CMSI 2210

Assignment #4

Breea Toomey, Khalid Steward, Kimberly Kubo

1. Consider the function with three inputs (A,B,C) and two outputs (X,Y) that works like this:

A	B	C	X	Y
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

1.

   Design two logic circuits for this function, one using AND, OR and NOT gates only, and one using NAND gates only. You DO NOT HAVE to draw the circuit, but it might be helpful to do that to visualize and trace the logic. However, for this question you are only required to write the two formulas — one for computing X and one for computing Y. They can take the form of a logical equation such as

   X := A and B or such as Y := not-B and (A or C).

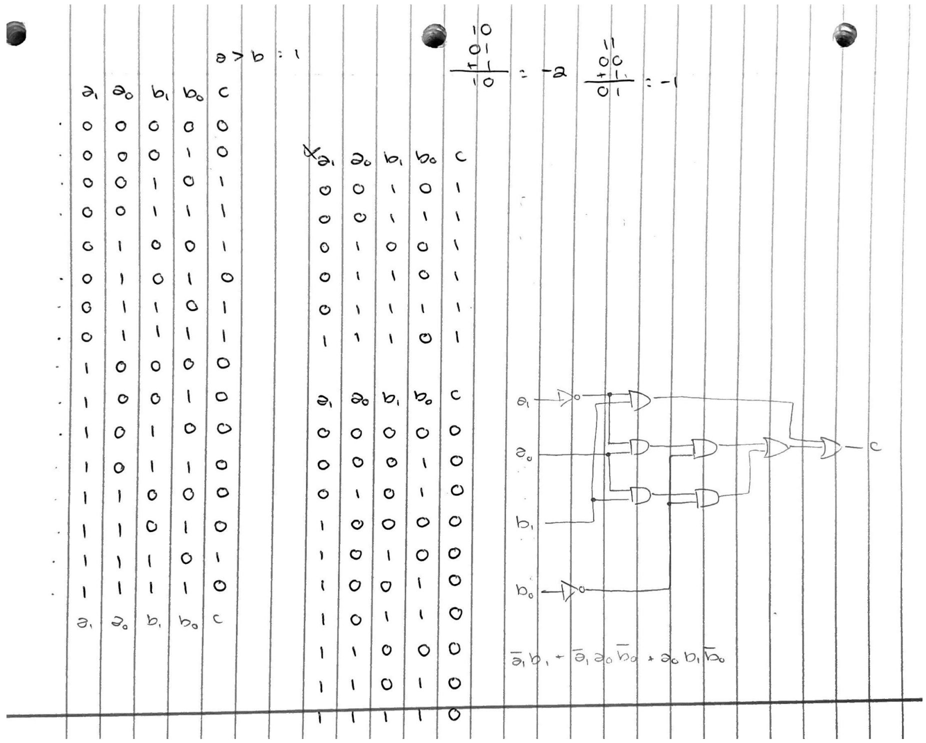
   X := A or (B and C)   Y := not-A or C
   X := (A nand A) nand (B nand C)   Y := A nand (C nand C)

2. Draw a logic circuit that compares two 2-bit signed numbers as follows. It should have four inputs a1, a0, b1, and b0. a1a0 is a 2-bit signed number (call it a) and b1b0 is a 2-bit signed number (call it b). The circuit has one output, c, which is 1 if a > b and 0 otherwise.

CMSI 2210

Assignment #4

Breea Toomey, Khalid Steward, Kimberly Kubo



(not-a1)b1 + (not-a1)a0(not-b0) + a0b1(not-b0) = c

3. Given a 32-bit register, write logic instructions to perform the following operations. For parts (c) and (f) assume an unsigned interpretation; for part (d) assume a signed interpretation.

    1. Clear all even numbered bits.
```
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
AND 10101010101010101010101010101010
```
    2. Set the last three bits.
```
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
OR  00000000000000000000000000000111
```
    3. Compute the remainder when divided by 8.
```
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
AND 00000000000000000000000000000111
```
    4. Make the value -1
```
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
OR  11111111111111111111111111111111
```
    5. Complement the two highest order bits
```
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
NAND 11000000000000000000000000000000
```
    6. Compute the largest multiple of 8 less than or equal to the value itself
```
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
AND 11111111111111111111111111111000
```

CMSI 2210

Assignment #4

Breea Toomey, Khalid Steward, Kimberly Kubo

4. For the sample single-accumulator computer discussed in class, write a complete assembly language program in the stanley/penguin language that sends the values 0 through 255 out to port 0x8. NOTE: the machine code for this will be written in the next problem.

```
           JMP        start      ; jump over the data area
current:   0                     ; store the current value
next:      1                     ; store the next value
limit:      255                  ; compute values 0 to 255
start:     LOAD       current    ; load current into accumulator
           WRITE      0x8        ; write to port 0x8
           ADD        next       ; add next to current
           STORE      current    ; store accumulator (current + 1) in
                                   current
           SUB         limit      ; if not yet past limit, keep going
           JLZ        start      ; if not past limit, jump to
beginning
end:       JUMP       end        ; stops the program
```

5. Translate your assembly language program in the previous problem to machine language.

```
C0000004
00000000
00000001
000000FF
00000000
30000008
40000001
10000000
50000003
E0000004
C000000A
```

6. For the sample single-accumulator computer discussed in class, write a complete assembly language program in the stanley/penguin language that computes a greatest common divisor. Assume the two inputs are read in from port 0x100. Write the result to port 0x200. You do not need to write machine code for this problem.

```
           JMP        start      ; jump over the data area
a:         0                     ; store the a value
b:         0                     ; store the b value
start:     READ       0x100      ; read into accumulator
           JZ         gcd        ; if a is 0, jump to gcd
           WRITE      b          ; write to b
           READ       0x100      ; read into accumulator
           MOD        a          ; b%a
```

CMSI 2210

Assignment #4

Breea Toomey, Khalid Steward, Kimberly Kubo

```
            WRITE      0x200       ; write to port
            JMP        start       ; jump to start
gcd:        READ       0x100       ; read into accumulator
            WRITE      0x200       ; write to port
end:        JMP        end         ; stops the program
```

7. For the sample single-accumulator computer discussed in class, give a code fragment, in assembly language of the stanley/penguin language, that swaps the accumulator and memory address 0x30AA. You do not need to write machine code for this problem.

```
temp:       0                      ; store the temp value
start:      WRITE      temp        ; write accumulator to temp
            LOAD       0x30AA      ; load memory address into
     accumulator
            READ       temp        ; read from temp into accumulator
            STORE      0x30AA      ; store accumulator in 0x30AA
```

8. For the sample single-accumulator computer discussed in class, give a code fragment, in assembly language of the stanley/penguin language that has the effect of jumping to the code at address 0x837BBE1 if the value in the accumulator is greater than or equal to 0. You do not need to write machine code for this problem.

```
            JMP        start
            LOAD       0x837BBE1  ; load address
            JLZ        start
            JZ         start
```

9. Part 1 of 2: Explain, at a high-level, what the following sequence of instructions does. In other words, suppose a programmer has stored data in r8 and r9. After executing these instructions, what does the programmer notice about the data?

   xor r8, r9
   xor r9, r8
   xor r8, r9

   This swaps the values in registers r8 and r9.

   Part 2 of 2: Also state as briefly as possible why that effect happens.
   The output of r8 xor r9 is stored in r8. Then the output of r9 xor r8 produces the original r8 value, stored in r9. The output of r8 xor r9 produces the original r9 value, stored in r8