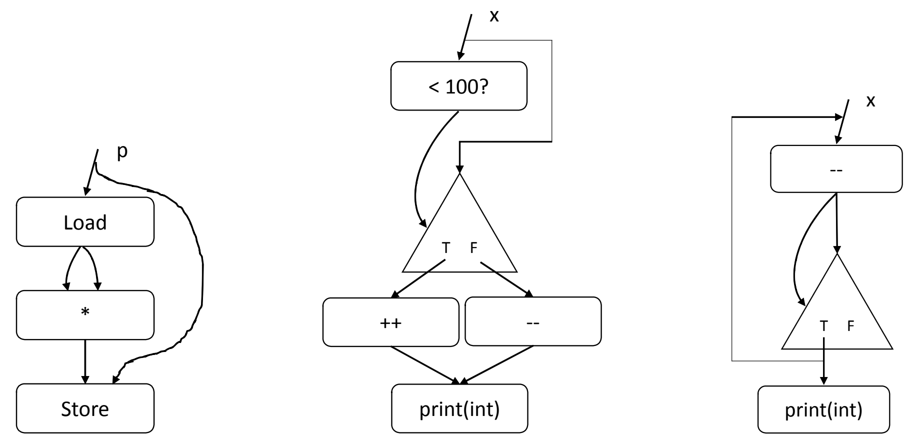
   5 points

   Advantage: Higher degree of parallelism

   Disadvantage: Hard to write programs

   (Other reasonable answers are also accepted)

   I did not give full mark for the answer "hard to debug", because modern superscalar is a limited form of data flow machines, and as we all know, debugging programs on superscalar is not that difficult



(a) `*p = (*p) * (*p)` (b) `print(x < 100 ? ++x : --x)` (c) `while(--x) print(x)`

Figure 1: Simple Dataflow Examples

The architecture this problem focuses on is called **WaveScalar**. It is a dynamic data-flow architecture that has the following features:

(a) Programs are executed in the unit of **waves**. A wave is a directed acylic subgraph of the program's data-flow graph. The compiler is responsible for dividing a program into waves. In this question, consider both the loop body and if-else branch as waves.

(b) Most instructions take two operands, and produces one output. Exceptions are `Load`, `Add/Sub/Compare immediate` which only take one operand, and `Store` instruction which does not produce any output. Fig. 1a shows a simple data flow graph that uses `Load`, `Store` and arithmetic.

(c) To support branches, a special "switch" node is added to the ISA. The triangular "switch" node takes two inputs: a predicate and a data item. It has two outputs. One of these two outputs would be active according to the value of the predicate input (T/F or non-zero/zero). Values would be discarded if an output is not connected to any other node. An example of switch node is given in Fig. 1b.

(d) To support loops WaveScalar tags each value with an associated wave number, and instructions only fire when the input values are ready and input wave numbers match. In this question, you do not need to worry about instructions that maintain wave numbers. Just assume the architecture has some way of knowing which data item belongs to which iteration of the loop. An example of loops is given in Fig. 1c.

(e) Instructions are loaded into a grid of execution units, called a **WaveCache**. Similar to the instruction cache in von Neumann architecture, WaveCache may suffer cache misses if an instruction is ready to fire, but there are no free execution units. In this (relatively rare) case, one instruction has to be evicted to make a free unit.

B. Data-flow machines require special compilers to transform sequential assembly code into the data-flow representation. Given the source code of a program and its assembly output from a RISC compiler in Fig. 2, draw the dataflow graph for this code on the next page using provided nodes in Fig. 3.

(Hint: You do not need more than ten nodes)

20 points

```
void s(char in[10], char out[10]) {
    int i = 0;
    do {
        char t = in[i];
        if(t) {
            out[i] = t;
        }
        i++;
    } while(i < 10);
    // no more uses of i
    // no more uses of in
}
```

```
; r1 = i
; r2 = in
; r3 = out
; r4 = t
loop:
add r6, r2, r1
ld r4, r6(0)
beq r4, #0, L1
add r6, r3, r1
st r4, r6(0)
L1:
addi r1, r1, #1
blt r1, #10, loop
```

Figure 2: Assembly of a Simple Program

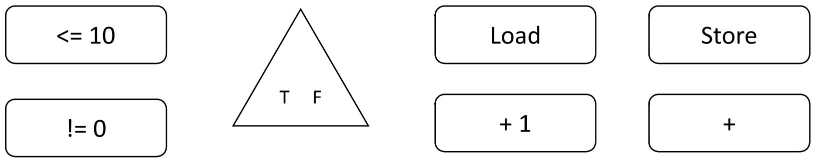
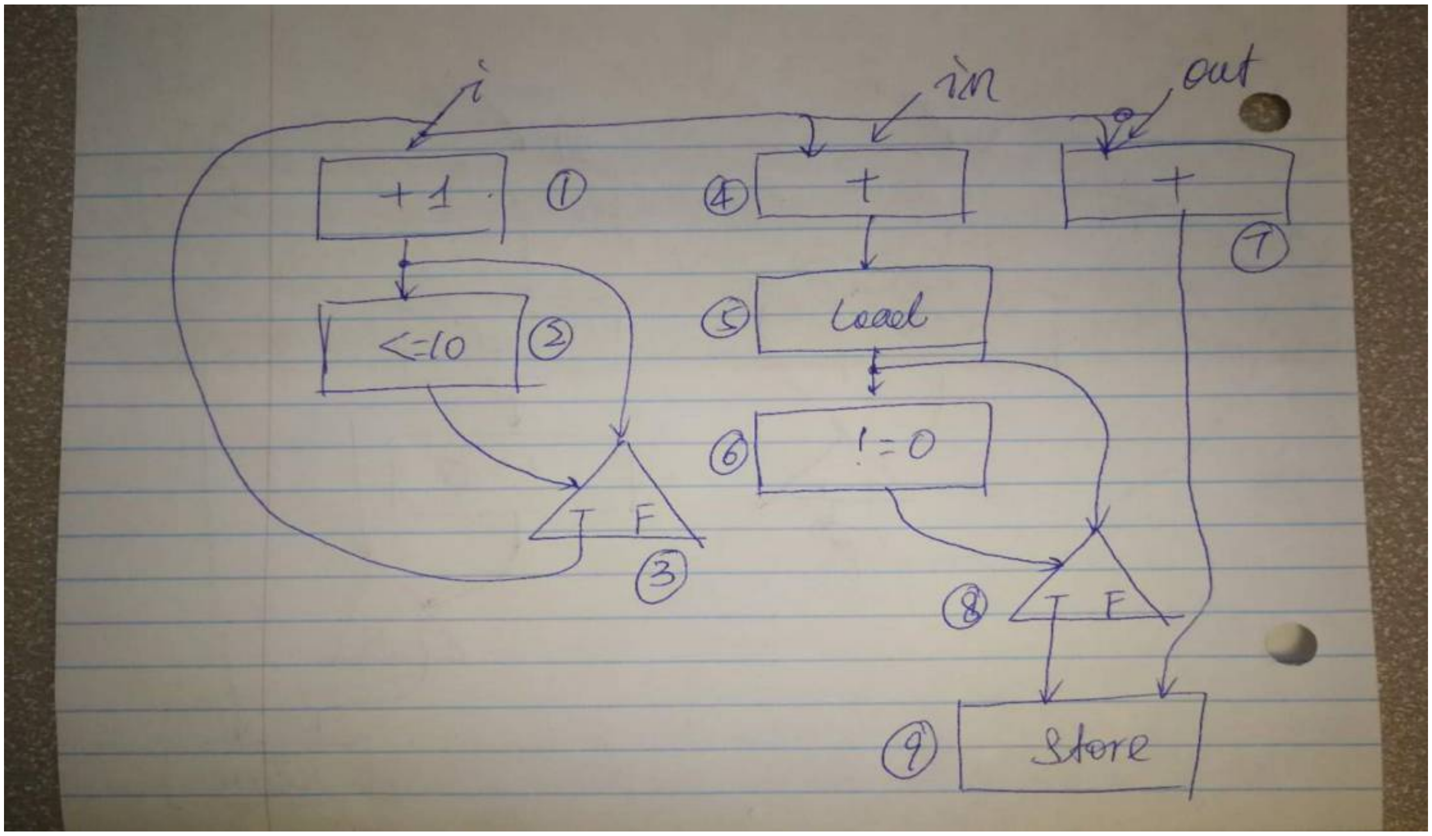


Figure 3: Available execution units for dataflow graph

$\swarrow$ i (= 0 at init) $\swarrow$ in $\swarrow$ out



C. Assume that all instructions require one execution unit, take one cycle to execute including Load/Store, and that there is an infinitly large WaveCache. Also assume instructions are executed as soon as their operands are ready. Label the nodes "A", "B", "C", ..., in the previous page in arbitrary order as you like. Starting with cycle zero, fill in Table 1 with labels of nodes that are active at corresponding cycles. You only need to do this for cycles 0 – 9 (first 10 cycles).

10 points

Cycle	Active Nodes
0	1 4 7
1	2 5
2	3 6
3	1 4 7 8
4	2 5 9
5	3 6
6	1 4 7 8
7	2 5 9
8	3 6
9	1 4 7 8

Table 1: Active Nodes on Each Cycle

What is the maximum instruction-level parallelism in this program?

4 (the maximum number of nodes that are active at the same cycle in the table)

D. Now assume that in reality, a WaveCache can only hold four instructions. Assume that WaveCache is fully-associative, and uses LRU eviction strategy. How many WaveCache misses would you observe during the first ten cycles in the previous execution? Write down assumptions you make, if any.

5 points

All instructions will miss the cache, because the reuse distance of instructions are all greater than the size of the cache

E. Instead of always executing the next available instruction, we split the dataflow graph into smaller waves that fit in WaveCache, and execute them one at a time. First, only loop indices `i` for all iterations are computed. Then in later phases we compute the rest of the waves.

You can assume that WaveScalar has an infinitely large, zero-latency token store to buffer intermediate data values. Ignore them for this question.

(a) Using the node labels, show how you would split data flow graph into smaller waves following the above description. (b) What effect does this have on cache locality? (c) Does overall parallelism become better or worse?

5 points

(a) We split the data flow graph into two parts: 1, 2, 3 and 4, 5, 6, 7, 8, 9

(b) Locality improves because the instruction cache will not miss on every instruction

(c) Overall parallelism becomes slightly worse

(d) This approach is called "Hierarchical Dataflow". Based on your answers for (a) to (c) above, and topics discussed in lectures, how can hierarchical dataflow help deal with some of the problems of classical dataflow?

5 points

In classical data flow, the degree of parallelism can become a problem as the instruction locality decreases.

Using hierarchical data flow it is possible that we maintain the parallelism of data flow designs while maintaining better locality

Cache Coherence with MOESI

Problem 2. (30 points total):

In the lecture we have seen a version of snoopy cache coherence protocols, MESI. In this problem, we explore further refinement of the snoopy MESI protocol by adding a new state, the "**O**wned" State.

A. One of the main advantages of MOESI over MESI is its ability to support "dirty sharing". Recall that in MESI protocol, a "**M**odified" cache line must be flushed and transit to "**S**hared" state on a non-exclusive bus read. MOESI optimizes this by adding an "**O**wned" state, which reflects the fact that a cache line might be shared somewhere else, but the content of the line is inconsistent with main memory. Accordingly, a bus read event will cause a "M" state line to transit into "O" state. Given a MOESI state machine in Fig. 4, add state transitions and events to support the "O" state. You can assume an invalidation-based MOESI protocol. Note that you do not have to complete the entire MOESI state machine. Just give **all** state transitions in and out the "O" state. State transition events are shown in Table 2.

15 points

(1) There should be one edge from M to O, BusRead/Transfer this is where the O state is entered
(2) There should be one edge from O to I, BusReadX/Transfer this is when other processors are to modify the line, and the Owned state is responsible for providing the content
(3) There should be one edge from O to M, PrWrite/--
(4) There should be one edge from O to itself, PrRead/-- and PrWrite/--
(5) There should be one edge from O to I, PrEvict/Flush

Note: Many students missed (4) and (5)

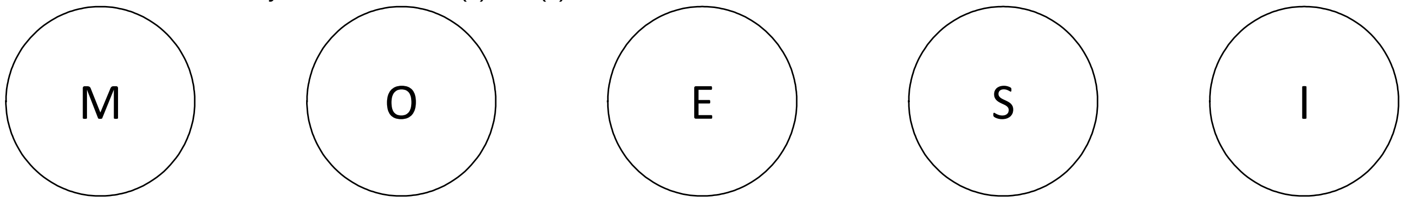


Figure 4: MOESI

Events and Actions						
PrRead	PrWrite	BusRead	BusReadX	PrEvict	Flush	Transfer

Table 2: Coherence Events and Actions

B. Does MOESI reduce the amount of bus traffic? Why or Why not? Make a case-by-case discussion if necessary. **5 points**

No. The total number of traffic remains the same. MOESI just postpones the write back of dirty cache lines when they are requested

C. In previous questions, we assumed that MOESI uses invalidation. In practice, MOESI can be implemented using another technique as an alternative to invalidation. What is this technique called? How would this change the state machine for transitions in and out the "O" state? Explain in plain text below.

5 points

Update-based coherence protocol.

When the O state cache line is modified by the processor, the changes that have been made must also be broadcasted on the bus. Other processors having a S state line on the same address must apply the change to their own cache lines.

D. Can you identify a hardware configuration where MOESI has no obvious advantage over MESI? Hint: Think of the speed of hardware components and/or interconnections.

5 points

When the speed between caches are comparable to the speed between caches and the lower level storage (DRAM or larger cache)

Scratch paper

Scratch paper

Scratch paper