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

EE 360N, Spring, 2001
Yale Patt, Instructor
Onur Mutlu, Kameswar Subramaniam, TAs
Exam 2, April 18, 2001

Name:________________________________

Problem 1 (10 points):_________

Problem 2 (10 points):_________

Problem 3 (15 points):_________

Problem 4 (10 points):_________

Problem 5 (20 points):_________

Problem 6 (20 points):_________

Problem 7 (15 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.

GOOD LUCK!

Name:________________________________

Problem 1 (10 points):

Part a (5 points):

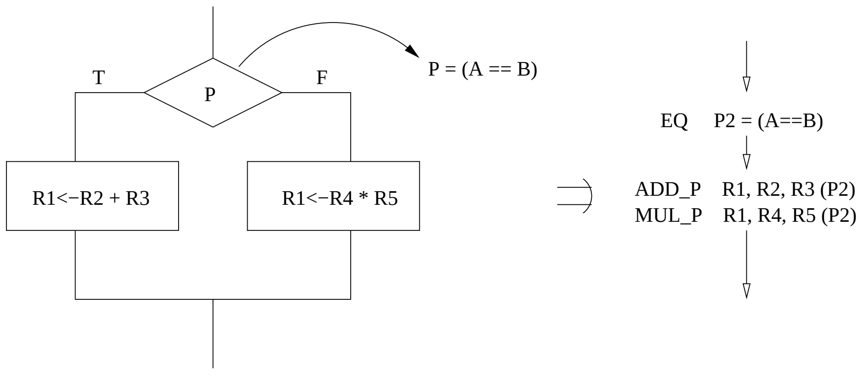
To multiply 011101 by 011101 using Booth's algorithm, how many steps are required? $\boxed{3}$

Show the contents of the temporary result register after each step:

After step 1:	0	0	0	1	1	1	0	1	0	0	0	0
After step 2:	1	1	1	0	1	0	1	0	0	1	0	0
After step 3:	0	0	1	1	0	1	0	0	1	0	0	1
After step 4:												
After step 5:												

Part b (5 points): What benefit does predicated execution provide? How does it provide this benefit?

The benefit of predicated execution is that it removes branches. It provides this benefit by converting the instructions after the branch, upto the merge point, into predicated instructions. For example:



Name:______________________________

Problem 2 (10 points):

In the following code sequence, the destination register for each instruction is indicated first, followed by the two sources. One instruction can be fetched each cycle, decoded and renamed in the next cycle, and issued to the reservation station in the third cycle. Note that there is a single common reservation station for all functional units. Instructions sit in the reservation station until their source operands are valid, at which point they can be scheduled for execution. Assume that the reservation station contains 20 entries, numbered 0 to 19, and that all instructions are issued from the decode and aliasing stage to the next available reservation station entry. Note that this will require adding an operation field to each entry.

```
DIV  R3,R1,R2
ADD  R5,R4,R3
ADD  R6,R4,R1
MUL  R6,R8,R6
ADD  R4,R3,R7
MUL  R9,R5,R6
```

Assume an initially empty reservation station, and the DIV instruction is issued in the first cycle to reservation station 0. Assume the DIV takes 16 cycles to execute, ADD takes 4, and MUL takes 6.

Show the contents of the register file after MUL R9,R5,R6 has been issued to the reservation station.

Initial Contents

	V	tag	value
R0	1	--	5
R1	1	--	10
R2	1	--	3
R3	1	--	20
R4	1	--	2
R5	1	--	1
R6	1	--	6
R7	1	--	7
R8	1	--	50
R9	1	--	10

Contents after MUL R9,R5,R6 issued

V	tag	value
1	--	5
1	--	10
1	--	3
0	0	20
0	4	2
0	1	1
0	3	6
1	--	7
1	--	50
0	5	10

Name:________________________________

Problem 3 (15 points):

A 256 KB write-through cache has been designed to work with a computer that has a 32 MB physical address space. The cache is virtually indexed and physically tagged. The ISA specifies a 32-bit virtual address space and a page size of 4 KB. The cache is 2-way set associative, has a block size of 8 bytes, and uses a perfect LRU replacement policy. The cache is not sectored. How many bits of storage are required to implement the tag store?

Virtual address space = 32bits

Physical address space = 32MB (25 bits)

Block size = 8 bytes (3 bits)

Number of blocks in cache = 256KB/8B = $2^{18}/2^3 = 2^{15}$

Number of indexed rows in cache = Number of blocks / associativity

= 2^{15} / 2 = 2^{14}

Hence, of the virtual address,

VA[2:0] are used for the offset within a block

VA[16:3] are used for the index

Page size = 4 KB (12 bits)

Virtual page number is VA[31:12]

So VA[16:12] are used to index into the cache, but get changed in the process of virtual to physical address translation

Hence, the tag field is the entire physical frame number, i.e. PA[24:12] (13 bits)

Format of tag store entry:

13 bits for tag

1 valid bit

No modified bit (write-through cache)

In addition, each row contains 1 bit for perfect LRU

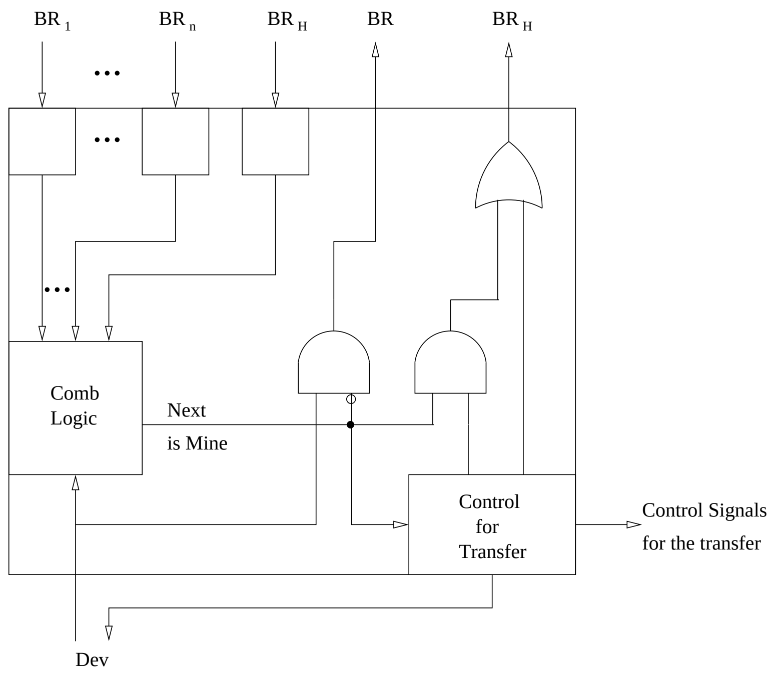
So, number of bits in each row = (14 * 2) + 1 = 29

Size of tag store = 29 * 2^{14} bits = 475136 bits

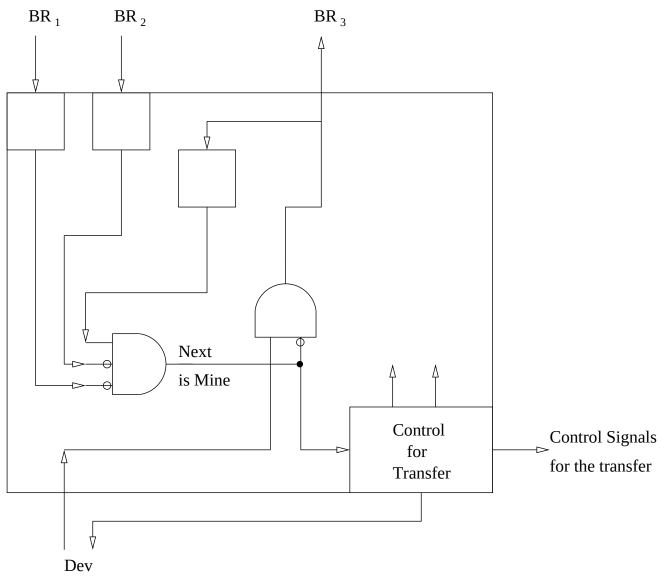
Name:________________________________

Problem 4 (10 points):

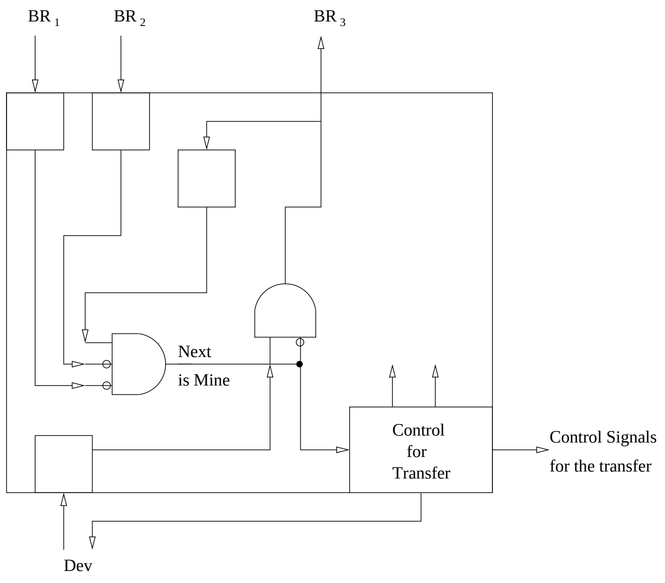
In class, we discussed synchronous distributed arbitration, and used as an example, a generic controller handed out in class. It is reproduced below.



We wish to change the protocol such that every bus transaction takes exactly one cycle. Augment the above figure to (a) incorporate this change, and (b) tailor this figure to handle the device of 3rd highest priority. That is, it requests the bus with the signal BR3. Show all the logic needed to implement the block labeled "Comb Logic" on your figure.



Acceptable solution for full credit



Better solution

Name:________________________________

Problem 5 (20 points):

Part I. If the vector stride register (Vst) has the value S, and the vector length register (Vln) has the value L, what does the following Cray-like assembly language instruction do?

VLD V0, A

Loads L elements starting from address A into vector register V0; incrementing the address by S after each element load.

Part II. Consider a two-dimensional square array with 16 elements (4 rows and 4 columns). The array is stored in row-major order in memory, in 16 consecutive locations starting from address A. Each element occupies one memory location.
We would like to find the sum of the elements in each row in the array and store the result in 4 consecutive locations starting from address B.

Write Craylike assembly code to perform this operation. Hint: You will need to load Vst and Vln with approriate values. (Feel free to use "A + constant" as an addressing mode. For example, Vld V5, A+7)

Note that though the VLDs and VADDs can be ordered differently, this ordering results in the minimum execution time for part III

```
MOVI Vln, 4
MOVI Vst, 4
VLD V0, A
VLD V1, A+1
VLD V2, A+2
VLD V3, A+3
VADD V4, V0, V1
VADD V5, V2, V4
VADD V6, V3, V5
MOVI Vst, 1
VST V6, B
```

Part III. How many cycles will this program take to execute on a vector processor with chaining which has 2 read ports and 1 write port to memory? (Assume a single memory access takes 11 cycles, a single multiply takes 6 cycles, and a single add takes 4 cycles. All the execution units are pipelined. Memory is 4-way interleaved and vector registers have 64 elements in each)

Even if you assumed 16-way interleaving, you would get full credit if it is correct

```
|1-|1-|-----11------|-3-|
       |-----11------|-3-|
                         |-----11------|-3-|
                          |-----11------|-3-|
                           |--4-|-3-|
                                       |--4-|-3-|
                                             |--4-|-3-|
                                              |1-|
                                                  |-----11------|-3-|
```

Number of cycles = 2 + 11 + 3 + 11 + 4 + 4 + (11+3) = 49

4-way interleaving: The assembly code needs to be reordered to get the optimal solution:

```
MOVI Vln, 4
MOVI Vst, 4
VLD V0, A
VLD V1, A+1
VADD V4, V0, V1
VLD V2, A+2
VLD V3, A+3
VADD V5, V2, V4
VADD V6, V3, V5
MOVI Vst, 1
VST V6, B
```

```
|1-|1-|-----11------|-----11------|-----11------|-----11------|
       |-----11------|-----11------|-----11------|-----11------|
                     |--4-|        |--4-|        |--4-|        |--4-|
                                                              |-----11------|-----11------|-----11------|-----11------|
                                                               |-----11------|-----11------|-----11------|-----11------|
                                                                           |--4-|        |--4-|        |--4-|        |--4-|
                                                                                 |--4-|        |--4-|        |--4-|        |--4-|
                                                                                                                      |1-|
                                                                                                                           |-----11------|-3-|
```

Number of cycles = 2 + (4 * 11) + (4 * 11) + 4 + 4 + 11 = 109 (note that the store of the last location can only start when the last add generates it)

Name:________________________________

Problem 6 (20 points):

Part I. Consider the 32-bit IEEE floating point format with 1 sign bit, 8 bits of excess-127 exponent, and 23 bits of fraction. What are the smallest normal and sub-normal positive numbers that can be represented using this format?

```
Smallest normal positive number:
0 00000001 00000000000000000000000
= 2^(1-127) = 2^-126

Smallest subnormal positive number:
0 00000000 00000000000000000000001
= 2^(-23) * 2^(-126) = 2^(-149)
```

Part II. We would like to convert this number into a 40-bit extended floating point format with 1 sign bit, 11 bits of excess-1023 exponent, and 28 bits of fraction. What are the smallest normal and sub-normal positive numbers that can be represented using this extended format?

```
Smallest normal positive number:
0 00000000001 0000000000000000000000000000
= 2^(1-1023) = 2^-1022

Smallest subnormal positive number:
0 00000000000 0000000000000000000000000001
= 2^(-28) * 2^(-1022) = 2^(-1050)
```

Part III. How would you convert a 32-bit format normalized number to the 40-bit format?

```
Sign bit remains the same

Zero extend exponent32 (append 5 zeroes at the beginning),
add 1023-127 = 896 to get exponent40

Append 5 zeroes to the end of the fraction32 to get fraction40

A quick way to get the new exponent in hardware:
Insert the complement of the highest bit of exponent32 duplicated 3 times
between the highest 2 bits of exponent32
```

Part IV. Describe the steps required to convert a subnormal 32-bit number to the 40-bit format. Will these numbers be represented as subnormal or normal numbers in the 40-bit format?

```
Sign bit remains the same

Initially, exponent40 is -126 ie (-126 + 1023) = 897 (01110000001)
fraction40 is exponent32 with 5 zeroes appended at the end

Left shift fraction40 till the 1 gets shifted out (normalize it!)
For each shift, decrement the exponent by 1

All the 32-bit subnormal numbers will be represented as normal numbers
```

Part V. Express the smallest subnormal 32-bit number in the 40-bit format.

```
2^(-149): Sign bit is 0, fraction is 0, exponent = -149+1023 = 874
0 01101101010 0000000000000000000000000000
```

Name:________________________________

Problem 7 (15 points):

In Lab Assignment 3, we were interested in augmenting the microsequencer to allow microbranches to the states that initiate interrupt handling and page 0 access control violations.

In this problem, we are interested in augmenting the microsequencer to take an exception if an illegal trapvector is specified.

We have decided to restrict trapvectors to the range x20 to xFF. That is, the TRAP instruction with trapvectors from x00 to x1F would not be allowed. Your task is to augment the original LC-2 microsequencer to take an exception if an illegal trapvector is present.

Define ITV (for illegal trapvector) as follows:

ITV = 1; illegal trapvector

ITV = 0; not an illegal trapvector

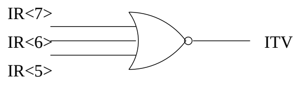
Part a. In what state should ITV be tested? The result of the test is a branch to the state that initiates illegal trapvector processing, or to continue normally.

In state 9

In what state should ITV be computed?

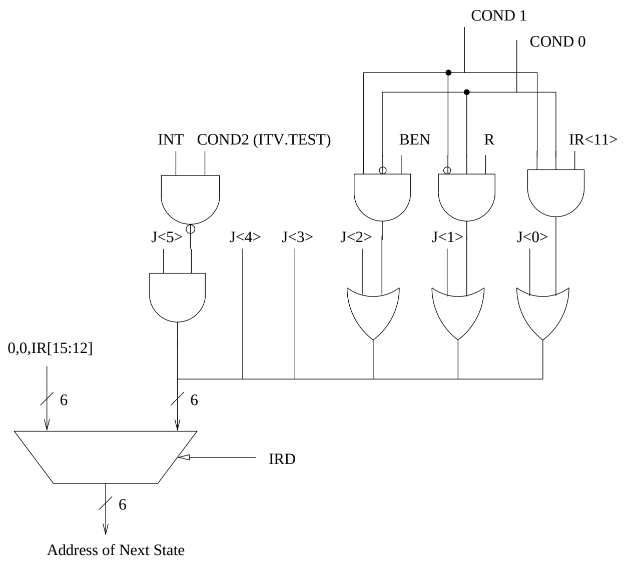
In state 4

Specify the logic function that computes ITV.



Part b. Augment the original microsequencer to take an exception in the presence of an illegal trapvector. Draw the augmented microsequencer below. Show clearly what additional control signals are needed. Identify (i.e., specify the bit encoding of) the state that will be used to initiate exception handling for ITV.

Additional control signal: COND2 (or "ITV.TEST"), will be 1 only in state 9



State encoding: 010101