

**EER-018**  
**INTRODUCTION to DIGITAL COMPUTERS**  
**LABORATORY NO. 9**  
**MC68HC05J1A INTERFACING**  
**TRAFFIC LIGHT CONTROLLER**

**OBJECTIVES:**

1. Learn more of the M68HC05 instruction subset.
2. To complete a practical sequential controller design from specification to implementation.
3. To gain experience with MC68HC05J1A interfacing.

**EQUIPMENT :**

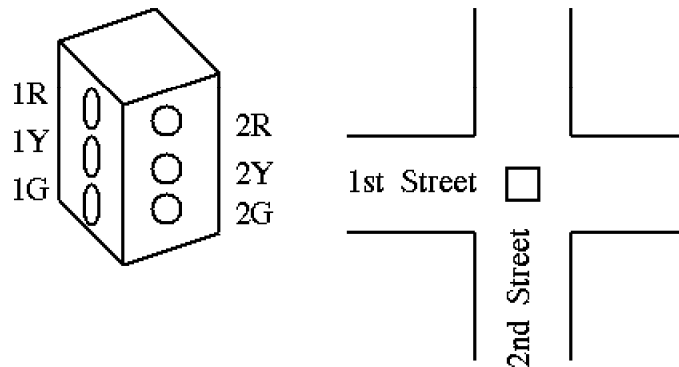
Desktop Computer  
RAPID Microcontroller Development System  
M68HC705J1CS kit and protoboard  
MM74HC244 buffer, wires, LEDs, resistors.

**REFERENCES:**

1. M68705J1A In-Circuit Simulator. User's Manual (copy in lab)
2. MC68HC705J1A Technical Data (copy in lab)

**INTRODUCTION:**

The scenario for this lab is to control a simple traffic light. We will assume that the intersecting streets are labeled 1 and 2 as shown below, and there are corresponding Red, Yellow and Green lights for each street.



A simple flow diagram is shown to illustrate the desired behavior of the traffic light. Each state describes the lights that are on and the transitions are labeled with "long delay" or "short delay" to indicate how long the light values should remain before transitioning.

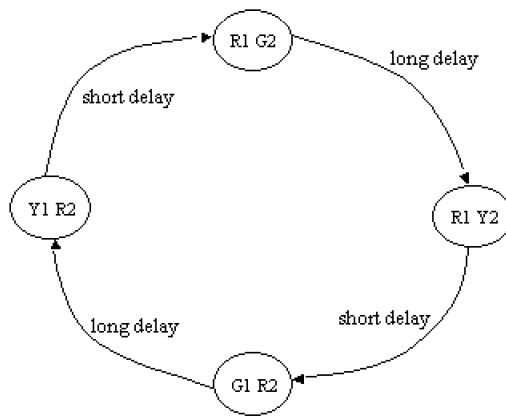


Figure 1. Flow Diagram for Traffic Light Behavior

To implement the traffic lights, we will use 6 bits of PortA as outputs to drive 6 LEDs. These outputs must be buffered since the microcontroller does not have enough current capacity to drive the LEDs directly. The diagram below shows how the 74HC244 buffer will be connected. Notice that you must connect the two enable inputs, 1G' and 2G' to ground to enable the buffers. You must also connect Vcc and GND of the 74HC244 chip. The buffer pin diagram is shown on the right.

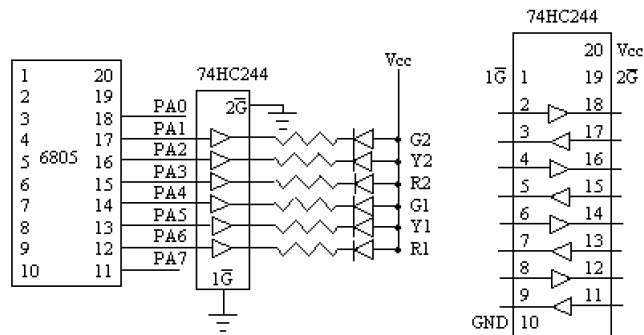


Figure 2. Port A Interface Hardware

Notice that the outputs of the 6805 will cause the LEDs to turn on only if the output is LOW.

### PRELAB

The following program will be used to implement the traffic light controller. Please analyze this program and answer the questions following the program.

```

temp1 equ $C0
tempo equ $C1
org rom
start: lda #$7E
      sta ddra
loop:  lda R1G2
      sta porta
  
```

```

        lda R1G2+1
        bsr delay
        lda R1Y2
        sta porta
        lda R1Y2+1
        bsr delay
        lda G1R2
        sta porta
        lda G1R2+1
        bsr delay
        lda Y1R2
        sta porta
        lda Y1R2+1
        bsr delay
        bra loop
delay:  sta tempo           ; the delay subrouting delays .2s * A
oloop:  lda #$FF           ; where A is the value of the A register
        sta tempi          ; when the subroutine is called
iloop:  dec tempi
        bne iloop
        dec tempo
        bne oloop
        rts

*DATA
R1G2    fcb %10111101
        fcb $0A
R1Y2    fcb %10111011
        fcb $05
G1R2    fcb %111100111
        fcb $0A
Y1R2    fcb %11010111
        fcb $05
reset:  org $7FE
        fdb start

```

1. When this program is assembled, the labels in the program have the following addresses (all in hex):

```

temp_i  00C0
temp_o  00C1
start   0300
loop    0304
delay   032E
oloop   0330
iloop   0334
r1g2    033D
r1y2    033F
g1r2    0341
y1r2    0343

```

The program consists of a main program, a subroutine, and some data. Identify the starting address of each section.

2. The data of the program is organized in sets of two bytes. One byte is the light value to be sent to porta, and the other value is the amount of delay for that light value. Identify the addresses of all the delay values.

3. The assembler is being used to calculate the address of the delay values, for example for the instruction:

```
lda R1G2+1
```

the assembler will create machine code as follows:

What will the machine code be for the next delay value lda instruction?

```
lda R1Y2+1
```

4. The main loop of the program consists of 4 copies of the following four instructions:

```
lda light_value_data
sta porta
lda delay_value_data
bsr delay
```

If more states were added to the flow diagram of the problem, then more copies would be needed, as well as more data entries. Describe another way to write this program so that only data items would be needed to expand it. Hint: use indexed addressing with the initial data address as the offset.

### PROCEDURE

1. Using the M68HC705JICS kit and protoboard, connect the 6805 extension pod to the 74HC244 buffer and LEDs as illustrated in figure 2.
2. Assemble and run the program "answer.asm" that is found in the EE18/Lab9 folder. Verify that it behaves as specified in figure 1.
3. Modify the program so that it goes through the two additional "safety" states as illustrated in the flow diagram below. The best solution will allow future modifications to be made to the flow diagram without changing the main part of the program.

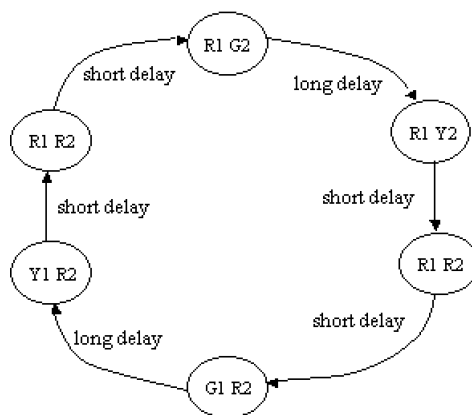


Figure 3. Enhanced Flow Diagram for Traffic Light

Submit your prelab and completed program to the lab instructor for evaluation. No report for this lab.

CT 11/2003