Assignment # 4 - CSI 2111 (Solutions)

Q1. You are asked to design a 3-bit synchronous counter that starts at 1 and cycles through prime numbers (i.e., number divisible by 1 or itself). The counter counts up when the mode M = 1 and down when M = 0.

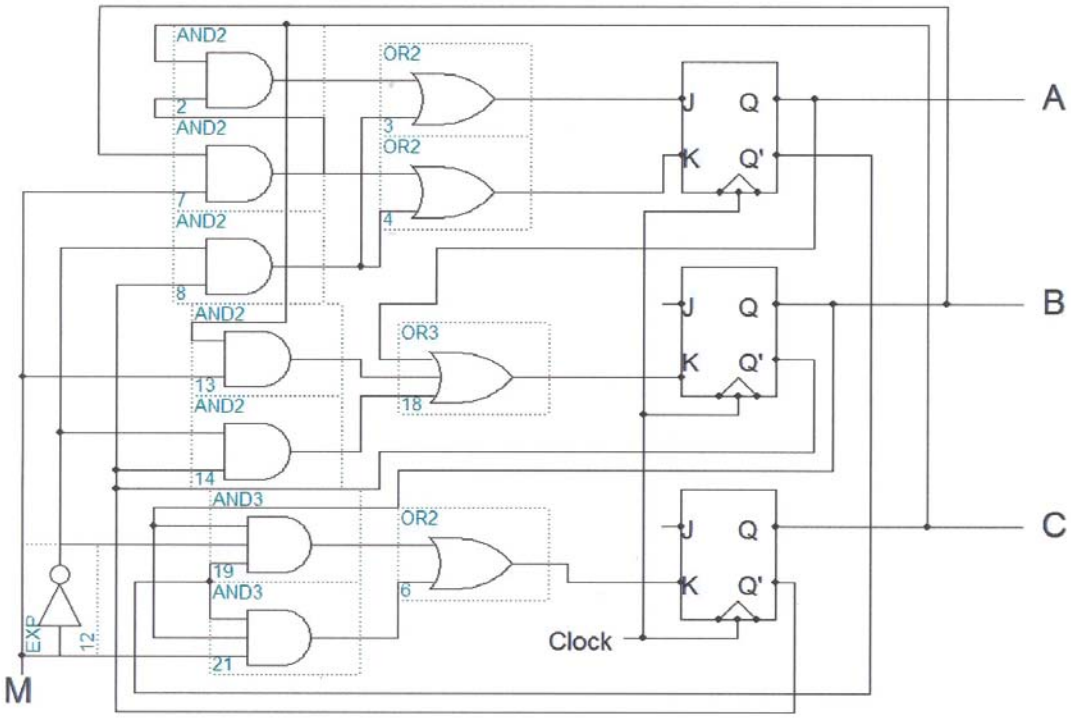
a) Design the counter using JK flipflops, transitioning on positive edge. Show all the steps. (15)

Present State	Next State	FF Input	FF Input	FF Input
M ABC	A'B'C'	J_A K_A	J_B K_B	Jc Kc
0 0 0 1	1 1 1	1 x	1 x	x 0
0 0 1 0	0 0 1	0 x	x 1	1 x
0 0 1 1	0 1 0	0 x	x 0	x 1
0 1 0 1	0 1 1	x 1	1 x	x 0
0 1 1 1	1 0 1	x 0	x 1	x 0
1 0 0 1	0 1 0	0 x	1 x	x 1
1 0 1 0	0 1 1	0 x	x 0	1 x
1 0 1 1	1 0 1	1 x	x 1	x 0
1 1 0 1	1 1 1	x 0	1 x	x 0
1 1 1 1	0 0 1	x 1	x 1	x 0

$J_A = M'B' + MBC$; $K_A = M'B' + MB$

$J_B = 1$; $K_B = A + MC + M'C'$

$J_C = 1$; $K_C = M'A'B + MA'B'$



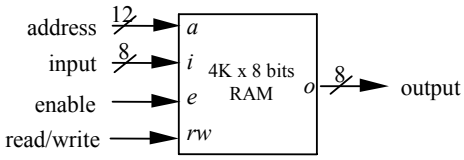
(Figure courtesey of Jamie Parsons & Brent Martel)

b) Is the counter self-correcting? Justify your answer. (5)

Yes, it is self-correcting.
For M = 0:
0 -> 7; 4 -> 3; 6 -> 5

For M = 1:
0 -> 3; 4 -> 7; 6 -> 1

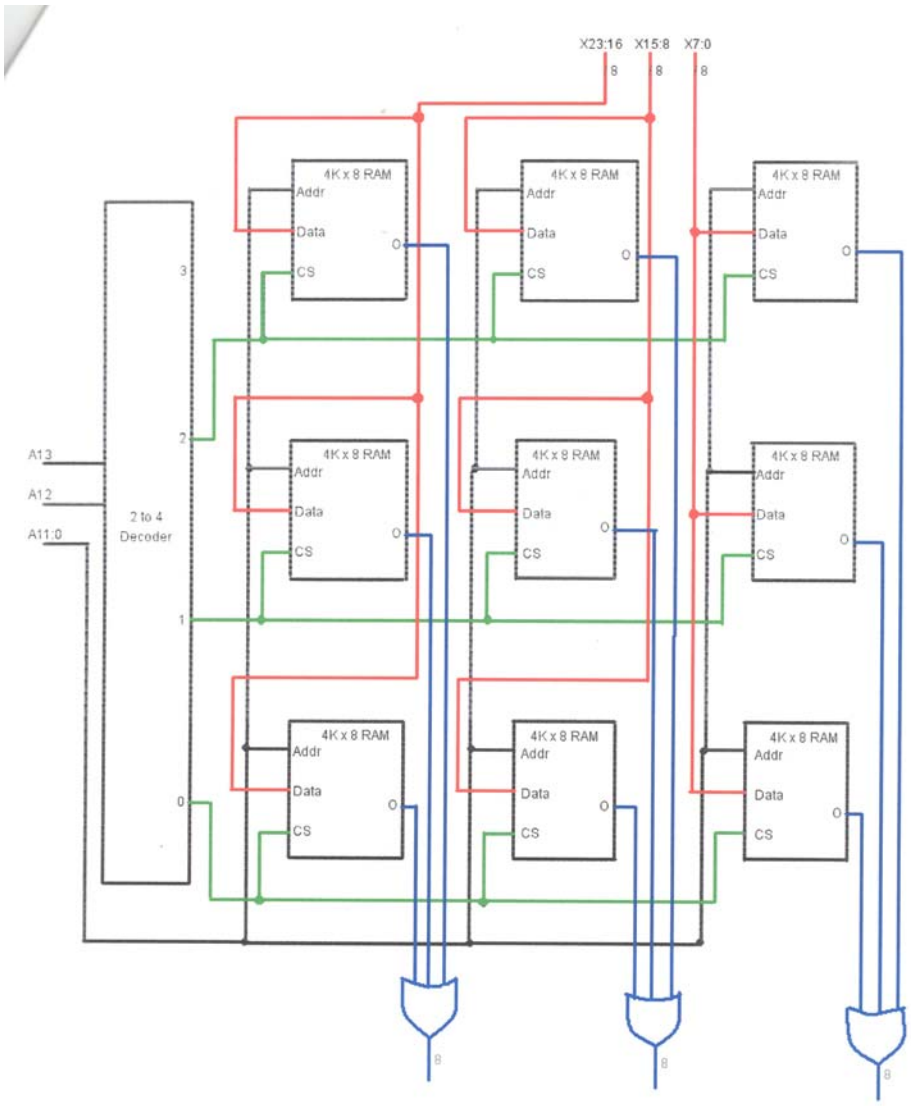
Q2. Using the following 4K x 8 bits RAM chip (the *enable* input is also called *chip select*):



a) Describe the circuits necessary by filling out the blanks as shown below for the realization of a 1M x 64 bits RAM. (5)

2048 4K x 8 bits RAM chips
64 OR gates with 256 inputs
One 8 to 256 decoder with *enable* input

b) Implement a 12K x 24 bits RAM. Clearly show the inputs/outputs of your RAM blocks and decoders, and show connections clearly (with different colors). For the OR gates, you can simply draw some of them and explain how the others would be connected. (10)



(Figure courtesey of Brett Michol)

Q3. Given the contents of the following memory addresses and registers (available before each of the following **MC68000** instructions), determine for each instruction, *when possible*, the contents of the modified memory addresses and registers. (15)

Some registers:

```
[A0] = $0000 2002
[A1] = $0000 2006
[D0] = $0000 0010
[D1] = $ABCD 5432
[D2] = $0001 0002
[D3] = $0000 2002
```

Memory ([address] = content):

```
[$002000] = $B020
[$002002] = $BBBB
[$002004] = $8000
[$002006] = $1111
[$002008] = $7000
[$00200A] = $1234
```

a) `MOVE.L  A0,D3`
b) `MOVE.L  #80,D1`
c) `MOVEA.W  D1,A1`
d) `MOVE.W  (A1),A1`
e) `MOVEA.B  (A1),D0`
i) `MOVE.L  $FFFC(A1), D0`
j) `MOVE.B  $4(A1),D1`
k) `ADDA.W  -(A1),A0`
l) `ADD.B  D1,D1`
m) `MULU  D0,D2`

```
f)  MOVE.L    D1,(A1)            n)  DIVS      D0,D2
g)  MOVE.W    -(A1),(A0)+        o)  ROR.W   $#5, D2
h)  MOVE.B    D2,-(A1)
```

a) [D3] = $0000 2002
b) [D1] = $0000 0050
c) [A1] = $0000 5432
d) Prohibited because <d> = A1
e) Prohibited because (.B is not supported and <d> = D0)
f) [$002006] = $ABCD; [$002008] = $5432
g) [A0] = $0000 2004; [$002002] = $8000; [A1] = $0000 2004
h) [A1] = $0000 2005; [$002002] = $8002
i) [D0] = $BBBB 8000
j) [D1] = $ABCD 5412 (A1 is unchanged)
k) [A1] = $0000 2004; [A0] = $0000 A002
l) [D1] = $ABCD 5464
m) [D2] = $0000 0020
n) [D2] = $0002 1000
o) [D2] = $0001 1000 (invalid instruction if $#5 is treated as an error)

Q4. Consider an array of **N** bytes located in memory at the address **B**. Write a program in Sim68K assembly language ("maximum.68a") which will find the largest and the second-largest value in this array and store them in the locations **First** and **Second**, respectively. Your program should also display the values of **First** and **Second** in the end. Test your program using an array of at least 10 elements. (Suggestion: You can put **N** and **B** at the end of your program after the `HLT.B`) (15)

```
; ******************************************************
; *** CSI2111 Assignment-4 Question-4                ***
; *** max2.68a                                       ***
; *** Sim68 program to the maximum and second maximum ***
; *** value in an array of bytes                     ***
; ******************************************************
;

; Main program
      MOVEA.W @B, A0        ; move starting address of array to A0
      CLR.L D0              ;
LABEL @Loop                 ;
      CMP.B D0, @N          ; Are we at the end of the array?
      BEQ.W @Done           ;
      CMP.B @First,(A0)     ; compare array element with First
      BGE.W @Found1         ; if greater, found new max
      CMP.B @Second,(A0)    ; else compare with Second
      BGE.W @Found2         ; if greater, found new 2nd-max
LABEL @Continue             ;
      ADDQ.B #$1, A0        ; go to next element
      ADDQ.B #$1, D0        ; increment counter
      BRA.W @Loop           ;

LABEL @Found1               ;
      MOVE.B @First, @Second ;
      MOVE.B (A0), @First   ;
      BRA.W @Continue       ;

LABEL @Found2               ;
      MOVE.B (A0), @Second  ;
```

```
      BRA.W @Continue        ;


LABEL @Done                  ;
; display First and Second.
      DSP.B D0               ; display the size of array
      DSP.B @First           ; display maximum value
      DSP.B @Second          ; display second maximum
      HLT.B                  ;

DEF.B @First, #$00           ;  Initialize to zero
DEF.B @Second, #$00          ;         ''
DEF.B @N, #$0C               ;  N = 12 (size of array)
DEF.L @B,   #$90080706       ;  Array starts here
DEF.L @B4,  #$05040302       ;
DEF.W @B8,  #$12345678       ;
```

Q5. Here is a binary **Sim68k** program. Decode it and provide the corresponding **Sim68k** assembly language program. You can check your answer by translating your assembly program… (15)

```
/ ;    Assembler Language       OpCode  Oper1   Oper2  /
/ ;                            MSB LSB MSB LSB MSL LSB /
/ ;------------------------------------------------- /
/ ; Input the number N  /
    /    INP.B  @N             ; / $E0 $60 $00 $2C
    /    MOVEQ.L #$2, D1         / $CC $21

    / LABEL @Loop      /
    /    CLR.L D0               / $3C $00
    /    MOVE.B @N, D0          / $C1 $60 $00 $2C
    /    CMP.B  D1, D0          / $81 $10
    /    BLE.W  @done           / $BA $60 $00 $26
    /    DIVS.L D1, D0          / $2D $10
    /    EOR.W  D0, D0          / $5B $00
    /    TST.L  D0              / $8C $00
    /    BEQ.W @Composite       / $A2 $60 $00 $22
    /    ADDQ.B #$1, D1         / $08 $11
    /    BRA.W @Loop            / $92 $60 $00 $06

    / LABEL @Composite   /
    /    MOVEQ.B #$1, @Result   / $C8 $16 $00 $2D

    / LABEL @done        /
    /    DSP.B  @Result         / $E8 $60 $00 $2D
    /    HLT.B                  / $F8 $00

    / DEF.B @N, #$00           / $00
    / DEF.B @Result, #$00      / $00
```

(b) Can you guess what this program does? (Bonus 5)

The program inputs a number and checks whether it is prime or composite. If the input number is prime then the Result is zero, else the Result is one.