You are given a 1 KHz clock for the following problem. Design a state machine that creates an output (OUT.H) based on three input variables: A.L, B.L and C.L. Assume ONE and ONLY one of the three INPUT variables will be TRUE at any given time.

If A = T, output a 500 Hz 50% Duty Cycle square wave. Else If B = T, output a 333.3 Hz 33.3% Duty Cycle square wave. Else if C = T output a 250 Hz 25% Duty Cycle square wave. Recall: Frequency = 1/Period and Duty Cycle = 100% * (Time High/Period). Best Design = Most pts.

1. Draw the Flow Chart below for the State Machine. Don't forget to number your states! (4 pt.)
2. Draw the **Functional Block Diagram** for the system assuming that logic gates & D flip-flops will be used in the implementation. *Label all signals.* (3 pt.)

![Block Diagram]

3. Fill in the **Next State Table** below and use only the Present and Next States required in your design (7 pt.)

Note: You may assume that we have created a **reset circuit** to always **start** in the first state 'State 0'.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*also ok*
4. Create a Logic Equation (SOP is OK!) & circuit for OUT.H using inputs A.L, B.L, C.L and Qx:Q0. (3 pt.)

\[ \text{Out} = A \overline{Q}_1 Q_0 + B \overline{Q}_1 \overline{Q}_0 + C \overline{Q}_1 Q_0 \]

![](image1)

5. Suppose we desired OUT.H to run at twice its current frequency, to do this we would need to create a clock that is twice the original 1KHz clock. Show how to create a 2 KHz clock from the original 1 KHz clock.

Hint: Use the asynchronous clear and/or set mechanism in a D flip-flop. (3 pt.)

![](image2)
5. A vector is stored in 4Kx8 RAM at address $1000$. It consists of $37_{10}$ signed numbers that are each two bytes in length. Write a GCPU program to count the number of positive numbers in the vector. Write your final sum to the address immediately after the vector stored in RAM. Use X as a pointer to the data and use address $1FFF$ for your loop counter. (10 pt.)

```
+1 Ldaa #37 ($25) ; Loop Counter
Staa $1FFF
Ldx #$1000 i ; data ptr
Top: Ldaa 1,X i ; get data

+1 Bn skip i ; check if positive
Ldaa #$4A ?? \text{Positive counter}
Ldab #1
Sumba
Staa #$4A

Skip: Inx 2 i ; inc ptr to next word
Inx

Ldaa $1FFF i ; Loop Counter = loop counter - 1
Ldab #$FF ??
Sumba
Staa $1FFF
Bne top +1

DONE
```

```
37_{10} = 37 \text{words} = 74 \text{bytes}
74 = 4A \text{Hex} \quad 0 - 73 = 74 \text{bytes}
37_{10} = 25 \text{Hex}
+1 Ldaa #0 i ; Zero Pos. Count
Staa $4A

$4A$ wrong -0.5
```

Page 4
For the next set of questions, consult the G-CPU code in Appendix A.

6. What is the effective address for the LDY #$00 instruction?  
   \[ \text{Hex } (1 \text{ pt.}) \]

7. What is the effective address for the STAA 0,X instruction the 1st time it is executed?  
   \[ \text{Hex } (1 \text{ pt.}) \]

8. How many RAM memory locations are modified by this program?  
   \[ 13 \text{ Decimal } (2 \text{ pt.}) \]

9. What are the values in RAM when the program has completed? (2 pt.)  
   \[ \begin{array}{c}
   \text{1000} \\
   \text{0x1D}
   \end{array} \]

10. In Appendix A, we are now using a SLOW ROM to fetch code. This ROM takes 2 cycles per memory fetch, i.e., 2 clock periods instead of 1 to fetch a byte out of memory. Fill in the cycle table below for the 1st loop pass for the lines of code shown below assuming that 2 cycles instead of one will be required to fetch a memory byte out of ROM. RAM still has 1 cycle R/W to/from memory. Label simply “NEW” for new states in the controller ASM below. (9 pt.)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Controller State</th>
<th>Addr Bus</th>
<th>Data Bus</th>
<th>IR</th>
<th>R/W</th>
<th>A Reg</th>
<th>Address Mux Sel1:0</th>
<th>Dev Outputting To Data Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>000F</td>
<td>x X</td>
<td>0C</td>
<td>1</td>
<td>03</td>
<td>O pc</td>
<td>Rom</td>
</tr>
<tr>
<td>2</td>
<td>New</td>
<td>000F</td>
<td>18</td>
<td>DC</td>
<td>1</td>
<td>03</td>
<td>O pc</td>
<td>Rom</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>000F</td>
<td>18</td>
<td>18</td>
<td>1</td>
<td>03</td>
<td>O pc</td>
<td>Rom</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0010</td>
<td>x X</td>
<td>18</td>
<td>1</td>
<td>03</td>
<td>0 pc</td>
<td>Rom</td>
</tr>
<tr>
<td>5</td>
<td>New</td>
<td>0010</td>
<td>10</td>
<td>18</td>
<td>1</td>
<td>03</td>
<td>0 pc</td>
<td>Rom</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0010</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>8B</td>
<td>0 pc</td>
<td>Rom</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
<td>0011</td>
<td>x X</td>
<td>10</td>
<td>1</td>
<td>8B</td>
<td>Pc</td>
<td>Rom</td>
</tr>
<tr>
<td>8</td>
<td>New</td>
<td>0011</td>
<td>00</td>
<td>10</td>
<td>1</td>
<td>8B</td>
<td>Pc</td>
<td>Rom</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>1000</td>
<td>8B</td>
<td>10</td>
<td>0</td>
<td>8B</td>
<td>Z MAR</td>
<td>CPU</td>
</tr>
</tbody>
</table>

;From Appendix A, show execution of the 1st pass of this code
11. Using only decoders, flip-flops, muxes, full adders and other elementary digital components, show the new hardware and modifications to the controller required to create a JMP ADDR instruction where ADDR is a 16 bit operand that corresponds to the jump address. Use BUS labeling when possible. (5 pt.)

Controller

IR, LD

INC_PC

with "wait" state

IR, LD

IR

JMP Addr 32H

PC, INC, T, LD

D7:0 T, LD

D7:0 T15:0

PC, INC

PC, LD

state 0

INCPC15:0

PC15:0

T15:0

PC15:0

PC, INC

PC, LD

(16) 4:1 Muxes, (16) D FFs

(16) 2:1 Muxes, (16) D FFs

HW signals show up in Asm + 0.5

PC, LD is new combined signal!

INC PC 15:0 = same as before!

PC15:0 $0001

16

16

(16) Full Adders

INCPC15:0

Page Score =
Appendix A. G-CPU Code for Problems 6 - 10:

Note1: 8K ROM starting address $0$ and 8K RAM starting at address $1000$.

Note2: ROM is a slow device that requires TWO cycles to read in a value from it.

```
ORG $0
LDAB #$F3
STAB $2FFF
LDY #$0
LDX #$1000
LDAA 0,Y
LDAB #$88
OR_BA
STAA 0,X
INX
INY
LDAA $2FFF
LDAB #1
SUM_BA
BNE T1
BEQ END1
```

```
\[ B = F3 \]
\[ 2FFF \]
\[ F3 \]

\[ X = 1000 \]
\[ A = 03 \]
\[ B = 88 \]
\[ 1000 \]
\[ 1011 \]
\[ 8 \]
\[ B = A \]
\[ 88 \]
\[ 100 \]
\[ 1002 \]
\[ F3 \]

\[ A = F3 \]
\[ B = 1 \]
\[ A = F4 \]

Note: Use the space below to hand assemble instructions as needed in 6 - 10:

```
Addr Data
0 03 LDAB #
1 F3
2 07
3 FF STAB addr
4 2F
5 09 LDY #
6 00
7 00
8 08
9 00 LDX #
A 10
```

```
B 0C LDX #
C 00
D 03 LD aa 0,Y
E 88
F 18 OR_BA
10 10 STAA 0,X
11 00
12
```