Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 460N Spring 2015
Y. N. Patt, Instructor
Ben Lin, Kishore Punniyamurthy, Will Hoenig TAs
Exam 1
March 11, 2015

Name: SOLUTION SHEET

Problem 1 (20 points):_____

Problem 2 (30 points):_____

Problem 3 (25 points):_____

Problem 4 (25 points):_____

Total (100 points):_____

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

Please sign the following. I have not given nor received any unauthorized help on this exam.

Signature: Solution Sheet

GOOD LUCK!

Name:_____

Problem 1 (20 points)

Part a (5 points): The x86 ISA has variable length instructions. This characteristic of the ISA has pluses and minuses.

The major plus is

DENSE ENCODING OF THE INSTRUCTIONS

The major minus is

DECODING MULTIPLE INSTRUCTIONS CONCURRENTLY IS VERY DIFFICULT

Part b (5 points): If two locations are in the same row buffer on a DRAM, the only bits of their corresponding addresses that are different are the

COLUMN BITS

Part c (5 points): If the contents of a memory location is protected with a parity bit, and two bits are inverted during transmission, what happens? Is this a problem? Why or why not?

ERROR WILL NOT BE DETECTED
NOT REALLY A PROBLEM IT RARELY HAPPENS (STATISTICAL INDEPENDANCE OF ERRORS)

Part d (5 points): If two processes translate different virtual addresses in their own virtual address space to the same physical address, and the cache is virtually indexed, physically tagged, the location corresponding to that one physical address can be present in two different locations in the cache. We call the two instances of the same physical cache line SYNONYM and the problem is referred to as the SYNONYM PROBLEM.

Name:_____

Problem 2 (30 points)
A computer system contains a physically-indexed, physically-tagged, 2-way set associative, write-back cache. The ISA specifies an n-bit physical address space, and an 128 byte page size.

With the cache initially empty, the processor makes ten consecutive memory reads, as shown in the table below. Note that some result in cache hits, others result in cache misses. Note that the actual number of bits of physical address is not shown.

	Physical Address	Hit/Miss
1	00...0000010000	Miss
2	00...0100000101	Miss
3	00...0000011000	Hit
4	00...0110000111	Miss
5	00...0111100001	Miss
6	00...0100000010	Hit
7	00...0110100111	Miss
8	00...0100100000	Miss
9	00...0111100010	Miss
10	00...0100001111	Hit

index bits must end ↗ before bit 7.

In order to concurrently access the TLB, the Tag Store, and the Data Store, the index bits are selected from the page offset, i.e., they are not affected by the virtual to physical address translation.

Part a (10 points): What is the cache line size? Why?
→ off of access 1
- access 3 can only be a hit if byte-on-line bits ≥ 4. → so, BoL bits = 4 ∴ line size = 2^4 B.
- access 7: if there were 5 BoL bits, contents from access 2 would be evicted under any legal number of index bits. ∴ BoL bits < 4.

Part b (10 points): What are the index bits? Why?
- If we had 3 index bits, access 9 would have been a hit, of access 5.
- If we had 1 index bit, access 10 would have been a miss. That tag would have been evicted. But 2 works. ∴ 2 index bits, 5 and 4.

Part c (5 points): What is the size of the cache (i.e., how many bytes of data can be stored in the cache)?

4 sets/cache (implied by part b) · 2 ways/set (given) · 2^4 bytes/line (part A) = 128 bytes

Part d (5 points): Given that the tag store requires 68 bits, and that the cache uses perfect LRU replacement, what is the physical address space of memory for this computer system.

$\frac{68}{4} = 17$ bits/set

16+1 LRU bit

16 = 2·8 → 8 bits/line

8 = V + M + tag

tag = 6 bits → phys addr =
6 tag + 2 index
+ 4 BoL = 12 bits

2^{12} Byte address space

Name:_____

Problem 3 (25 points)

The following program fragment executes on a machine that supports virtual memory.

```
LD R0, A        ;R0 <-- M[A]
LD R1, B        ;R1 <-- M[B]
ADD R1, R1, R0  ;R1 <-- R1 + R0
ST R1, C        ;M[C] <-- R1
```

A, B, and C are virtual addresses.

BOP = 8 bits

The ISA specifies:
Virtual address space: 64KB PFN = 5 bits
Physical address space: 8KB
A page is 256 bytes

The memory management system uses the two-level page table scheme similar to the VAX. Virtual memory is partitioned into two halves. User space starts at x0000, System space starts at x8000. A PTE is 16 bits. For purposes of this exam only, we will assume that a PTE has the following form:

V	00..0	PFN

Assume no cache and no shared memory(that is, no two pages map to the same frame). Assume the TLB only contains PTEs for pages in user space.

For the snippet of code above, if one gets no TLB hits, the processor makes nine accesses to physical memory (we are ignoring the fetching of instructions). The table below shows the sequence of nine memory accesses needed to do the job.

SBR = x1AA8 − (2 x x01) = x1AA6

Virtual Address	Physical Address	Data
—	x1AA8	x8016
x8124	x1624	x8007
x5400	x0700	x5410
—	x1AA6	x8008
x80C2	x08C2	x8006
x2346	x0646	x0414
—	x1AA8	x8016
x8168	x1668	x8005
x7618	x0518	x5824

PageNo. BOP
x23 x46 =

(C − A)
= x5824 − x5410

PBR = x80C2
− (x23 x 2)

On the other hand, IF the TLB initially contained the entries shown below,

Page No. Size of PTE

PBR = x807C

TLB HIT →

V	Page No	PTE
1	x14	x8001
1	x23	x8006
1	x28	x8004
0	-	-

← PFN bits match with Page frame of address x0646

the processor would only need to make seven accesses to physical memory.

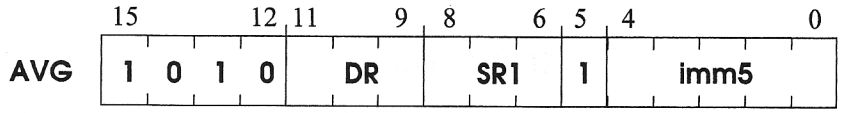
1 TLB HIT

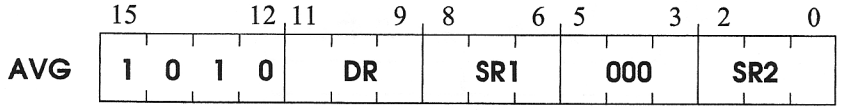
Your job: Complete the table.

Name:_____

Problem 4 (25 points)

Let's use one of the unused opcodes to add an instruction AVG (i.e., average) to the LC-3b ISA. Its format will be





depending on whether the second operand is an immediate or the contents of a register. AVG will sum the n ($n > 0$) 16-bit integers in consecutive memory locations starting at the location specified by SR1, divide that sum by n, and load the result into DR, setting the condition codes. The value n is either an immediate value or the contents of SR2. Assume that the n integers are aligned in memory (i.e. SR1 holds an even number), and assume that DR, SR1, and SR2 (if SR2 is being used) all refer to different registers.

For this problem, you can assume no overflows will occur. Note: execution of this instruction will destroy the initial contents of SR1.

Your job: augment the LC-3b state machine, the data path and the microsequencer as necessary to add AVG to the LC-3b ISA.

Part a (8 points): The state machine (see page 6). From state 32 (the decoder) we go immediately to the eight states needed to carry out the work of the AVG instruction. One of the states has been specified for you, and another (state 39) has been partially specified. **Your job is to complete the specifications of all the states and add the missing state numbers**

Part b (8 points):
The data path (see page 7). To implement AVG you will need additional structures. Four are shown in boldface on the data path diagram.

The DIVIDE UNIT takes two inputs X,Y and produces a result X/Y. The divide is a multi-cycle operation that latches X, Y internally in the first cycle, and signals completion with a DIV_R (for Divide Ready) signal when done. The DIVIDE UNIT starts processing when the control signal DIV is asserted. **Your job is to connect the DIVIDE UNIT to other elements of the data path as needed.**

The CTR is a step-down counter. It can be loaded when a LD.CTR control signal is asserted, and it can be decremented when a DEC.CTR control signal is asserted. **Your job is to connect it to other elements of the data path as needed.**

Registers containing x0000 and x0002 have also been added to the data path. **Your job is to connect it to other elements of the data path as needed.**

Feel free to add tri-state devices for any signals you wish to put on the bus.

There is also a dashed box in the data path. **Your job is to put in that box any other necessary structures(register or combinational logic) to complete the data path for implementing AVG.**

Part c (9 points): The microsequencer (see page 8). The augmented state machine requires additional control provided by the microsequencer. **Your job is to augment the microsequencer.** You will need an additional COND bit, call it COND2, which will be used to modify J[5] and J[4] when necessary. The necessary OR gates have already been put in place. Add whatever additional logic structures you need.

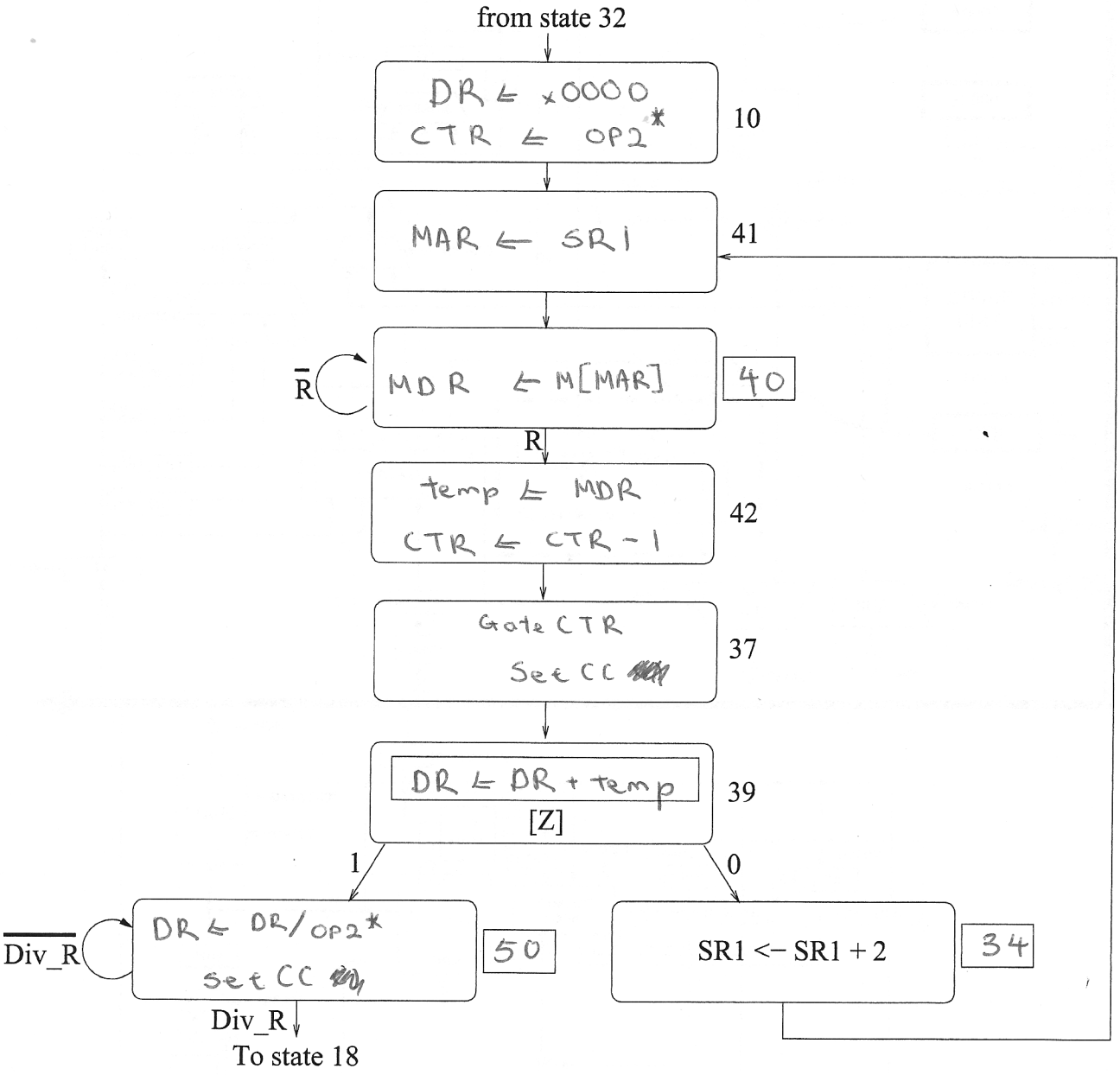


Figure 1: State diagram for AVG instruction

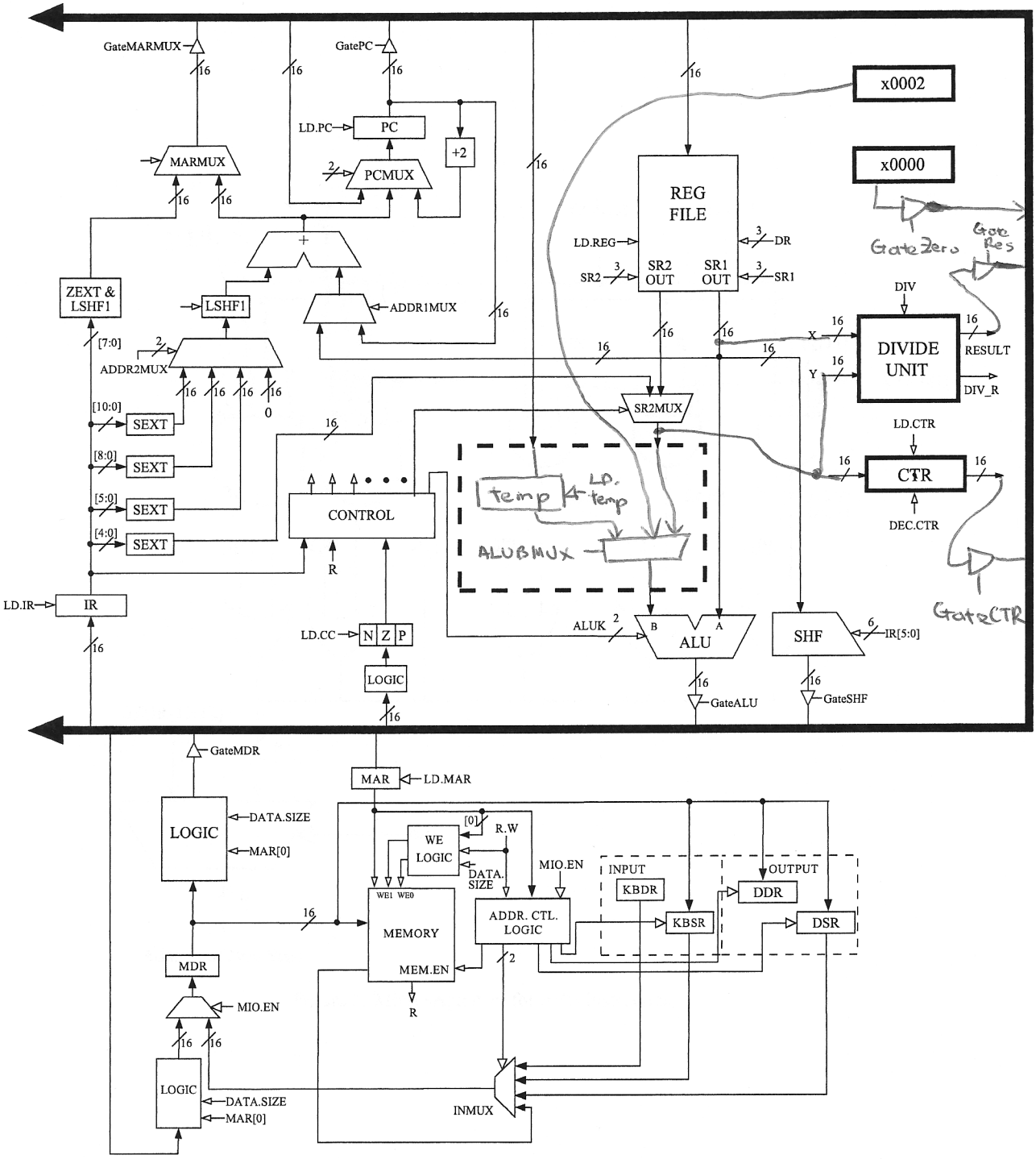


Figure 2: Data path for AVG instruction

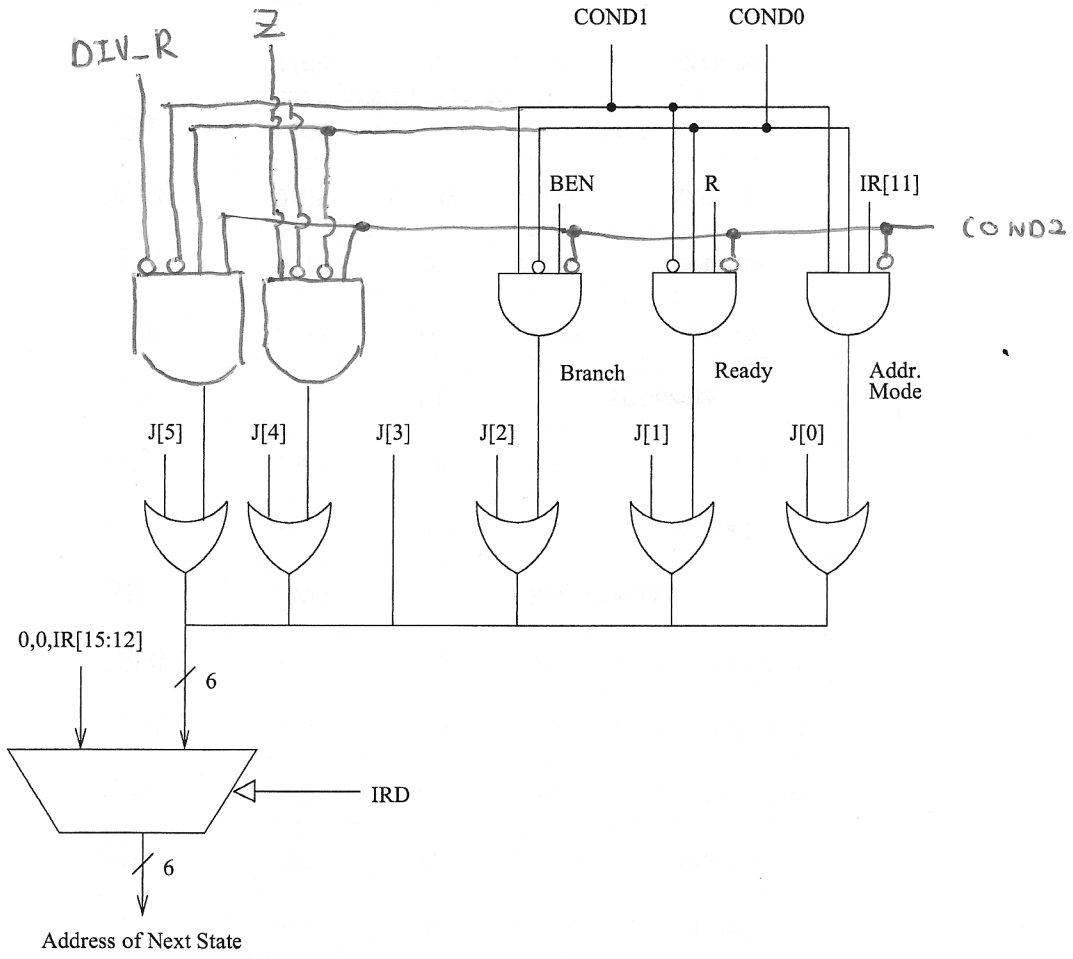


Figure 3: Microsequencer for AVG instruction