The University of Texas at Austin
EE460N/EE382N.1 Exam 2 (Fall 2015)

Mattan Erez, Alex Hsu, Tommy Huynh, Anoop Naravaram, Kishore Punniyamurthy

Your signature is your promise that you have **not cheated** and will not cheat on this exam, nor will you help others to cheat on this exam. Cheating means exchanging any information whatsoever with any other entity (with the exception of private searches). **Plus -- no typing (except minimally for search)!**

Signature:		Name:		EID:	

Note: Please be sure that your answers to all questions (and all supporting work that is required) **are contained in the box provided for each question;** we reserve the right to ignore anything else.

Note: **"I DON'T KNOW"** is a valid answer that automatically gives you 15% of the maximum possible grade. Be sure to either erase, or cross-out everything else and write I DON'T KNOW in, large, capital letters. A blank answer does not get you the 15%. No partial I DON'T KNOW credit (only on entire questions/parts that have a marked point value on the exam).

Note: Please put your name at the top of each page of the exam!

Note: For all questions, unless otherwise stated, find the most efficient solution. Efficiency counts.

The exam has a total of 3 questions and 4 pages. It's a good idea to read through first because some questions might be easier for you than others.

For each page that doesn't include your name and EID you will be deducted 1 point!

Question 1 (35 points total)

Consider an LC3b machine which supports VAX like virtual memory. The physical address space is 4KiB and the virtual space is 64KiB. The 2 most significant bits of virtual address indicate the memory region as in VAX.

Consider 2 processes PA and PB, both the processes have some shared memory in P0 memory space.

PA's P0 Base Register = x8000, SBR = x0100, PB's P0 Base Register = x8300

The PTE is of the form:

7	6:4	3:0
V	000	PFN

<<<Continued on next page>>>

Name: EID:

PA executes the following code, to store data into the shared memory.

```
      .ORIG x3000
      LEA R0,ADDR
      LDW R1,R0,#0
      AND R2,R2,#0
      ADD R2,R2,#1
      STW R2,R1,#0;      store in shared memory location
      HALT
ADDR  .FILL x2A2C
      .END
```

The above code generated 3 physical memory access while performing the STORE to the shared memory location. Ignore other memory access (including instruction access).

Find each of the following 5 items (4 points each):
Part a. The virtual page number to which the shared location belongs: ___**x2A**_(42)__
Part b. The virtual address of the PTE for the page to which shared location belongs: __**x802A**__
Part c. Physical address in the system page table used to locate PTE of shared location: __**x0100**__
Part d. Physical address of the PTE for the page to which shared location belongs: __**x022A**__
Part e. Physical address of shared location to which the store is performed: __**x012C**__

Assume PB runs after PA has completed execution. No page fault occurred with either PA or PB.
Part f (15 points): Complete the incomplete code of PB below, which is executed by PB to read the shared data modified by PA into R2.

```
      .ORIG x2000
      LEA R0,ADDR
      LDW R1,R0,#0
      LDW R2,R1,#0
      HALT
ADDR  .FILL x282C
      .END
```

The contents of physical memory is shown below (values are in hex).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x0100	82	6	E	83	D	8F	87	9	34	8D	8C	3	23	1F	4F	2A
0x0120	2D	0	49	62	1	0	30	15	A0	C3	2F	F3	1	0	1	24
0x0140	50	39	53	49	BC	42	46	B7	C0	D5	19	FD	31	31	3A	10
0x0160	1F	21	83	BE	6	89	4C	6E	20	E1	E5	AB	8	6B	4A	8F
0x0180	3B	CC	37	78	8A	E0	2	F2	24	99	31	DF	C2	60	F0	50
0x0200	29	5D	D6	DE	88	C2	8A	4D	A5	4C	7	5C	32	7C	18	3
0x0220	A	E	80	8F	1	0	6	8C	85	3	81	83	A	8D	2	0
0x0240	4D	5A	3F	E7	43	9D	EF	16	6F	7E	87	DE	A0	9A	39	5E
0x0260	AD	F9	10	B3	4D	51	FA	E	5	FD	1	0	C3	85	C7	90
0x0280	E	5	8A	81	E	9	4C	4	3E	4D	4F	9A	3	21	26	7A
0x0300	5E	8D	79	48	3F	18	3	F2	0	2A	55	E3	7F	61	11	2C
0x0320	4	7	8F	D	8E	0	8C	A	81	8B	83	A	F	4	89	6

Name: EID:

Question 2 (35 points total)

You are asked to reverse engineer a scoreboard design. The following program takes 16 cycles to execute on the system. There is one register writeback port and the arbitration scheme for writeback is oldest-instruction-writes-back-first. The pipeline is 4 stage (FDEW). There is 1 fetch, 1 decode, and 1 writeback unit. Fetch and execute can happen concurrently. Bypass is available in the writeback stage.

```
1. ADD R0, R1, R2
2. ADD R1, R1, R3
3. MUL R2, R3, R4
4. MUL R4, R3, R3
5. MUL R7, R0, R3
6. ADD R5, R4, R3
7. ADD R6, R0, R1
```

The following table shows the busy cycles of these units.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
MULTIPLIER					X	X	X	X	X	X	X	X	X	X		
ADDER			X	X	X	X					X	X	X	X	X	

Part a (12 points):

Adder(s): How many (minimum needed)? **1** Are they pipelined? [✓ **Yes**] / No

Multiplier(s): How many (minimum needed)? **2** Are they pipelined? Yes / [✓ **No**]

How many cycles does the adder take if there is no dependency? **3 cycle(s)**

How many cycles does the multiplier take if there is no dependency? **5 cycle(s)**

Part b (23 points):

Show the execution stages of the program:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
R0=R1+R2	F	D	A1	A2	A3	W										
R1=R1+R3		F	D	A1	A2	A3	W									
R2=R3*R4			F	D	M	M	M	M	M	W						
R4=R3*R3				F	D	M	M	M	M	M	W					
R7=R0*R3					F	D	D	D	D	M	M	M	M	M	W	
R5=R4+R3						F	F	F	F	D	A1	A2	A3	W		
R6=R0+R1										F	D	A1	A2	A3	A3	W

Name: EID:

Question 3 (total 30 points)

You are given a code snippet below, which was written for a traditional 5-stage in-order single-functional unit pipeline (FDEMW). Each stage of this pipeline was meant to have a latency of 1 second (yes, second). There is no result forwarding path, and the value produced by one instruction is available for other instructions to read one cycle *after* WB. However, unconditional branches update the PC immediately at the end of the decode stage (full decode latency). You are tasked with making modifications to the architecture, microarchitecture, and codes to accommodate the fact that the register file turned out to be much slower than expected. The resulting slowdown caused the decode and writeback stages to take 3s each with the original design, causing the clock period to be lengthened accordingly.

You are specifically interested in optimizing processor execution for the following loop **in steadystate**:

```
LB R0=R0+R1
   BRn LB   ; Practically always not taken
   R0=R0+R3
   R2=R0
   BRnzp LB
```

Part a (10 points): If you make no changes to the processor or code, what would be the performance of this code on this processor? State your answer in terms of instructions per second (IPS), note that your answer is probably less than 1 IPS.

_____**5/(11*3) = 5/33 IPS**__________

```
FDEMW
 FDEMW
  FDDDEMW
   FFFDDDDEMW
      FFFFDEMW
          FF(starting point)
```
11 cycles steady state
Would also accept overlapping register read in decode with writeback, saving 3 cycles.

Part b (20 points): In order to improve performance, your friend decides to pipeline the decode and writeback stages such that each has 3 stages. You are allowed to make a single additional change to the architecture and/or microarchitecture and make the appropriate changes to the code. Would you choose to: (1) change *all* branch instructions to have 2 delay slots, *or* (2) add a forwarding path for bypassing the writeback stage and using the outputs of the execution unit for back-to-back instructions.
Given your choice, what is the IPS now? Explain your answer (briefly) in the box below!

____**5/(8*1) = 5 / 8 IPS**__________

The code has 2 RAW dependencies that are pretty tight (one back-to-back), and one unconditional branch. Two delay slots would reduce 3 cycles of waiting for decode to complete for the unconditional branch down to 1, saving 2 cycles steady state. Forwarding paths save much more.

```
FDDDEMWWW
 FDDDEMWWW
  FDDDEMWWW
   FDDDEMWWW
    FDDDEMWWW
     FFFF(starting point)
```