EECS 281: Sample Test 4 (4 pages) Due: Tuesday, November 23, 2004

Name: _____ Email: _____ Grade: _____ (100 points max)

1. (15 points) Write a multi-precision function which shifts left logical 1-bit and returns the shifted out bit:
(hint: other possible exam problems: rotate right, rotate left, AND/NAND/OR/NOR/XOR/XNOR, 1's or 2's complement add/subtract, shift left/right by n, compare equal/not equal/.../less than.)
(hint: see chalkboard lectures on multi-precision.)

```
#include <stdio.h>
#include <stdlib.h>
/*--------------------SOLUTION--------------------------------------------------------*/
/* f[0..n-1]=a[0..n-1] << 1; where "big" endian: a[1]=d15d14..d8; a[0]=d7..d0 */
int sll_char(char *f, char *a, int an) {
  int i, shiftin=0, shiftout=0;

  for(i=an-1; i>=0; i--) { /* DECEMENTING i: why? */
    shiftout=(a[i] & 0x80)?1:0; /* grab bit upper bit (i.e. d7) for next shift */
    a[i]=((a[i]<<1)&0xfe) | shiftin; /* shift a[i] left by 1-bit */
    shiftin=shiftout;
  }
  return shiftout;
}
/*-------------------------------------------------------------------------------*/
void printf_charx(char *msg, char *f, int fn) {
  int i;
  printf("%s=[", msg);
  for(i=0; i<fn; i++) { printf("%02x%s", f[i]&0xff, (i<fn-1)?",":"]"); }
  printf("\n");
}
int main(int argc, char *argv[]) {
  char wn=7, w[7]={ 0xA0, 0x00, 0x80, 0x00, 0x81, 0x22, 0x11 }; /* =w[0..6] */
  /* w is negative since bit 7 of W[0]=0xA0 is 1 and will change since it is a logical shift */
  printf_charx("w   ", w, wn);
  sll_char(w, w, wn); printf_charx("w<<1", w, wn);
  sll_char(w, w, wn); printf_charx("w<<2", w, wn);
  sll_char(w, w, wn); printf_charx("w<<3", w, wn);
  return 0;
}
/* sample output:
     w   =[a0,00,80,00,81,22,11]
     w<<1=[40,01,00,01,02,44,22]
     w<<2=[80,02,00,02,04,88,44]
     w<<3=[00,04,00,04,09,10,88]
*/
```

2. (15 points) Draw the Dataflow diagram for the following code. The char size is 8-bits.

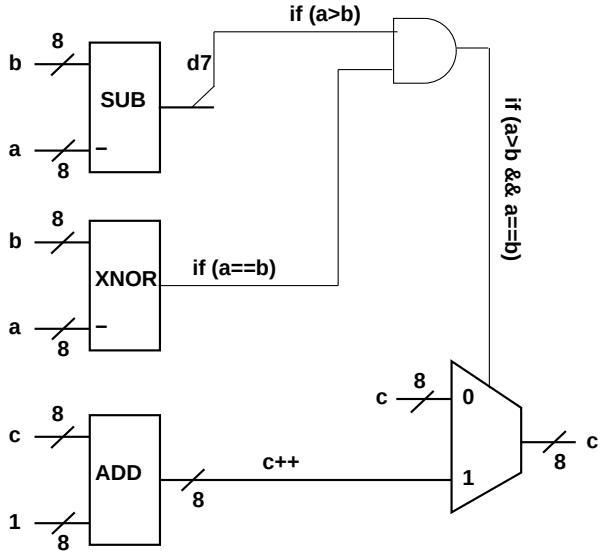
```
char a, b, c, d;
if ( a>b && a==3 ) { c++; }
```

Solution: rewrite as..

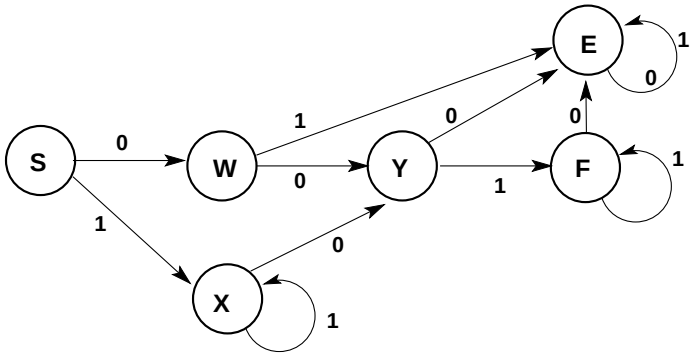
```
if ( a > b) { if (a == 3) { c++; } }
```

rewrite as..(why not use (a-b)&0x80 ?)

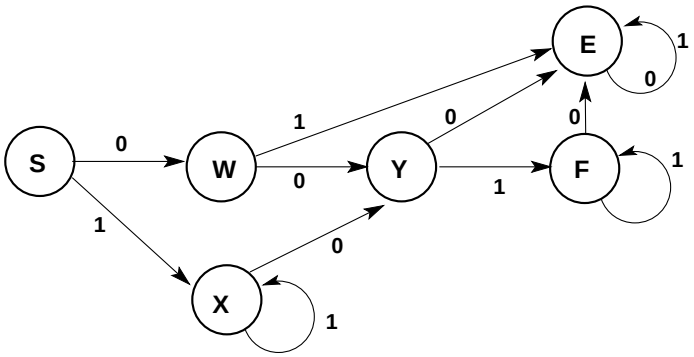
```
if ( (b-a)&0x80 ) { if (a == 3) { c++; }  }
```



3. (10 points) Draw the state transition diagram for a 1-bit input for the regular expression "(0|1+)01+". Use the following state symbols: S for start, F for final, E for Error and W, X, Y, Z, for all others.



4. (10 points) Given the state transition diagram for a 1-bit input give the transistion table.



Q	0	1
S	W	X
X	Y	X
Y	E	F
E	E	E
F	F	F

4. (10 points) Given the transistion table, give the state encodings., let S=000, W=001, X=010, Y=011, Z=100, E=101, F=110.

Q	0	1
S	W	X
X	Y	X
Y	E	F
E	E	E
F	F	F

$q_2q_1q_0i$ =QI	$q_2q_1q_0$ =Q
0000 =S0	001 =W
0001 =S1	010 =X
0010 =W0	011 =Y
0011 =W1	100 =E
0100 =X0	011 =Y
0101 =X1	010 =X
0110 =Y0	100 =E
0111 =Y1	101 =F
1000 =E0	100 =E
1001 =E1	100 =E
1010 =F0	100 =E
1011 =F1	101 =F
1100	XXX
1101	XXX
1110	XXX
1111	XXX

5. (20 points) Give the three excitation optimal k-map of the transistion encoded table (problem 4) and clear show circles. Treat each k-map independantly (i.e. do not do multi-output k-map optimization).

q_2	$\overline{c}\overline{d}$	$\overline{c}d$	cd	$c\overline{d}$
$\overline{a}\overline{b}$				
$\overline{a}b$				
ab				
$a\overline{b}$				

q_1	$\overline{c}\overline{d}$	$\overline{c}d$	cd	$c\overline{d}$
$\overline{a}\overline{b}$				
$\overline{a}b$				
ab				
$a\overline{b}$				

q_0	$\overline{c}\overline{d}$	$\overline{c}d$	cd	$c\overline{d}$
$\overline{a}\overline{b}$				
$\overline{a}b$				
ab				
$a\overline{b}$				

5. (10 points) Draw the Moore machine of problem 5 showing only the internals of q_0.

..see book, lecture notes, and recitation.

6. (10 points) Fill in the PLA of problem 5.
..see book, lecture notes, and recitation.

7. (10 points) Given the following gate logic diagram draw the timing diagram.
..see book, lectures, and recitation.

x1. (extra credit, 10 points) Write a multi-precision function which shifts right arithmetic 1-bit and returns the shifted out bit:

```
/* f[0..n-1]=a[0..n-1] << 1; where "big" endian: Example: an=2; a[0]=d15d14..d8; a[1]=d7..d0 */
/* Example: d15->d15 (keep signbit), d15->d14, d14->d13, ..., d8->d7, d7->d6, ..., d1->d0, d0->return */
int sra_char(char *f, char *a, int an) {
  int i, shiftin, shiftout=0;

  shiftin=(an>0)? (a[0]&0x80) : 0; /* signbit must be preserved */
  for(i=0; i<an; i++) { /* INCREMENTING i */
    shiftout=(a[i] & 0x01)?0x80:0; /* 0x01=grab bit lower bit (i.e. d8) for next shift into 0x80 (i.e. d7)
    a[i]=shiftin | ((a[i]>>1)&0x7f); /* shift a[i] shift right by 1-bit */
    shiftin=shiftout;
  }
  return shiftout;
}
int main(int argc, char *argv[]) {
  char wn=7, w[7]={ 0x81, 0x00, 0x80, 0x00, 0x81, 0x22, 0x11 }; /* =w[0..6] */
  /* w is negative since bit 7 of W[0]=0x80 is 1 and will not change */
  printf_charx("w   ", w, wn);
  sra_char(w, w, wn); printf_charx("w>>1", w, wn);
  sra_char(w, w, wn); printf_charx("w>>2", w, wn);
  sra_char(w, w, wn); printf_charx("w>>3", w, wn);

  return 0;
}
/* sample output using:
     w   =[81,00,80,00,81,22,11]
     w>>1=[c0,80,40,00,40,91,08]
     w>>2=[e0,40,20,00,20,48,84]
     w>>3=[f0,20,10,00,10,24,42]
*/
```

x2. (extra credit, 10 points) Write a multi-precision function which shifts left arithmetic 1-bit and returns the shifted out bit:

```
/* f[0..n-1]=a[0..n-1] << 1; where "big" endian: a[n-2]=d15d14..d8; a[n-1]=d7..d0 */
int sla_char(char *f, char *a, int an) {
  int i, shiftin, shiftout, signbit;

  if (an>0) { signbit=a[0]&0x80; } /* signbit must be preserved */
  shiftin=shiftout=0;
  for(i=an-1; i>=0; i--) { /* DECRMENTING i */
    shiftout=(a[i] & 0x80)?1:0; /* grab bit upper bit (i.e. d7) for next shift */
    a[i]=((a[i]<<1)&0xfe) | shiftin; /* shift a[i] left by 1-bit */
```

```
    shiftin=shiftout;
  }
  if (an>0) { a[0]|=signbit; } /* original signbit must be preserved */
  return shiftout;
}
int main(int argc, char *argv[]) {
  char wn=7, w[7]={ 0x81, 0x00, 0x80, 0x00, 0x81, 0x22, 0x11 }; /* =w[0..6] */
  /* w is negative since bit 7 of W[0]=0xA0 is 1 and will not change */
  printf_charx("w   ", w, wn);
  sla_char(w, w, wn); printf_charx("w<<1", w, wn);
  sla_char(w, w, wn); printf_charx("w<<2", w, wn);
  sla_char(w, w, wn); printf_charx("w<<3", w, wn);
  return 0;
}
/* sample output using:
  w   =[81,00,80,00,81,22,11]
  w<<1=[82,01,00,01,02,44,22]
  w<<2=[84,02,00,02,04,88,44]
  w<<3=[88,04,00,04,09,10,88]
*/
```

x3. (extra credit, 10 points) Write a multi-precision function which xor's a bit string and if the bit-string is non-zero then return 1 else returns 0 if zero string:

```
int xor_char(char *f, char *a, int n) {
```

x4. (extra credit, 10 points) Write a multi-precision alu function which operates on a bit string if alu=1 then XOR and if alu=2 then AND. If the bit-string is non-zero then return 1 else returns 0 if zero string:

```
int alu_char(char *f, char *a, int an, int alu) {
```