Name: ______________________________

Practice Final

Revised December 9, 2020

None of these problems will be due for credit. This is more of a review check-list than an actual practice final; however there are a few practice problems.

1. When we first looked at instruction sets, we listed four design principles:

   (a) regularity promotes simplicity,

   (b) smaller is faster.

   (c) good designs require good compromises,

   (d) make the common case fast

   How do these design principles apply to each of the following:

   - The MIPS instruction set
   - Pipelining
   - The memory hierarchy

2. What aspects of the above seem to violate the four principles?

3. Why do CPUs have caches?

4. Main memory is built using a technology called *Dynamic RAM* or DRAM. Caches are built using a technology called *Static RAM* or SRAM.

   (a) What is the main difference between the designs of DRAM and SRAM?

   (b) Why do these differences make DRAM much cheaper but much slower?

5. List two reasons that L1 caches are typically only a few kilobyes.

6. Show how 32-bit addresses are divided into tag, index, and offset given the following cache descriptions:

   (a) 32KB, byte addressable, 8-way set associative cache with 4 byte blocks.

   (b) 16KB, byte addressable, direct mapped cache with 8 byte blocks.

   (c) 64KB fully associative cache with 64 byte blocks

7. Calculate total size and overhead for each of the above caches.

Name: ______________________________

8. You have an 128KB, byte addressable, 2-way set associative cache with 16 byte blocks. This cache uses an LRU replacement policy. For the following sequence of addresses, show whether each is a hit or a miss. For each cache miss, tell which bytes were replaced.

   Extra Credit: *Write your own cache replacement problem (a problem similar to this one with a configuration and different sequence of memory accesses) and post it to Piazza.*

Address	Tag	Index	H / M	Tags Replaced	Notes
0x12341110					Cold
0x12341113					Same block
0x12351113					Cold
0x12341113					
0x12361114					
0x12351113					
0x12361114					
0x12341110					
0x12361110					Same block
0x12351113					

9. Construct a repeating sequence of memory references for which a Random cache replacement policy has a lower miss rate than LRU. Clearly explain *why* Random does better in this case.

10. Explain the difference between spatial locality and temporal locality.

11. How does a program's temporal locality affect its cache hit rate?

12. How does a program's spatial locality affect its cache hit rate?

13. Explain the difference between a write-through and a write-back cache.

14. List the advantages and disadvantages of a split cache over a unified cache.

15. Assume that variables A, B, C, D, and E all map to the same line in a 4-way set-associative cache that uses the pseudo-LRU replacement policy. Furthermore, assume they are accessed in this order: A B C A D E. Which variable is evicted to make room for E? Clearly show the state of the pseudo-LRU algorithm before E is requested and explain how that state determines which block is evicted.

Name: ______________________________

16. Consider a non-pipelined, single-cycle CPU (i.e., all instructions complete in one CPU cycle). Suppose you are asked to design a similar CPU with a two-stage pipeline. List the factors that may prevent you from making the cycle time of the new pipelined machine half of the cycle time of the non-pipelined machine.

17. Consider two CPUs that differ only in that one is pipelined and one is not. Is the total time needed to completely execute a single instruction greater on the pipelined or non-pipelined machine? Why? (In other-words, does pipelining tend to increase or decrease the latency of individual instructions?)

18. Look at the two tables below.

    (a) Assume that your processor is a 5-state pipeline with both data and register forwarding. Which table is correct? Why?

    (b) Draw a diagram showing how the instructions would flow through the pipeline if it didn't have any data or register forwarding.

	1	2	3	4	5	6	7	8	9	10
LD R1 0(R2)	F	D	E	M	W					
ADD R3, R4, R1		F	D		E	M	W			
ADD R6, R3, R4			F		D	E	M	W		

	1	2	3	4	5	6	7	8	9	10
LD R1 0(R2)	F	D	E	M	W					
ADD R3, R4, R1		F	D		E	M	W			
ADD R6, R3, R4			F	D		E	M	W		

19. What are structural hazards? What causes them? How are they resolved?

20. Are structural hazards unique to pipelined CPUs? (As opposed to non-pipelined CPUS?) Why or why not?

21. What are data hazards? What causes them? How are they resolved?

Name: ______________________________

22. Are data hazards unique to pipelined CPUs? (As opposed to non-pipelined CPUs?) Why or why not?

23. What are control hazards? What causes them? How are they resolved?

24. Are control hazards unique to pipelined CPUs? (As opposed to non-pipelined CPUs?) Why or why not?

25. (Problem 6.4 from Patterson and Hennessy): Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved by forwarding? Which dependencies are data hazards that will cause a stall?

```
1: add $3, $4, $2
2: sub $5, $3, $1
3: lw  $6, 200($3)
4: add $7, $3, $6
```

26. Clearly explain why there must be a stall between instructions 3 and 4 in the example above (and why this stall can't be removed with a forwarding path).

27. Explain the purpose of each component and control wire in the figure below:

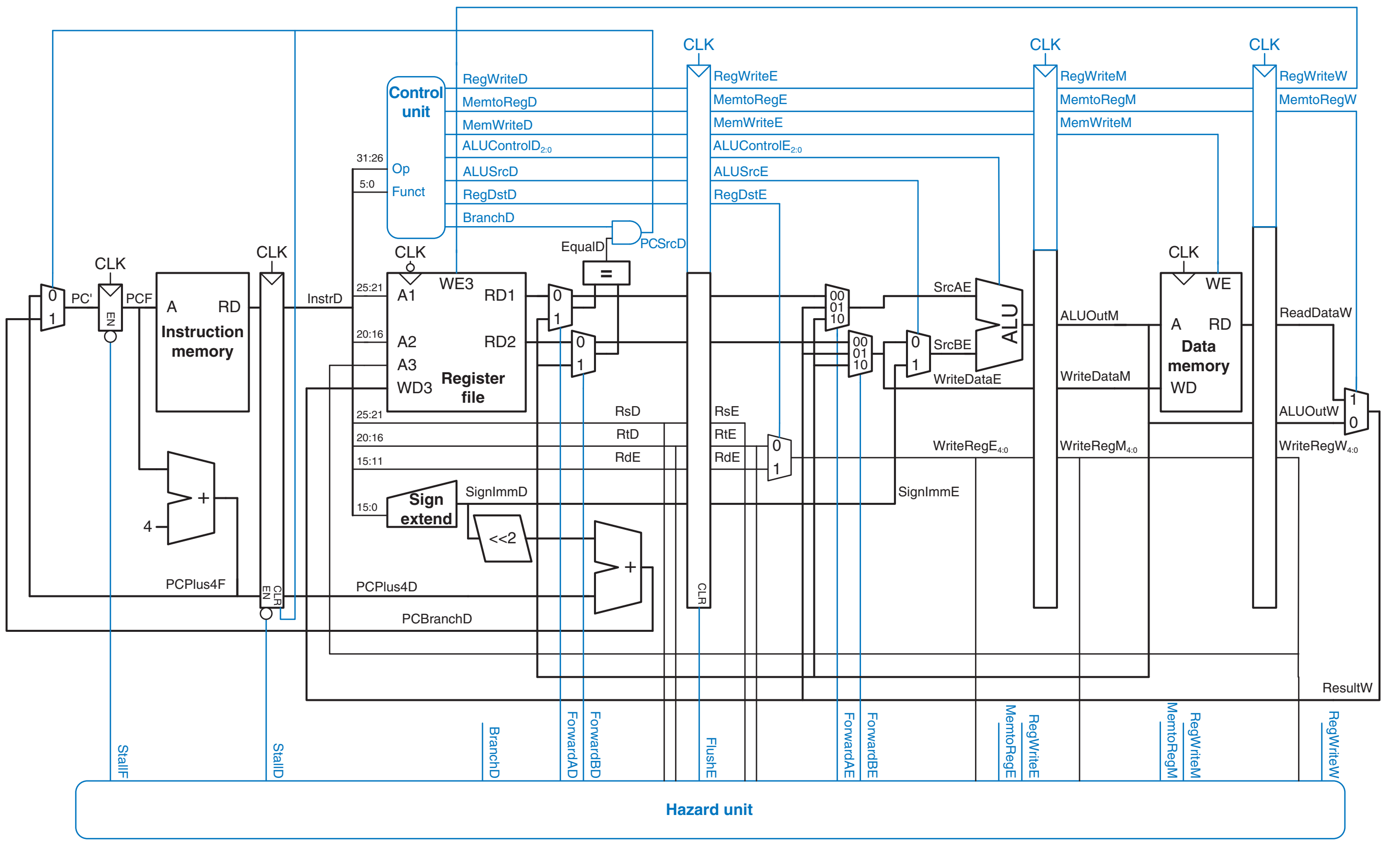
28. Highlight the forwarding paths used by the following sequence of instructions:

```
add $t0, $t1, $t2
sub $t3, $t4, $t5
add $t5, $t0, $t3
```

29. Highlight the forwarding paths used by the following sequence of instructions:

```
add $t0, $t1, $t2
sub $t3, $t4, $t5
beq $t0, $t3, END
```

Name: ______________________________



CLK
CLK
CLK
RegWriteD
MemtoRegD
MemWriteD
ALUControlD$_{2:0}$
ALUSrcD
RegDstD
BranchD
RegWriteE
MemtoRegE
MemWriteE
ALUControlE$_{2:0}$
ALUSrcE
RegDstE
RegWriteM
MemtoRegM
MemWriteM
RegWriteW
MemtoRegW
Control unit
31:26
5:0
Op
Funct
EqualD
PCSrcD
=
CLK
PC'
PCF
EN
A
RD
Instruction memory
CLK
InstrD
CLK
25:21
WE3
A1
RD1
20:16
A2
RD2
A3
WD3
Register file
0
1
0
1
SrcAE
ALU
SrcBE
WriteDataE
00
01
10
ALUOutM
WriteDataM
CLK
WE
A
RD
Data memory
WD
ReadDataW
ALUOutW
WriteRegW$_{4:0}$
25:21
RsD
RsE
20:16
RtD
RtE
15:11
RdE
RdE
WriteRegE$_{4:0}$
WriteRegM$_{4:0}$
15:0
Sign extend
SignImmD
SignImmE
<<2
+
4
PCPlus4F
PCPlus4D
PCBranchD
CLR
EN
CLR
ResultW
StallF
StallD
BranchD
ForwardAD
ForwardBD
FlushE
ForwardAE
ForwardBE
MemtoRegE
RegWriteE
MemtoRegM
RegWriteM
RegWriteW
Hazard unit

Name: ______________________________

30. Consider the assembly code fragment below:

```
sw $16, 12($6)
lw $16, 8($6)
beq $5, $4 LABEL
add $5, $1, $4
sub $5, $15, $4
```

Suppose we modify the pipeline so that it has only one memory (that handles both instructions and data). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

(a) Draw a pipeline diagram to show were the code above will stall.

(b) In general, is it possible to reduce the number of stalls/`nop`s resulting from this structural hazard by reordering code?

(c) Must this structural hazard be handled in hardware? We have seen that data hazards can be eliminated by adding `nop`s to the code. Can you do the same with this structural hazard? If so, explain how. If not, explain why not.

(d) Approximately how many stalls would you expect this structural hazard to generate in a typical program? (Assume 25% of instructions are loads and 10% are stores.)

31. Implement the following recursive code in assembly. Use standard calling and register-use conventions. You may assume `moveDisc` has already been written. (In other words, just call it, don't try to write it.)

```
public static int solveHanoi(int numDiscs, int start, int end, int swap) {
  if (numDiscs == 1) {
    moveDisc(start, end);
    return 1;
  }
  int left = solveHanoi(numDiscs-1, start, swap, end);
  moveDisc(start, end);
  int right = solveHanoi(numDiscs-1, swap, end, start);
  return left + 1 + right;
}
```

32. Implement the following function in assembly. Use standard calling and register-use conventions. You may assume `f1` and `f2` have already been written. (In other words, just call them, don't try to write them.)

```
public static int composition(int a, int b, int c) {
  return f1(2 + f2(a, b), f2(b, c), a);
}
```