4. [14 points] Out-of-Order

(a) Dr. Chip D. Signer wants to provide higher performance, by treating the N, Z and P bits as separate 1-bit registers and renaming each individually. Will this help performance? Why or why not?

[4 points] This will not improve performance, as the NZP bits are always read and written together. Treating them as separate registers will result in more overhead: extra pregs, a larger rename table, etc.

(b) Computer architect Meg Ahertz needs your help to perform register renaming for the LC4 code below, by rewriting the insns with physical register names. Assume the system has 16 physical registers p0-p15. The initial mapping of architectural to physical registers is r0:p0, r1:p1 ... r7:p7 and nzp:p8. The initial free list is [p9...p15] where p9 is the next free register.

```
SUB  r2 <- r0, r0
LDR  r0 <- r2, #0
CMP  r2, r0
ADD  r1 <- r2, #3
BRz target
STR  r0, r3, #0
SRL  r3 <- r1, r2
```

[10 points] ½ point for each correct register.
An alternative answer where nzp is allocated before the dest reg is also fine.

```
SUB p9 <- p0, p0 (nzp=p10)
LDR p11 <- p9 (nzp=p12)
CMP p9, p11 (nzp=p13)
ADD p14 <- p9 (nzp=p15)
BRz (reads p15)
STR p11 -> [p3]
SRL p8 <- p14, p9 (nzp=p2)
```

- Why does LDR need to rename the NZP register?
  - because the destination of the LDR instruction is a register.
  - refer to LC4 ISA.

NOTE:
- NZP bit registers must also be updated, but check to see if the instruction requires NZP register.
  example: BRz does not
  STR does not
- This is because the NZP register is updated on any instruction that writes to a register and on CMP instructions

4. [12 points] Out-of-Order

(a) Computer architect C3PU wants to build a two-wide superscalar OoO LC4 pipeline. Assuming the design has a 64-entry ROB, how many physical registers are needed to avoid ever stalling due to a lack of physical registers?

Maximum physical registers per instruction = 2 (NZP + destination)
Architectural registers = 8

$d \times K + A$

physical registers: $= 2 \times 64 + 8$
$= 128 + 8 = 136$

(b) Chip designer R2P2 needs your help to perform register renaming for the LC4 code below, by rewriting the insns with physical register names. Assume the system has 16 physical registers p0-p15. The initial mapping of architectural to physical registers is r0:p0, r1:p1 ... r7:p7 and nzp:p8. The initial free list is [p9...p15] where p9 is the next free register.

```
ADD  r2, r1, r0
SUB  r0, r1, r2
CMPU r2, r4
SRA  r5, r2, #3
BRnp target
STR  r0, r3, #0
LDR  r2, r0, #1
```

```
ADD  p9 ← p1, p0        NZP = p10
SUB  p11 ← p1, p9       NZP = p12
CMPU p9, p4             NZP = p13
SRA  p15 ← p9, #3       NZP = p15
BRnp target             NZP = p15
STR  p11 ← [p3, #0]     NZP = p15
LDR  p2 ← [p11, #1]     NZP = p8
```

CIS 501 – Computer Architecture | **Final Exam**
Prof. Devietti | **December 20, 2017**

4. [20 points] Out-of-Order

(a) Perform register renaming for the code below, by rewriting the insns with physical register names. Assume the system has 4 architectural registers r0-r3 and 8 physical registers p0-p7. The initial mapping of architectural to physical registers is r0:p0, r1:p1, r2:p2, r3:p3. The initial free list is [p4,p5,p6,p7], p4 is the next free register.

```
sub r1, r0   -> r2
add r2, 6    -> r1
div r3, r2   -> r0
load [r0,2]  -> r2
store r1     -> [r2,3]
xor r0, r1   -> r0
```

Initial Mapping
R0 → P0 P6 P2
R1 → P1 P5
R2 → P2 P4 P7
R3 → P3

Free
P4 ×
P5 ×
P6 ×
P7 ×
~~P2~~ ×
P1
P0
P4
P7

SUB P1, P0 → P4
ADD P4, 6 → P5
DIV P3, P4 → P6
LOAD [P6, 2] → P7
STORE P5 → [P7, 3]
XOR P6, P5 → P2

- Why does R2 retain the mapping to P7 in instructions (4) and (5)? (WAW should be eliminated right?)
- R2 is only read and used to calculate the memory offset

(b) In an OoO processor, if a physical register *p* is not on the free list, list the **two** other places where it may be.

(i) It could be in the MapTable (some register maps to it).
(ii) Overwritten physical register field of some renamed instructions ROB entry.

(c) Consider the insns in the scalar OoO pipeline diagram below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
load [r1] -> r3	F	De	Rn	Di	I	RR	X	M1	M2	M3	W	C					
store r0 -> [r3]		F	De	Rn	Di						I	RR	X	SQ	C		
store r4 -> [r5]			F	De	Rn	Di	I	RR	X	SQ						C	
load [r3] -> r6				F	De	Rn	Di					I	RR	X	M_1	M_2	M_3

In what cycle should the 2nd load issue under...

i) conservative load scheduling?

Anytime on or after CYCLE 12

(i) this needs to be here

ii) maximally optimistic/aggressive load scheduling?

CYCLE 8

iii) perfect load scheduling?

CYCLE 12

→ somewhat ill-posed because what if R3 = R5? and what if R3 ≠ R5?

3.) Order of Out (20 points total)

Complete the cycle chart for the program listing below, including arrows indicating all bypassed values (including from W to D). Assume a dual-issue out-of-order pipeline with full bypassing. The front end of the pipeline contains the Fetch (F) and Decode (D) stages, with register read occurring in decode. The back end contains the Issue (I), Execute (X), Memory (M), and Writeback (W). And of course there is a Commit (C) stage at the very end. Branches are predicted not taken and are resolved in execute, just like in your assignments. The pipeline can also retire at most two instructions per cycle. **Do not include squashed instructions in the chart.** You should not need all the rows or columns provided. The first instruction has been filled in for you.

```
.CODE
.ADDR 0x0000
I0 CONST R7, 0
I1 LDR   R5, R7, #6
I2 LDR   R2, R5, #4
I3 ADDI  R7, R7, #0
I4 BRz   I7
I5 ADDI  R5, R5, #0
I6 BRz   .END
I7 STR   R5, R7, #1
I8 RET
.END (program ends here)
```

Label	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
I0	F	D	I	X	M	W	C																		
I1	F	D		I	X	M	W	C																	
I2		F	D			I	X	M	W	C															
I3		F	D	I	X	M	W			C															
I4			F	D	I	X	M	W			C														
I7							F	D	I	X	M	W	C												
I8							F	D	I	X	M	W	C												
I0												F	D	I	X	M	W	C							
I1												F	D			I	X	M	W	C					

3.) Cuckoo for Cocoa Puffs (24 points total)

Chris and Andy are cuckoo for Cocoa Puffs, and are therefore running amok and doing their grading completely out of order. Shrek has been trying to figure out if this increases their efficiency or not, but ever since he got hit on the head with the coconut, he's been cuckoo himself and needs your help.

3.1) (4 points) Give two disadvantages of static code scheduling. Each reason must be **10 words or less**, but does not need to be a complete sentence.

Reason 1: Limited scope because of branches (iii) limited by architectural registers

Reason 2: Load/store aliasing (iv) unknown cache miss latency

3.2) (4 points) Give two disadvantages of dynamic codes scheduling. Each reason must be **10 words or less.**

Reason 1: Requires more hardware (iii) branch misprediction is more expensive

Reason 2: Requires longer pipelines (iv) uses more power (v) more hazard detection

3.3) (4 points) What are two mechanisms that dynamic code scheduling uses to mitigate the shortcomings of static code scheduling, and why does each mechanism help? Each answer must be **10 words or less**, and still does not need to be a complete sentence.

Mechanism 1: Branch prediction - speculative execution

Mechanism 2: Hardware to detect dependency violation

3.4) (6 points) What types of dependencies can occur during out-of-order execution; explain each type of dependency in **15 words or less.**

RAW
WAR
WAW

3.5) (6 points) Which of the dependencies that come up during out-of-order execution can be totally eliminated, and what techniques are used to do so? Your answer should list dependencies by their acronyms and also list the techniques; it does not need to say which specific techniques eliminate each type of dependence. Your entire answer must be **20 words or less.**

WAR and WAW using Register Renaming + Store Queue

5. [12 points] Register Renaming

(a) [2 points] For an OoO processor with a k-entry ROB, how many physical registers are needed to ensure that the Rename stage never runs out of physical registers? Make your ISA assumptions clear.

$d \times k + A$

d: maximum number of destination registers produced by any instruction
A: number of architectural registers

(b) [2 points] List the two types of dependences that register renaming eliminates.

WAR : Write after Read
WAW : Write after Write

(c) [8 points] The following insns have been renamed, and parentheses indicate the map table entry overwritten by renaming the given insn. Write the **original insns** with architectural register inputs/outputs. The system has 4 architectural registers `r0-r3` and 8 physical registers `p0-p7`.

The map table after the final insn is renamed is:
`r0:p1  r1:p4  r2:p7  r3:p5`

The free list is empty after the final insn is renamed.

```
add   p1, p0 -> p4       (r1:p1)
mul   p2, p0 -> p5       (r3:p3)
load  [p5+2] -> p6       (r2:p2)
sub   p0, p6 -> p7       (r2:p6)
store p7     -> [p4+4]
xor   p7, p4 -> p1       (r0:p0)
```

ADD R1, R0 → R1
MUL R2, R0 → R3
LOAD [R3+2] → R2
SUB R0, R2 → R2
STORE R2 → [R1+4]
XOR R2, R1 → R0

6. [9 points] Load/store queue (LSQ)

(a) [3 points] When a load L searches the store queue, what are the 3 criteria we use to find a store that should forward its value to L?

YOMS
(i) Youngest
(ii) Older
(iii) Matching store

(b) [2 points] Describe a problem that arises if addresses in the LSQ are *virtual*.

We cannot perform correct STORE ⇒ LOAD forwarding for virtual memory aliases

(c) [1 point] Describe a problem that arises if addresses in the LSQ are *physical*.

Require an additional ADDRESS TRANSLATION stage in the pipeline

(d) [3 points] Consider the following load queue state:

birthday	size	address	birthday of forwarding store
3	1	0x16	1
5	4		

i) If a store with birthday 0 executes, writing 1B to address 0x16, what (if any) load(s) should get squashed?

No one needs to get squashed

ii) Given the same load queue state, if a store with birthday 2 executes, writing 4B to address 0x14, what (if any) load(s) should get squashed?

(i) Load with birthday = 3 should get squashed
(ii) Load with birthday = 5 because it could depend on the previous load (birthday = 3)

4. [18 points] OoO Execution

(a) [6 points] Perform register renaming for the code below, by rewriting the insns with physical register names. Assume the system has 4 architectural registers `r0-r3` and 8 physical registers `p0-p7`.

The initial mapping of architectural to physical registers is:
`r0:p0  r1:p1  r2:p2  r3:p3`

The initial free list is `[p4,p5,p6,p7]` in that order; p4 is the next free register.

```
xor r0, r1    -> r1
store r2      -> [r3+4]
sub r3, r1    -> r2
load [r2+r3]  -> r0
add r0, r1    -> r2
mul r2, r2    -> r2
```

XOR P0, P1 → P4
STORE P2 → [P3+4]
SUB P3, P4 → P5
LOAD [P5+P3] → P6
ADD P6, P4 → P7
MUL P7, P7 → P1

(b) [1 point] What is conservative load scheduling?

Loads are not ISSUED until all older stores have EXECUTED.

(c) [2 points] Under conservative load scheduling, is a *store queue* necessary? Why or why not?

Yes, a STORE QUEUE allows O-o-O execution and memory forwarding.

(d) [2 points] Under conservative load scheduling, is a *load queue* necessary? Why or why not?

No, because there won't be any ordering violations of load instructions.

(e) [2 points] Give two advantages of having in-order commit

i) Easy recovery

ii) maintains program order

(f) [5 points] Dr. Chip D. Signer has come up with a new performance optimization for OoO cores. He wants to free physical registers before commit by tracking a reference count for each physical register. When a new physical register p0 is allocated during Rename, its reference count is initialized to zero. When an insn *i* enters Rename and uses a physical register p0 as one of its sources, the reference count for p0 is incremented by one. When *i* finishes the Register Read stage, p0's reference count is decremented. Once p0's reference count reaches zero again, p0 is freed and can be reallocated for another insn.

Dr. Signer's scheme is clever but doesn't quite work. Describe a series of events that would cause Dr. Signer's scheme to execute a program incorrectly.

5. [8 points] Register Renaming

(a) [2 points] Dr. Chip D. Signer is considering building an OoO processor without register renaming. He claims this will help improve performance by removing a pipeline stage. Is this a good idea or not? Why or why not?

(i) Not a good idea.
(ii) False dependencies will cause trouble and lower the ILP.

(b) [6 points] Perform register renaming for the code below, by rewriting the insns with physical register names. Assume the system has 4 architectural registers `r0-r3` and 8 physical registers `p0-p7`.

The initial mapping of architectural to physical registers is:
`r0:p0  r1:p1  r2:p2  r3:p3`

The initial free list is `[p4, p5, p6, p7]` in that order; p4 is the next free register.

```
load [r1+4] -> r2
add r1, r2  -> r3
sub r3, r1  -> r2
mul r0, r2  -> r2
add r0, r1  -> r1
store r2    -> [r1+8]
```

LOAD [P1+4] → P4
ADD P1, P4 → P5
SUB P5, P1 → P6
MUL P0, P6 → P7
ADD P0, P1 → P2
STORE P7 → [P2+8]

6. [7 points] OoO Memory Operations

(a) [1 points] What is one specific role of the Store Queue in enabling OoO memory operations?

(i) Allows STORES to execute out of order
(ii) Allows memory forwarding (STORE ⇒ LOAD)
(iii) Cleanup of speculative STORES

(b) [1 points] What is one specific role of the Load Queue in enabling OoO memory operations?

Checks for LOAD ordering violations

(c) [1 points] What does conservative load scheduling mean?

LOADS are not ISSUED before STORES have EXECUTED.

(d) [4 points] Consider a snapshot of an OoO core, where the store queue has the following state:

birthday	address	value
0	0x18	6
3		

i) If a load of address 0x18 executes under the aggressive load scheduling policy and with the above store queue state, what value should the load return?

since the store (1) has birthday=0
the load should return 6

ii) Describe the conditions under which the load may get squashed, or describe why the load will never get squashed.

squashing if:
(i) pending store (birthday == 3)
writes to address 0x18
and is older than the load