Remember to show ALL work here and in EVERY problem on this exam.

[10\%] 1. Circuit Analysis

What is the logic equation for X in the given circuit? Do not simplify or transform it into an SOP or POS form. Leave the logic expression as it is after analysis. Also, draw the intermediate expression at the input to each gate.

- Notation reminder: A(H) is the same as A.H
- Boolean expression answers must be in lexical order, i.e., IA before A, A before B, etc.

EQUATION: \( X = \overline{(A \cdot B)} + (A \cdot \overline{B + C}) \cdot D + E \)

\[
\begin{align*}
\text{A(H)} & \quad \overline{B_H} \\
\text{B(L)} & \quad \overline{D_H} \\
\text{C(H)} & \quad \overline{E_H} \\
\text{D(L)} & \quad \overline{E.H} \\
\text{X(H)} & \\
\end{align*}
\]
2. **Circuit Synthesis**

Draw a mixed-logic circuit diagram (with the minimum number of gates) to directly implement the below equation. All inputs and the output can be of any activation-level desired. Be sure to specify the desired activation levels. Do not simplify this equation. You may only use gates available on 74HC27 chips (shown). Use as many 74HC27 chips as you need, but use the minimum number required to solve this problem.

\[ F = A \cdot B \cdot C + C \cdot \overline{D} \cdot \overline{E} + A \cdot \overline{B} \cdot \overline{D} \]

---

**Note:**

\[ x \rightarrow \overline{x} \equiv x \rightarrow x \overline{x} \]

\[ x + x + x = \overline{x} \]
3. **Implementation of an ASM chart using J-K flip-flops**

(a) Given the above ASM chart, complete the following block diagram of its implementation using the minimum number of J-K flip-flops: (2%)

- Determine how many J-K flip-flops that are needed.
- Draw in all the inputs and outputs of the combinational circuit.
- Make all necessary connections to complete the block diagram (between the combinational circuit and the flip-flops).

J-K characteristic table:

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q</th>
<th>Q+</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(Inputs) | **Combinatorial circuit** | (Outputs) |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Q</td>
<td>J K</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>A B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C D E F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A D G</td>
</tr>
</tbody>
</table>

(Draw flip-flops here.)
3. (continued) (ASM chart is repeated here for your convenience/)

(b) Finish the implementation of the ASM by determining the **minimum sum-of-products (MSOP)** logic expressions for the following signals: (10%)

\[
\begin{align*}
J &= Q + \overline{X} \\
K &= 1 \\
A &= 1 \\
B &= \overline{Q} \\
C &= \overline{Q}.X = Q + \overline{X} \\
D &= Q + X \\
E &= \overline{Q}.X \\
F &= \overline{Q}.X \\
G &= Q
\end{align*}
\]

(If necessary, use the bottom/back of the previous page to do your work.)

\[
\begin{array}{cccccccc}
Q = 0: \text{State 1} & | & Q & X & Q^+ & J & K & A & B & C & D & E & F & G \\
0 & 0 & 0 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
4. Given the following program segment, EQU statements, and contents of memory locations:

```
ORG 0
X = $000C
A = $9B
B = $3C
A = $D2

LDX #X
LDAA #A;
LDAB #B;
SUM BA
"next instruction
a, b, c, d, e, or f"

Data0 EQU $0026
Data1 EQU $0032
Data2 EQU $0037
Data3 EQU $0038
Data4 EQU $0050
Data5 EQU $006F
Data6 EQU $5000
Data7 EQU $73AB
Data8 EQU $73AC
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0026</td>
<td>$66</td>
</tr>
<tr>
<td>$0032</td>
<td>$BB</td>
</tr>
<tr>
<td>$0037</td>
<td>$74</td>
</tr>
<tr>
<td>$0038</td>
<td>$EC</td>
</tr>
<tr>
<td>$0050</td>
<td>$C3</td>
</tr>
<tr>
<td>$006F</td>
<td>$AB</td>
</tr>
<tr>
<td>$5000</td>
<td>$3E</td>
</tr>
<tr>
<td>$73AB</td>
<td>$AA</td>
</tr>
<tr>
<td>$73AC</td>
<td>$E2</td>
</tr>
</tbody>
</table>

a) Assume the 4 instructions in the above program segment have already been executed and the "next instruction" (i.e., 5th instruction) is **LDAA 50**. Hand-assemble the LDAA instruction and fill in the blanks (including EA and register A) using **HEX**.

Note: EA is the "effective address", the actual address in memory where the data is loaded from in a "load" instruction (like LDAA, LDX, etc.), or where the data is stored in a "store" instruction (like STAA). If no memory is accessed, write "none" for EA.

```
ADDRESS | INSTRUCTION | HEX ADDRESS | HEX VALUE
---------|-------------|-------------|-------------
$0008    | LDAA 50     | $0008       | 04 32 (decimal 50)
EA = $0032
A = $BB
```

Repeat the same problem if the "next instruction" is the instruction in "b", "c", "d", "e", or "f", again assuming instructions 1 through 4 have been executed.

b) $0008    | LDX Data7  | $0008       | OA
EA = $73AB
X = $E2AA (little endian)

```
0008 | 0032 | 000C 26
Value stored = $D2
```

c) $0008    | STAA 38,X  | $0008       | 10 26 (38 decimal)
EA = $0032
Value stored = $D2

```
0008 | 0009 | 000A | 000B
```

d) $0008    | LDY #Data7 | $0008       | 09
EA = $0009
Y = $73AB

```
0008 | 0009 | 000A | 000B
```

e) $0008    | BN $EF     | $0008       | 22
EA = none ($0009 OK)
PC after this instruction = $00EF (D2 is negative)

```
0008 | 0009 | 000A | 000B
```

f) $0008    | BEQ $EF    | $0008       | 20
EA = none ($0009 OK)
PC after this instruction = $000A

(D2 is not equal to 0)
5. EEPROM and SRAM

You are given as many of the following EEPROM and SRAM devices that you need for the problem below:

- EEPROM
  - A11:0
  - D7:0
  - -CE

- SRAM
  - A12:0
  - D3:0
  - R/W
  - -CE

Note: D3:0

You are to implement a 16Kx8 memory module, containing 8Kx8 SRAM, starting in location 0, followed immediately by 8Kx8 EEPROM.

The 16K X 8 memory module should have an "R/W" input and a "-CE" input.

For ease of grading, draw the 8Kx8 SRAM circuit here (i.e., show the devices & connections):

Draw the 8Kx8 EEPROM circuit here (i.e., show the devices & connections):
6. **Program Execution** (Use the G-CPU information in the back of the test.)

Given the following program in EPROM memory, fill out the following cycle table that illustrates its execution:

<table>
<thead>
<tr>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>07</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
</tr>
</tbody>
</table>

Using the G-CPU Controller ASM & Block Diagram (at the end of this test), complete the cycle table below. Use as many rows as you need.

<table>
<thead>
<tr>
<th>Cycle#</th>
<th>R/W</th>
<th>Addr Sel:1:0</th>
<th>PC (Hex)</th>
<th>MAR (Hex)</th>
<th>A15:0 (Hex)</th>
<th>Data (Hex)</th>
<th>IR (Hex)</th>
<th>A (Hex)</th>
<th>B (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>00</td>
<td>0000</td>
<td>0400</td>
<td>0000 01</td>
<td>06</td>
<td>32</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>00</td>
<td>0000</td>
<td>0000</td>
<td>0000 01</td>
<td>01</td>
<td>32</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>00</td>
<td>0001</td>
<td>0001 07</td>
<td>01 16</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>00</td>
<td>0001</td>
<td>0001 07</td>
<td>07 16</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>00</td>
<td>0002</td>
<td>0002 01</td>
<td>07 16</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>00</td>
<td>0003</td>
<td>0003 07</td>
<td>07 16</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>01</td>
<td>0701</td>
<td>0701 0701</td>
<td>16 07 16</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>MAR connected to Addr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 = memory read
0 = memory write
7. **GCPU Assembly Programming** (Use the G-CPU instruction set in back of this test.)

Write an assembly language program using **only** the G-CPU instructions.

- There are $25$ bytes already stored in memory starting in Location $1000$.
- Your program should go through the $25$ bytes and count the number of "non-zero" bytes (i.e., bytes that are not equal to $00$).
- At the end of the program, Memory Location $00FF$ should contain the count of non-zero bytes.
- To increase your chances of partial credit, comment your program.

*Important restriction:* You have to use R00A as the "loop counter".

```
LDAA #$25
STA A $OFFE; init loop count
LDX #$1000; init index
LDAA #$00  ; init byte count
STA A $OFFF
AGAIN: LDA A 0,X ; load next byte
BZ NEXT
LDA A $OFFF  ; increment byte count
LDAB #$01
SML BA
STA A $OFFF

NEXT: INX
LDA A $OFFE  ; decrement loop count
LDAB #$FF
SML BA
STA A $OFFF
BNE AGAIN
FIN: BRA FIN
```
8. **Controller (ASM) design for the G-CPU.** (Use the G-CPU block diagram and ASM charts in the back of the test.)

Using the signal names shown in controller G-GPU block diagram in the back of the text, complete the ASM chart (on the next page) to implement the following 3 instructions (AND_BA, LDAA $addr$, and BEQ $addrL$), including the completion of State A and State B.

**Notes:**

1. In each state and conditional output, specify **only the signals that should be true.**
2. However, you should use the notation: **MSA=01, MSB=10, MSC=000.**
3. The default actions for each state should be specified to **“hold” REGA and REGB** and **OUT = REGA.**
4. For your convenience, the required values for MSA, MSB, and MSC are given below:

---

**Table 1: Input source MUXs for Registers A and B.**

<table>
<thead>
<tr>
<th>MSA1/MSB1</th>
<th>MSA0/MSB0</th>
<th>Bus Selected as Input to REGA/REGB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>INPUT Bus</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>REGA Bus</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>REGB Bus</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OUTPUT Bus</td>
</tr>
</tbody>
</table>

**Table 2: ALU function selection MUX. (MUX C)**

<table>
<thead>
<tr>
<th>MSC2:0</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>REGA Bus to OUTPUT Bus</td>
</tr>
<tr>
<td>001</td>
<td>REGB Bus to OUTPUT Bus</td>
</tr>
<tr>
<td>010</td>
<td>complement of REGA Bus to OUTPUT Bus</td>
</tr>
<tr>
<td>011</td>
<td>bit wise AND REGA/REGB Bus to OUTPUT Bus</td>
</tr>
<tr>
<td>100</td>
<td>bit wise OR REGA/REGB Bus to OUTPUT Bus</td>
</tr>
<tr>
<td>101</td>
<td>sum of REGA Bus &amp; REGB Bus to OUTPUT Bus</td>
</tr>
<tr>
<td>110</td>
<td>shift REGA Bus left one bit to OUTPUT Bus</td>
</tr>
<tr>
<td>111</td>
<td>shift REGA Bus right one bit to OUTPUT Bus</td>
</tr>
</tbody>
</table>

**without sign extension**

Put the solution on the next page.
8 (continued): Put solution here:

State A

RI-W (to addr)
IR: LD
MSA = 01
MSB = 10

(Note ADD_SEL = 00 to connect PC to Address)

State B

PC_INV

(Note that $addrL denotes the lower byte of the branch address)

$16
AND_BA

MSA = 11
MSB = 10
AND, MSC = 011

State A

$04
LDAA $addr

IR5:0

20
BEQ $addrL

MSA = 01
MSB = 10

$Null

MAR_LD (L)
MSA = 01
MSB = 10
PC_INV

Low byte of address to MAR (Low)

High byte of address to MAR (High)

0
Z flag

State A

MAR_LD (U)
MSA = 01
MSB = 10
PC_INV

State A

ADD_CE = 01
R1-W
MSA = 00
MSB = 01

State A

State A

$addrL loaded into PC (low byte)