CS120B – Homework #2

Given August 12, 2002. Due August 19, 2002 at the beginning of class.

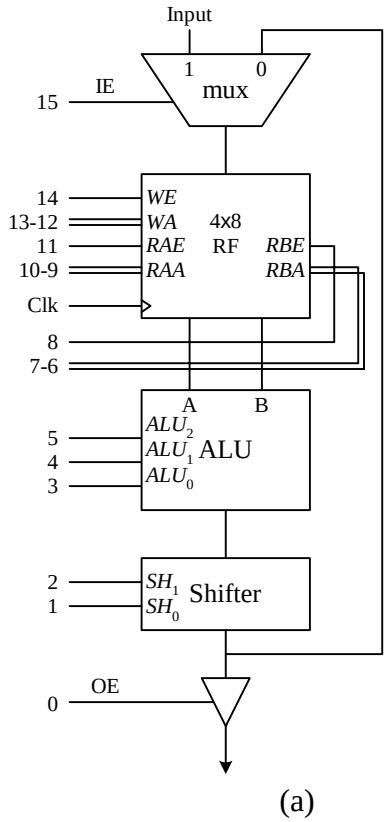
Use the datapath and functional operations shown in Figure 1 below to implement the following algorithm:

> Input a number *n* and output the factorial of *n*, i.e. *n*! Assume that *n* is small and no overflow error results from the calculations.

Use D flip-flops for the FSM.

You need to do the following:

a) Write the high-level pseudo-code to implement the algorithm. (2)
b) Convert the pseudo-code to control words. (2)
c) Draw the state-diagram for the control words. (2)
d) Derive the next-state/implementation table. (2)
e) Derive the excitation equations. (2)
f) Derive the output equations. (2)
g) Draw the complete FSM circuit. (2)
h) Draw the circuit(s) that generates signals from the datapath for the FSM. (2)



(a)

ALU_2	ALU_1	ALU_0	Operation
0	0	0	Pass through *A*
0	0	1	*A* AND *B*
0	1	0	*A* OR *B*
0	1	1	NOT *A*
1	0	0	*A* + *B*
1	0	1	*A* – *B*
1	1	0	*A* + 1
1	1	1	*A* – 1

(b)

SH_1	SH_0	Operation
0	0	Pass through
0	1	Shift left
1	0	Shift right
1	1	Rotate right

(c)

Figure 1. 8-bit datapath with register file: (a) circuit; (b) ALU operations; (c) Shifter operations.

Answer

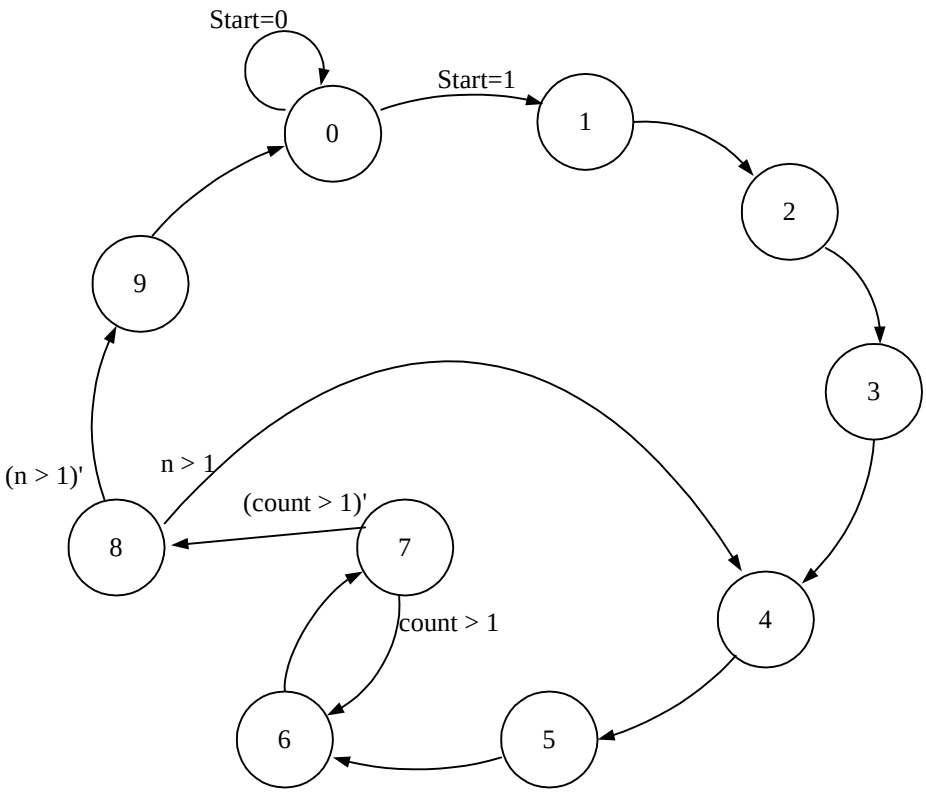
a)

```
1) cin >> n;
2) fac = 1;
   while (n > 1){
      //fac = fac * n;
4)    count = n;
5)    add = fac;
      while (count > 1){
6)       fac = fac + add;
7)       count = count - 1;
      }
8)    n = n - 1;
   }
9) cout << fac << endl;
```

b)

State	15	14	13 12	11	10 9	8	7 6	5 4 3	2 1	0	Comment
	IE	WE	WA	RAE	RAA	RBE	RBA	ALU	SH	OE	
1	1	1	00	0	××	0	××	×××	××	0	input *n*
2	0	1	01	0	00	0	00	101	00	0	*fac* = 0
3	0	1	01	1	01	0	××	110	00	0	*fac*++
4	0	1	10	1	00	0	××	000	00	0	*count* = *n*
5	0	1	11	1	01	0	××	000	00	0	*add* = *fac*
6	0	1	01	1	01	1	11	100	00	0	*fac* = *fac* + *add*
7	0	1	10	1	10	0	××	111	00	0	*count* = *count* – 1
8	0	1	00	1	00	0	××	111	00	0	*n* = *n* – 1
9	0	0	××	1	01	0	××	000	00	1	output *fac*

c)


Start=0
Start=1
0
1
2
3
4
5
6
7
8
9
n > 1
(n > 1)'
(count > 1)'
count > 1

d)

We can use one signal for both ($n > 1$) and ($count > 1$) since the comparator taps into the same point in the datapath and when that signal is needed, the value for the corresponding variable is available at that point.

	Next State			
Current State	*Start*, (*n_count*>1)			
$Q_3\ Q_2\ Q_1\ Q_0$	00	01	10	11
0000	0000	0000	0001	0001
0001	0010	0010	0010	0010
0010	0011	0011	0011	0011
0011	0100	0100	0100	0100
0100	0101	0101	0101	0101
0101	0110	0110	0110	0110
0110	1000	0111	1000	0111
0111	0110	0110	0110	0110
1000	1001	0100	1001	0100
1001	0000	0000	0000	0000

e) Note that the equations are not necessarily minimized. There can be other solutions.

$D_3 = Q_3'Q_2Q_1Q_0'\ Start'(n\_count>1)' + Q_3'Q_2Q_1Q_0'\ Start(n\_count>1)' + Q_3Q_2'Q_1'Q_0'\ Start'(n\_count>1)' + Q_3Q_2'Q_1'Q_0'\ Start(n\_count>1)'$
$= Q_3'Q_2Q_1Q_0'(n\_count>1)' + Q_3Q_2'Q_1'Q_0'\ (n\_count>1)'$

$D_2 = Q_3'Q_2Q_1' + Q_3'Q_2(n\_count>1) + Q_3'Q_2'Q_1Q_0 + Q_3'Q_2Q_1Q_0$
$= Q_3'Q_2Q_1' + Q_3'Q_2(n\_count>1) + Q_3'Q_1Q_0$

$D_1 = Q_3'Q_2'Q_1'Q_0 + Q_3'Q_2'Q_1Q_0' + Q_3'Q_2Q_1'Q_0 + Q_3'Q_2Q_1Q_0 + Q_3'Q_2Q_1Q_0'(n\_count>1)$
$= Q_3'Q_2'Q_1'Q_0 + Q_3'Q_2'Q_1Q_0' + Q_3'Q_2Q_0 + Q_3'Q_2Q_1Q_0'(n\_count>1)$

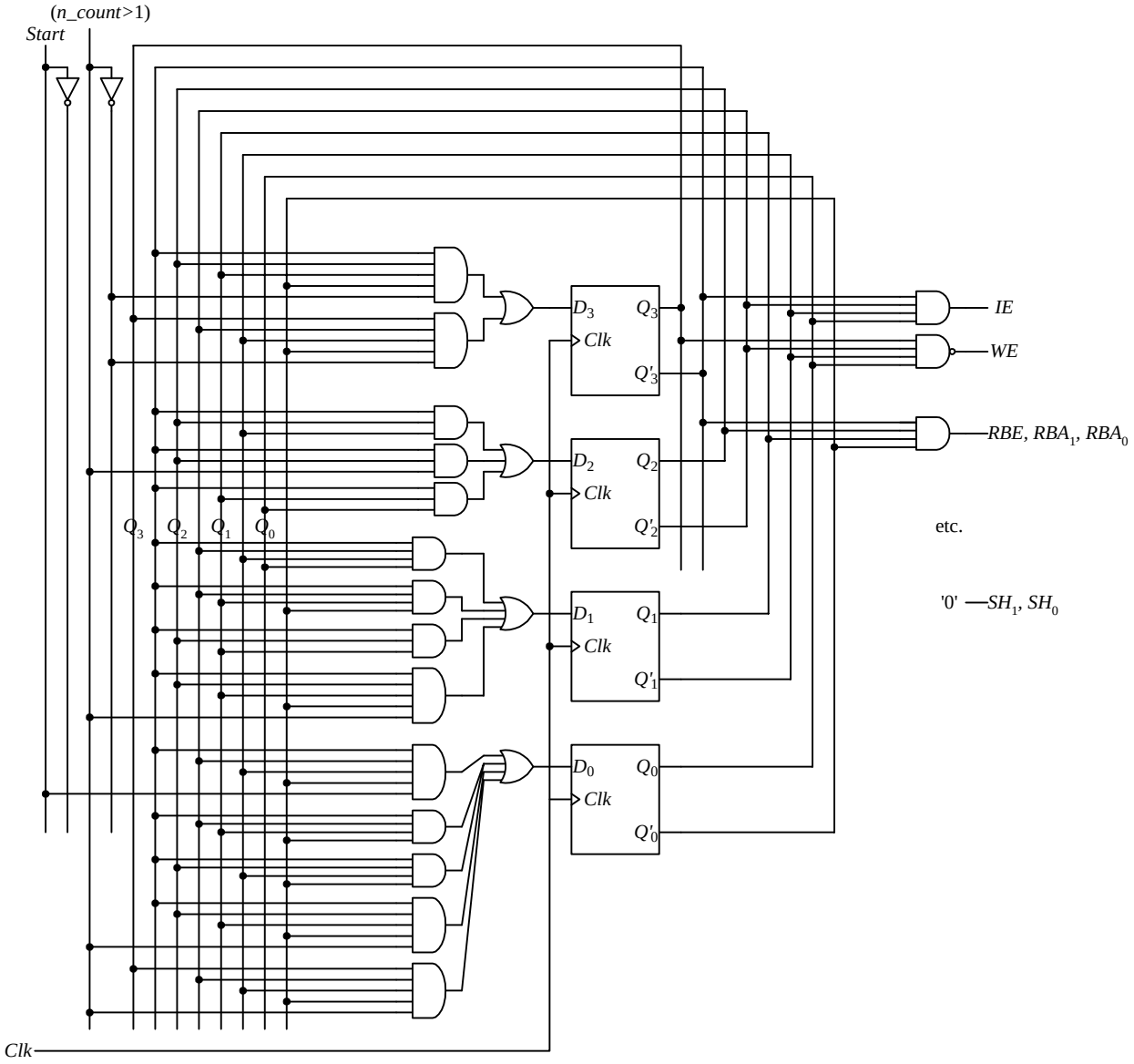
$D_0 = Q_3'Q_2'Q_1'Q_0'start + Q_3'Q_2'Q_1Q_0' + Q_3'Q_2Q_1'Q_0' + Q_3'Q_2Q_1Q_0'(n\_count>1) + Q_3Q_2'Q_1'Q_0'(n\_count>1)'$

f) The output equations are obtained from b). Note that the equations are not necessarily minimized. There can be other solutions.

$IE = Q_3'Q_2'Q_1'Q_0$
$WE = (Q_3Q_2'Q_1'Q_0)'$
$WA_1 = Q_3'Q_2Q_1'Q_0' + Q_3'Q_2Q_1'Q_0 + Q_3'Q_2Q_1Q_0$
$= Q_3'Q_2Q_1' + Q_3'Q_2Q_1Q_0$
$WA_0 = Q_3'Q_2'Q_1Q_0' + Q_3'Q_2'Q_1Q_0 + Q_3'Q_2Q_1'Q_0 + Q_3'Q_2Q_1Q_0'$
$= Q_3'Q_2'Q_1 + Q_3'Q_2Q_1'Q_0 + Q_3'Q_2Q_1Q_0'$
$RAE = (Q_3'Q_2'Q_1'Q_0 + Q_3'Q_2'Q_1Q_0')'$
$RAA_1 = Q_3'Q_2Q_1Q_0$
$RAA_0 = Q_3'Q_2'Q_1Q_0 + Q_3'Q_2Q_1'Q_0 + Q_3'Q_2Q_1Q_0' + Q_3Q_2'Q_1'Q_0$

$RBE = Q_3'Q_2Q_1Q_0'$
$RBA_1 = RBE$
$RBA_0 = RBE$
$ALU_2 = Q_3'Q_2'Q_1Q_0' + Q_3'Q_2'Q_1Q_0 + Q_3'Q_2Q_1Q_0' + Q_3'Q_2Q_1Q_0 + Q_3Q_2'Q_1'Q_0'$
$ALU_1 = Q_3'Q_2'Q_1Q_0 + Q_3'Q_2Q_1Q_0 + Q_3Q_2'Q_1'Q_0'$
$ALU_0 = Q_3'Q_2'Q_1Q_0' + Q_3'Q_2Q_1Q_0 + Q_3Q_2'Q_1'Q_0'$
$SH_1 = 0$
$SH_0 = 0$
$OE = Q_3Q_2'Q_1'Q_0$

g) I think I got the following correct ☺.



h)

For $n = 0$ or $n = 1$, all the bits except the least significant bit will be zero. So to test for ($n > 1$), we can simply use an OR gate with inputs from all the bits except the least significant bit.

We can connect this to the output of the shifter. You have to be careful that when the test is needed, you do have the value of n at that point in the state that the test is used. This is ok with our state diagram.