ECE411 Computer Organization and Design
Fall 2019 Final Exam
12/18/2019

NetID:_____

First Name:_____

Last Name:_____

Exam Guidelines:

1. This exam has **5 problems.** Make sure you have a complete exam before you begin **[22 pages + the cover page]**.
2. Write your name on every page in case pages become separated during grading.
3. You will have **3 hours** to complete this exam.
4. Write all of your answers on the exam itself. If you need more space to answer a given problem, continue on the back of the page, but clearly indicate that you have done so.
5. This exam is closed-book. You may use two sheets of notes. You may use a calculator.
6. **DO NOT** do anything that might be perceived as cheating. The minimum penalty will be a grade of zero.
7. Show all of your work on all problems. Correct answers that do not include work demonstrating how they were generated may not receive full credit, and answers that show no work cannot receive partial credit.
8. The exam is meant to test your understanding. Ample time has been provided. So be patient and read the questions/problems carefully before you answer.
9. **Good luck!**

Problem	Points
1. **12** pt	
2. **14** pt	
3. **15** pt	
4. **10** pt	
5. **12.5** pt (+0.5EC)	
Total: **63.5** pt (+0.5EC)	

NetID:_____

Question 1: Coherence

Consider a multicore system with a snoop bus topology. The system implements the MSI protocol with write-invalidates. The "magic memory" MSI protocol is given in the table below (Table entries contain "next-state/message-sent" tuples).

State \ Input	Load	Store	OtherGetX	OtherGetS
M	-/-	-/-	I/-	S/-
S	-/-	M/OwnGetX	I/-	-/-
I	S/OwnGetS	M/OwnGetX	-/-	-/-

Consider that memory access latencies are non-deterministic and that each core may have more than one outstanding request to memory. In the basic MSI protocol, this creates issues! If, upon sending an OwnGetX message, the message sending core moves immediately to 'M' state, then, upon receiving an OtherGetS message before receiving the relevant data, the core will send invalid data to both the other core and the shared cache. Clearly, the 'three-state' MSI protocol must have more than three states!

For a snoop-bus system, the additional states may be:

IS-Intermediate (IS) --- Wait for data on snoop bus before transitioning to S
SM-Intermediate (SM) --- Wait for data on snoop bus before transitioning to M
IM-Intermediate (IM) --- Wait for data on snoop bus before transitioning to M

Following additional Messages/Events are also required:

Snoop Data --- Cache line on snoop bus (may trigger state change)

a. (6 points) Fill in the following table to implement the MSI protocol without "magic memory", ensuring to keep data coherent even with non-zero memory access latencies:

State \ Input	Load	Store	OtherGetX	OtherGetS	Snoop Data
M					
S					
I					
SM					
IS					
IM					

A snoop-bus is often thought of as a single bus with multiple drivers and multiple listeners. This is, of course, difficult to actually fabricate. Instead, we consider the properties of a network which allow snoop-like behavior. A snoop-bus is a network with the following properties:

- For all nodes *a, b* if message *m* is delivered to node *a*, it is also delivered to node *b* (a snoop-bus is a broadcast network).
- For all nodes *a,* and *b,* and messages *m*, and *n*, if *m* is delivered to *a* before *n*, then *m* is delivered to *b* before *n* (a snoop-bus enforces total ordering).
- For all nodes a, b, and messages *m, n* if *m* is delivered to *a* before *n*, then *m* is delivered to *b* before *n* is delivered to *a*.

b. (6 points) Describe a system (using English and a block diagram or RTL pseudo-code) which implements a snoop-bus (meaning it has the three properties of a snoop-bus) but does not have any wires driven by multiple sources (i.e. tristate buffers are not allowed). Use no more space than what is available on the remainder of this page.

NetID:_____

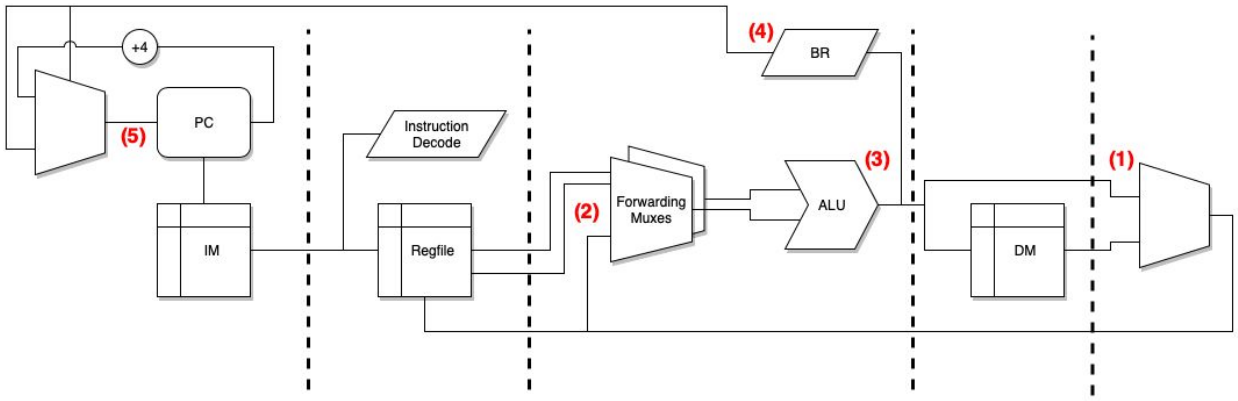
NetID:_____

Question 2: Pipelining and Branch Prediction

Part 1: Pipeline analysis (8 points)

Thanks to the skills you have acquired in 411, you have just been hired by Ra-cache Semiconductors to perform design validation on a new pipelined CPU. Your first task in your new job is to analyze the performance of this new design. The figure below shows a simplified view of a 5 stage pipeline implementation. The BR module is responsible for branch resolution logic and providing information to the PC mux.

Note: You are permitted to answer the following questions algebraically for full credit. Answers which show more work will have a better chance of earning partial credit.



The combinational and path delays are as follows:

- (1) to (2) takes 5ns
- (2) to (3) takes 6ns
- (3) to (4) takes 4ns
- (4) to (5) takes 4ns
- Register clock-to-out delay is 1ns
- You may assume all other delays are insignificant

a. (2 points) To measure the performance quantitatively, you use a benchmarking suite that consists of 50 million dynamic instructions, 5% of which are branches. The designers have assured you that the branch prediction is 95% accurate. What is the total program execution time, assuming no causes of stall other than control hazards?

While you were looking over the company's design at the local Pete's Coffee shop, Professor Lumetta walked by and noticed you needed help, and decided to return the favor since you have helped him so much in the past. He took a glance at the profile of the target application your processor would be running (from part A), and had an idea. Professor Lumetta thinks that you might benefit from delaying branch prediction to the MEM stage, and changing the **BR** module location to instead take input of the ALU outputs once they have latched into the MEM stage.

b. (2 points) Why might Professor Lumetta be correct? Name one key benefit of this change, and one key drawback. (Hint: think about your answers to part 1)

c. (2 points) You figure that Professor Lumetta is probably on to something, and so consider the design with his changes. What is the new maximum operating frequency, F_2, of the pipeline with Professor Lumetta's change?

d. (2 points) Again looking to evaluate the design quantitatively, you use a benchmarking suite that consists of 50 million dynamic instructions, 5% of which are branches. The designers have reassured you of an expected 95% branch predictor accuracy. What is the total program execution time, assuming no causes for stall other than control hazards?

NetID:_____

Part 2: Branch predictor (6 points)

Research has shown that very far apart branches (i.e. 100's of branches apart) are often correlated. To capture this, branch predictors such as TAGE (partially TAgged GEometric sequence) and LTAGE (Loop TAGE) frequently have global history registers of hundreds of bits representing hundreds of branches. Since the register is so large, it is often implemented as a circular buffer with pointer(s), rather than as a shift register. This large global history register must be updated speculatively, thus it is imperative that, upon detection of mispeculation, the global history register can be reset to the condition it was in prior to the misspeculated branch prediction.

a. (2 points) Consider such a global history register which supports a branch predictor that uses 512 bits of global history in a processor which supports up to 6 in flight branch speculations. What is the size (in bits) of the global history register alone (i.e. do not consider the size of pointer registers or other metadata)? **Why?**

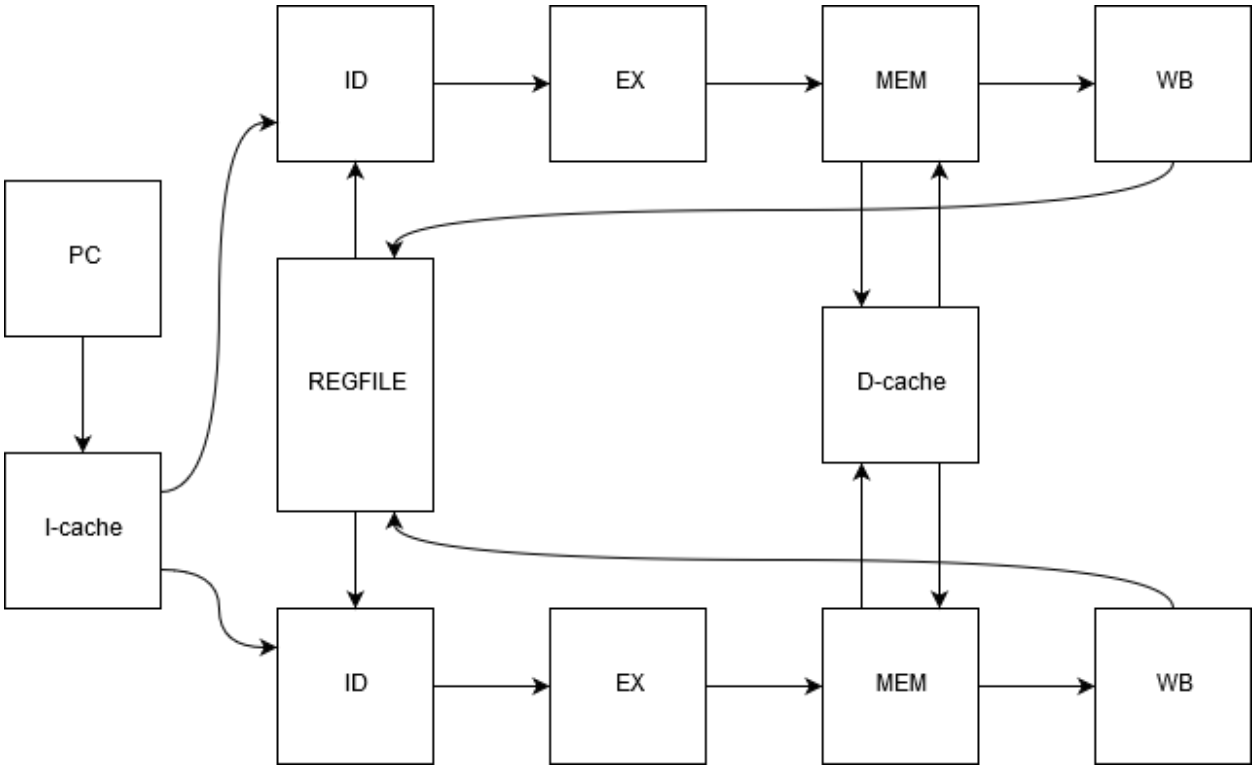
b. (2 points) Consider a scenario in which there are 5 in-flight branches. The third of the five branches is the first to be resolved and determined to be misspeculated. In no more than 3 sentences, describe how a branch predictor would use a single tail (insertion) pointer to recover from its mispeculated state. Ensure to describe any other metadata which the branch predictor must contain to enable this recovery or, if it does not require any other metadata, explain why.

c. (2 points) With your answer from parts (a) and (b), determine the total size of the global history register (including the pointer register, and any other needed metadata).

Question 3: MP

Welcome to Leg Holdings, the newest money-losing venture of the VC-backed Lack of Vision fund. You were hired to start after you graduated, and you've eagerly been looking forward to your start date. However, in the short time between your interview and start date, everyone you talked to during the interview was fired. After the disastrous We Don't Work failed IPO, the Lack of Vision fund is demanding all of its companies to start turning a profit. Since you (a new hire) are the cheapest employee, you survived the layoffs, but everyone more senior to you is gone.

The project you were hired to work on is a superscalar RISC-V processor. The current specifications mention a dual-issue, dual-commit, in-order design. For completely arbitrary reasons, you have decided to implement two physical pipelines that are very similar to your basic MP3CP3 design. There will be two different ID, EX, MEM, and WB stages, however the design shares a program counter, register file (no SMT), I-cache, and D-cache. For simplicity, you will always fetch PC and PC+4 (8 bytes of data). Upon fetching the instruction, the shared PC is incremented by 8. Once the instructions are fetched, they always stay in sync with each other (both in ID, then both in EX, etc). If, for whatever reason, one pipeline issues a stall, the other pipeline will stall in the same cycle. This diagram describes the design you will be implementing.

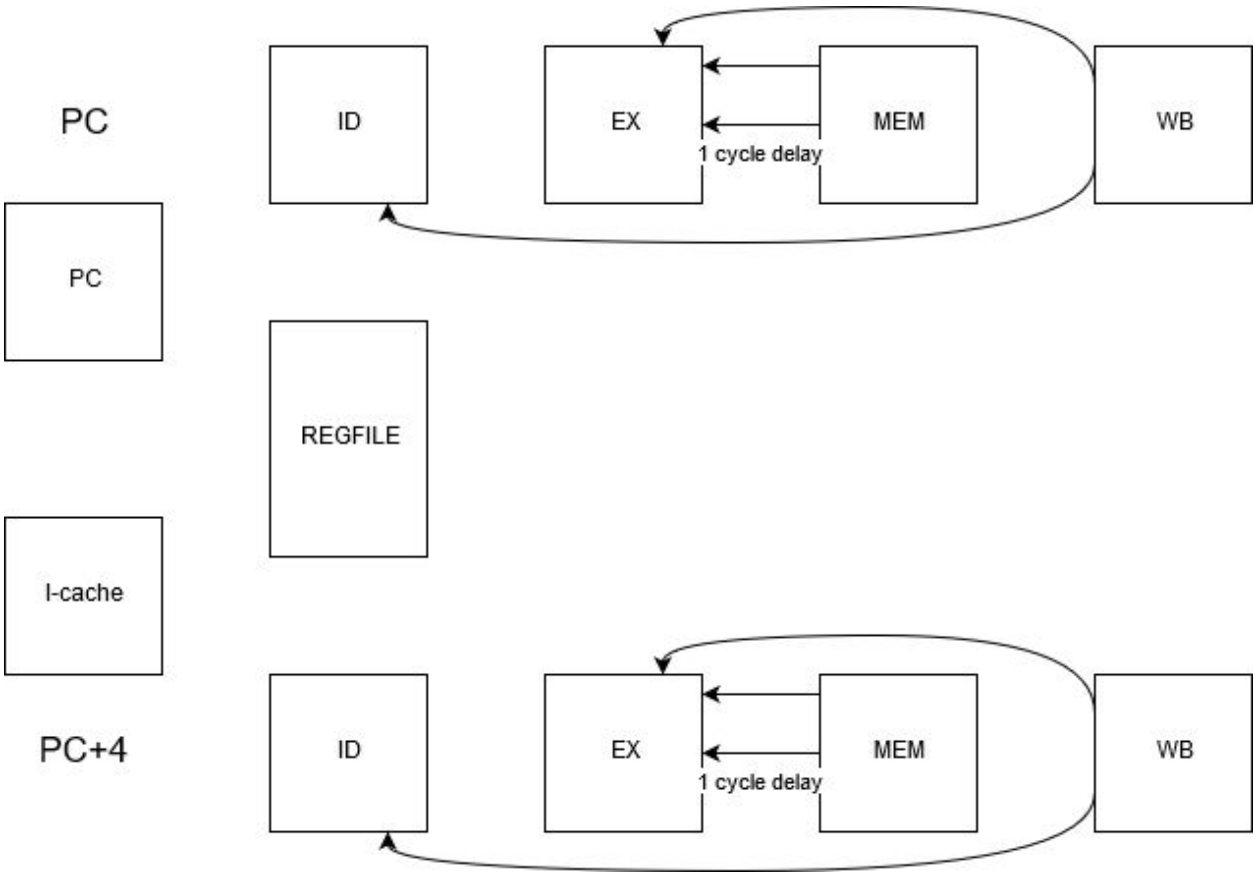


a. (3 points) The first design choice you want to tackle is how to issue two instructions at once. You, being the efficient engineer that you are, decide to modify your MP3 I-cache. Your compiler guarantees that all instructions are 4-byte aligned, and branches and jumps are either 8-byte aligned or followed by a NOP, so you don't need to consider control hazards at this time. What changes do you make to your I-cache and/or pipelines to support fetching PC and PC+4?

   (Note, you do not need to use all the space available below.)

b. (3 points) Now that you have two instructions going down the pipeline at the same time, you've started to run into some hazards, and you don't have proper handling for them. To help with resolving these issues, you decide to draw out a diagram of all of the possible forwarding and delay paths. Since you already have a working diagram from MP3 CP3, you just need to show how the two pipelines interact with each other.

Modify the below diagram to add in all of the required forwarding and delay paths. Draw a plain arrow for a forwarding path, and draw an arrow and label it with a delay to indicate a RAW dependency that requires a certain delay. Insert the minimum number of paths possible. To show a forwarding path from the MEM/WB register to EX, draw an arrow from the WB block to the EX block, as is done below.



c. (4 points) It turns out you weren't left completely on your own, and to help give you a leg up over your competition, the fired hiring manager left you a short piece of code with the only note being "Get this working". So you try it on your MP3 design, and it works fine. But when you try running it on your superscalar design, it breaks down. You decide to actually look at the code a little closer, and find the following snippet:

```
unsigned int *a = (unsigned int *)0x40000;
unsigned int *b = (unsigned int *)0x46000;
unsigned int *c = (unsigned int *)0x4C000;

for (int x = 0; x < 255; x++) {
    unsigned int first = a[x] ^ b[x];
    unsigned int second = a[x+1] ^ b[x+1];
    c[x] = first + second;
}
-----------------------------------------------------------------
lui x10, 0x40
lui x11, 0x46
lui x12, 0x4C

addi x5, x0, 256
loop:
    lw x6, 0(x10)
    lw x7, 0(x11)
    lw x8, 4(x10)
    lw x9, 4(x11)
    xor x6, x6, x7
    xor x8, x8, x9
    add x6, x6, x8
    sw x6, 0(x12)
    addi x10, x10, 4
    addi x11, x11, 4
    addi x12, x12, 4
    addi x5, x5, -1
    bgez x5, loop
    nop
```

You think this is happening because of a structural hazard, since you only have a single d-cache, but two instructions are attempting to access it at once. In order to check your assumption, you decide to rewrite the software to avoid the structural hazard. How might you rewrite the assembly code to avoid the structural hazard? (You don't need to rewrite the code, just a description of what you might do would suffice)

d. (2 points) After doing some research, you come across a possible solution to your structural hazard, called banked caches. A banked cache allows for multiple requests to be handled at once, as long as they are directed to separate banks. The two cache banks are designed identically to each other and operate mostly independently of one another, but share the same connection to the next level of the memory hierarchy. You can think of the I-cache/D-cache split in the MP3CP3 design as a type of banked cache.

   For this problem, assume you have a 4-way set associative D-cache, with 32 bytes/cache line and 8 lines/way. You would like to modify it to be banked, with two independent banks, each 2 way set associative. You still have 32 byte cache lines and 8 lines/way. For the above code, which address bit would make the most sense to use to determine which bank to route a request to? (MSB = 32, LSB = 1).

e. (3 points) Unfortunately for you, this project is taking longer than your boss would like. Now, your direct manager (the CEO after all the restructuring before you arrived) is asking for a progress update. You are able to show her some progress with the I-cache and hazards, but she would like information about her favorite test code (so that's why you had to get the above code working). She isn't satisfied with your explanation of why it isn't working, and asks you to show her something that would make it work.

   Sketch out a simple datapath for the banked D-cache. It should show the data array(s), tag array(s), valid array(s), dirty array(s), and LRU array(s). You should be able to tell at a minimum how hits are detected and how reads from the next level cache are handled. The direct-mapped cache discussed during lecture is included in the appendix for your reference. You should use the next page for space.

Question 4: Out of Order Execution

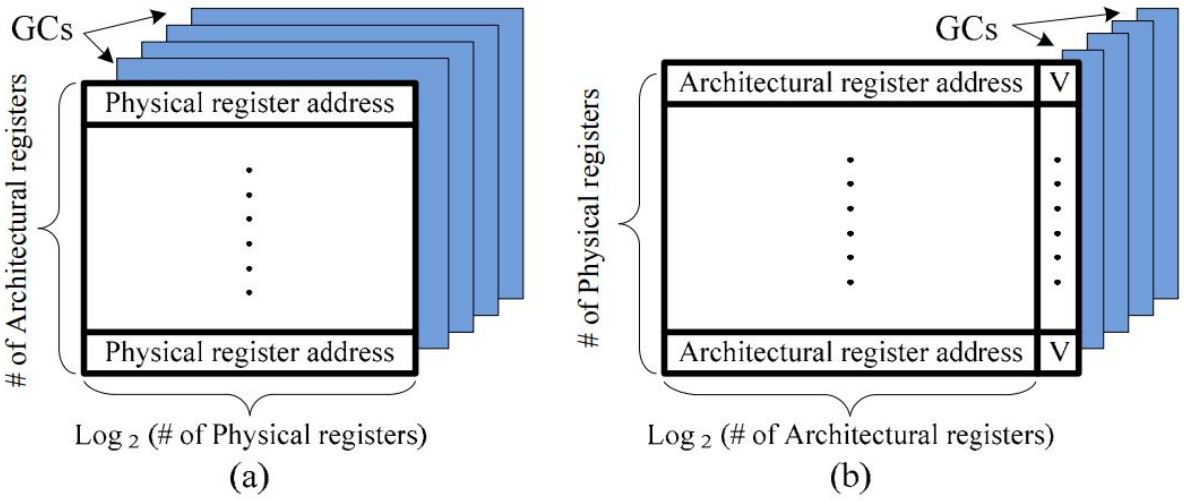


Figure 1. Structure of RAT: (a) SRAM-based, (b) CAM-based.

Consider an out-of-order core with 2^n architectural registers and 2^m physical registers ($m > n$) which implements explicit register renaming. Register Alias Tables (RATs) are commonly implemented with one of two structures: a RAM, or a CAM (content addressable memory).

In a RAM-based RAT, the architectural register number indexes a memory which contains the physical register associated with that architectural register, and its data can be represented in SystemVerilog as:

```
logic [m-1:0] ramrat [1 << n];
```

In a CAM-based RAT, each memory entry contains two fields: 1) an architectural register number, and a valid bit which indicates that the architectural register stored in the entry does indeed correspond with the physical register represented by the entry's address. The CAM-based RAT's data can be represented in SystemVerilog as:

```
logic [n-1:0] areg [1 << m];

logic [1 << m] valid;
```

a. **(2 points)** Let B_{RAM} and B_{CAM} represent the number of memory bits required to implement the RAT using the RAM and CAM structures, respectively. Write expressions for B_{RAM} and B_{CAM} in terms of n and m.

b. **(3 points)** Consider recovery from branch misspeculation in this system. One approach is to traverse the ROB and use metadata stored in it to change the RAT to its prespeculative state. The downside of this approach is that the ROB traversal delay is proportional to the size of the ROB (224 entries in Sky Lake circa 2015). Thus architects use *global checkpointing* to enable faster recovery from branch misspeculation. Using a RAM-based RAT, multiple RAT's are placed in a circular buffer. When branch speculation occurs, the current *working* RAT is copied into an additional RAT, and this copy becomes the *working* RAT. To recover from misspeculation, the prespeculative RAT is made the *working* RAT again.

Write SystemVerilog code which represents the **data structures** (i.e. only the types needed to store the data, no assignments, always blocks, control flow, etc) needed to implement a RAM-based RAT capable of supporting k in flight branches using one global checkpoint per branch. Include necessary metadata such as pointer register(s) and speculative execution bit(s). What is the size (in bits) of these structures as a function of m, n, and k? If the role of a data field is not clear from its identifier, give a brief description of its usage as a single line comment. (Reminder: *$clog2(a)* returns the ceiling of the base-2 logarithm of *a).*

c. **(3 points)** CAM based RATs also support speculative execution. However, rather than duplicating the entire CAM table, we must only duplicate the valid bit vector.

Write SystemVerilog code which represents the **data structures** needed to implement a CAM-based RAT capable of supporting k in flight branches using one global checkpoint per branch. Include necessary metadata such as pointer register(s) and speculative execution bit(s). What is the size (in bits) of these structures as a function of m, n, and k? If the role of a data field is not clear from its identifier, give a brief description of its usage as a single line comment.

d. **(2 points)** Consider a branch predictor which provides both a branch prediction, and a confidence value for that prediction. How could you use this to support more in-flight branches than global checkpoints?

Question 5: Potpourri.

a. Byte-addressable Non-volatile memories (NVM) are gaining more attention in both industry and academia. They are expected to achieve a closely matched performance with DRAM while offering the durability feature (data in NVM can survive a system reboot or crash, like storage devices). In Fig 2.1, NVM is attached to the memory bus, and can only retain data that is written to the NVM before a system crash or reboot. Due to the volatile cache hierarchy including the memory controller's queues, there is additional work required to ensure the data is still consistent inside the NVM in the presence of system failure (crash-consistency) because of two reasons. One of them is the order of write operations to NVM:

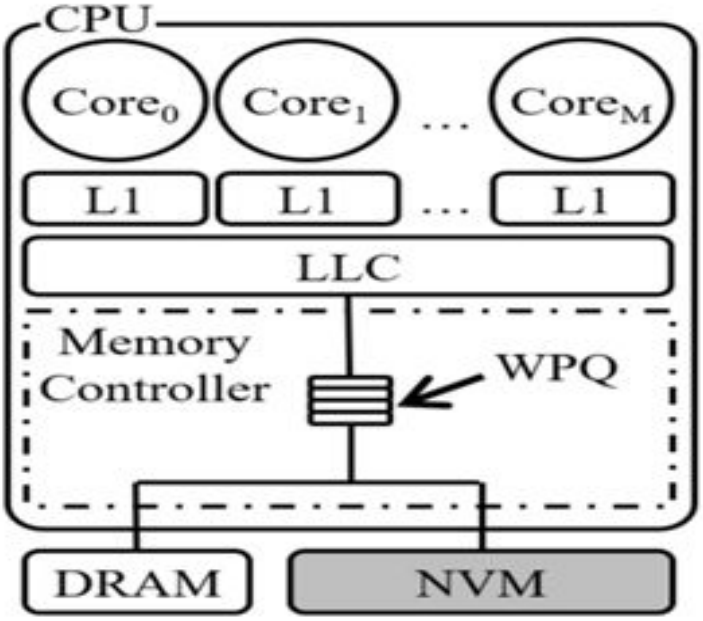


Fig 2.1 Heterogeneous Memory System

i. (2 points) Assume the following code snippet:

```
Entry.data_1 = newData1
Entry.data_2 = newData2
Entry.valid  = true      // data_1 & data_2 are valid to use
```

The object "`Entry`" is mapped to the NVM. To retain "`Entry`" in a consistent state in case of a crash, we need to ensure that "`data_1`", "`data_2`", and "`valid`" have been written to NVM in the correct order. Therefore, we need a way to write them to NVM with the correct order explicitly. About eight years ago, the only available instruction in x86 was "clflush" instruction that allows the programmer to invalidate the cache line and write it back to the write pending queue "WPQ" in the memory controller. WPQ buffers the write requests to the memory subsystem and the memory controller schedules and re-orders those

requests to the memory subsystems. The execution of "clflush" stalls the CPU pipeline until the cache line of the "clflush" instruction is completely written back to the WPQ before resuming the execution of subsequent instructions (serializing memory references). What is the major drawback of using "clflush" to write back data to the NVM?

ii. **(2 points)** As an optimized version of "clflush", the x86 ISA was expanded by "clwb" (cache line write back) instruction to fix the major drawback of "clflush". **Additionally**, "clwb" does not stall the CPU pipeline allowing the write back operation to be overlapped with subsequent instructions. This is an optimization that allows overlapping the write-back operations of both "`data_1`" and "`data_2`" instead of serializing those costly operations. Even with using "clflush" and/or "clwb," it is not enough to ensure writing the data back to NVM in the correct order with the system shown in Fig 1.1. That is the reason behind extending x86 ISA with the "pcommit" instruction. Note that "pcommit" stalls the CPU pipeline until it completes. The following code snippet is the crash-consistency version of the code mentioned in (1.a).

```
Entry.data_1 = newData1
Entry.data_2 = newData2
clwb(&Entry.data_1)
clwb(&Entry.data_2)
pcommit()
Entry.valid = true
clwb(&Entry.valid)
pcommit()
```

Why do you think "clflush" and "clwb" are not enough to enforce the correct order of writing data back to NVM?

b. Jeff Bezels is a business tycoon who made his fortune by maximizing the screen space on his company's smartphones, starting a trend among competitors Pear, Scroogle, and Jonsing. Now, to pursue greater efficiency, he enlists your help in the design of an asymmetric processor to power a subsystem in an upcoming line of phones.

You may assume for this problem that power scales linearly and performance scales by the square root of the area of a core. For example, a core with area 4A will consume 4 times the power and run twice as fast as a core with area 1A.

i. **(3 points)** You are given a square chip of die size **16A** containing one large 3x3 core and seven 1x1 cores. Due to a bad controller, it is **impossible to enable all cores at the same time - either the 7 1x1 cores run together, or the single 3x3 core runs on its own.** Bezels mentions that this core will be used primarily for an application that is **2/9ths serial**. Calculate the maximum speedup achievable using each of the two schemes (single large core vs. many small cores). Which one is better? Why?

Hint: Amdahl's Law will be useful here: $\frac{1}{S+\frac{1-S}{N}}$

ii. **(2 points)** Now, you are assigned to the design of a new processor, still with **total die size 16A.** This processor contains **one large core of size NxN**, and the remaining area on the die is filled with 1x1 cores. **The serial fraction, S, of the program is run on the single large core, and the parallel fraction is run across all cores, including the large core.** Derive an equation based on Amdahl's Law modeling the maximum speedup of this processor over a 1x1 core based on N and S. **No credit will be given without work.**

iii. **(3 points)** Now, Bezels would like you to finalize the design of the processor you modeled in part 2 by choosing the value of N - again, the large core will have dimensions NxN, where N is a natural number. To that end, he gives you the expected workload of this processor, which will only run a few specific functions, whose specifications are listed in the table below:

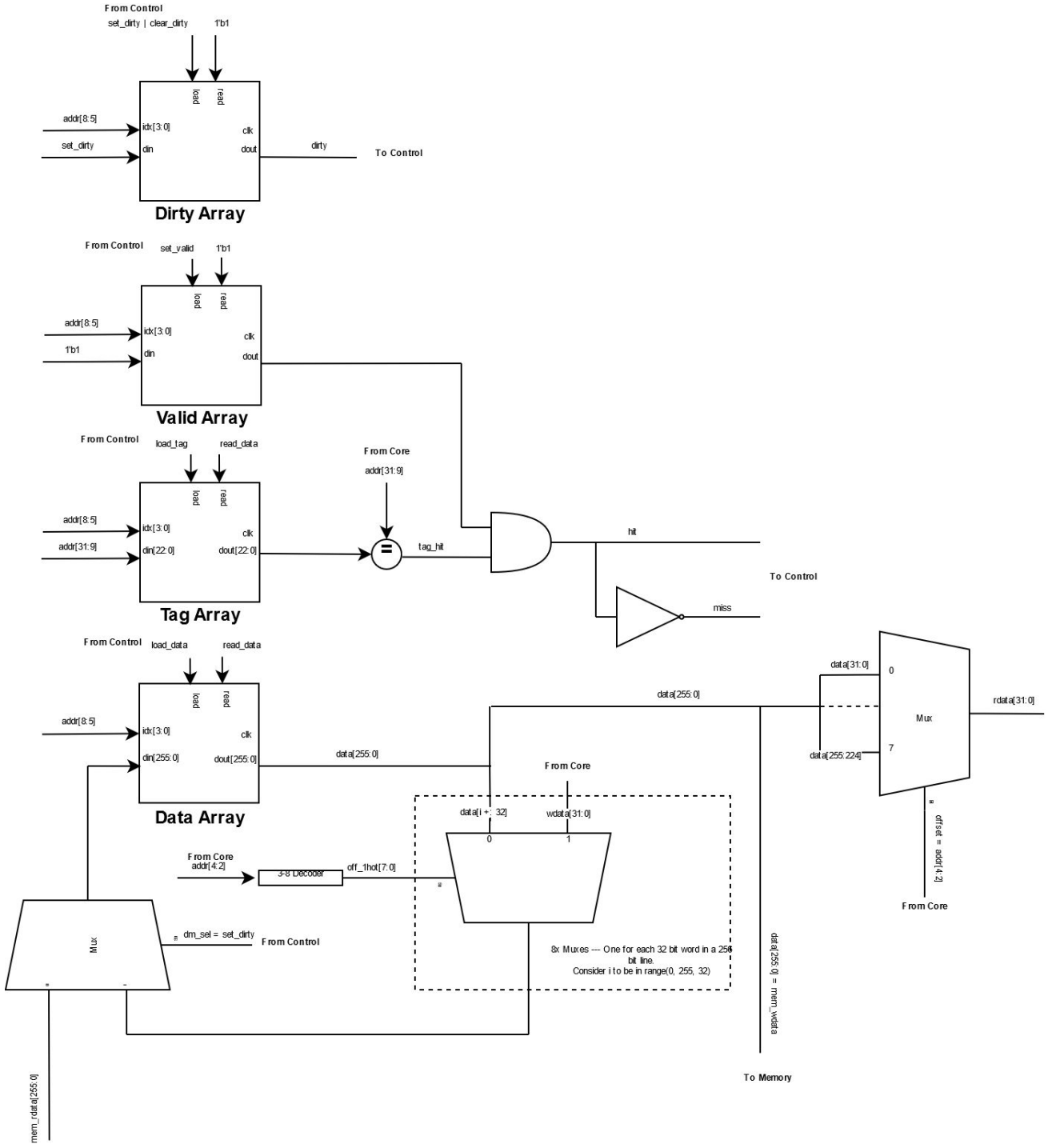
	Code A	**Code B**	**Code C**
% Time Running	25	40	35
Serial Portion	30%	55%	95%

What is the value of N that will give the largest speedup on average? Also report the speedup value for your chosen N. You may disregard the effects of power for this question, and assume that the % time running is not affected by processor speed (ie. bound to external clock). **Partial work is required to receive credit.**

If you were unable to derive an answer for part (b), use this alternate equation: $\frac{N(16+N-N^2)}{16S-SN^2+N}$

c. **(0.5 point + 0.5 EC)** What is your favorite class that you took this semester? Why? Correct answers will earn an additional bonus point.

Appendix


From Control
set_dirty | clear_dirty
1'b1
load
read
addr[8:5]
idx[3:0]
clk
set_dirty
din
dout
dirty
To Control
Dirty Array
From Control
set_valid
1'b1
load
read
addr[8:5]
idx[3:0]
clk
1'b1
din
dout
Valid Array
From Control
load_tag
read_data
From Core
addr[31:9]
load
read
addr[8:5]
idx[3:0]
clk
addr[31:9]
din[22:0]
dout[22:0]
tag_hit
hit
To Control
miss
Tag Array
From Control
load_data
read_data
data[31:0]
0
data[255:0]
rdata[31:0]
Mux
load
read
addr[8:5]
idx[3:0]
clk
din[255:0]
dout[255:0]
data[255:0]
data[255:224]
7
From Core
data[i +: 32]
wdata[31:0]
0
1
offset = addr[4:2]
From Core
From Core
addr[4:2]
3-8 Decoder
off_1hot[7:0]
Mux
dm_sel = set_dirty
From Control
8x Muxes --- One for each 32 bit word in a 256 bit line.
Consider i to be in range(0, 255, 32)
data[255:0] = mem_wdata
To Memory
mem_rdata[255:0]
Data Array