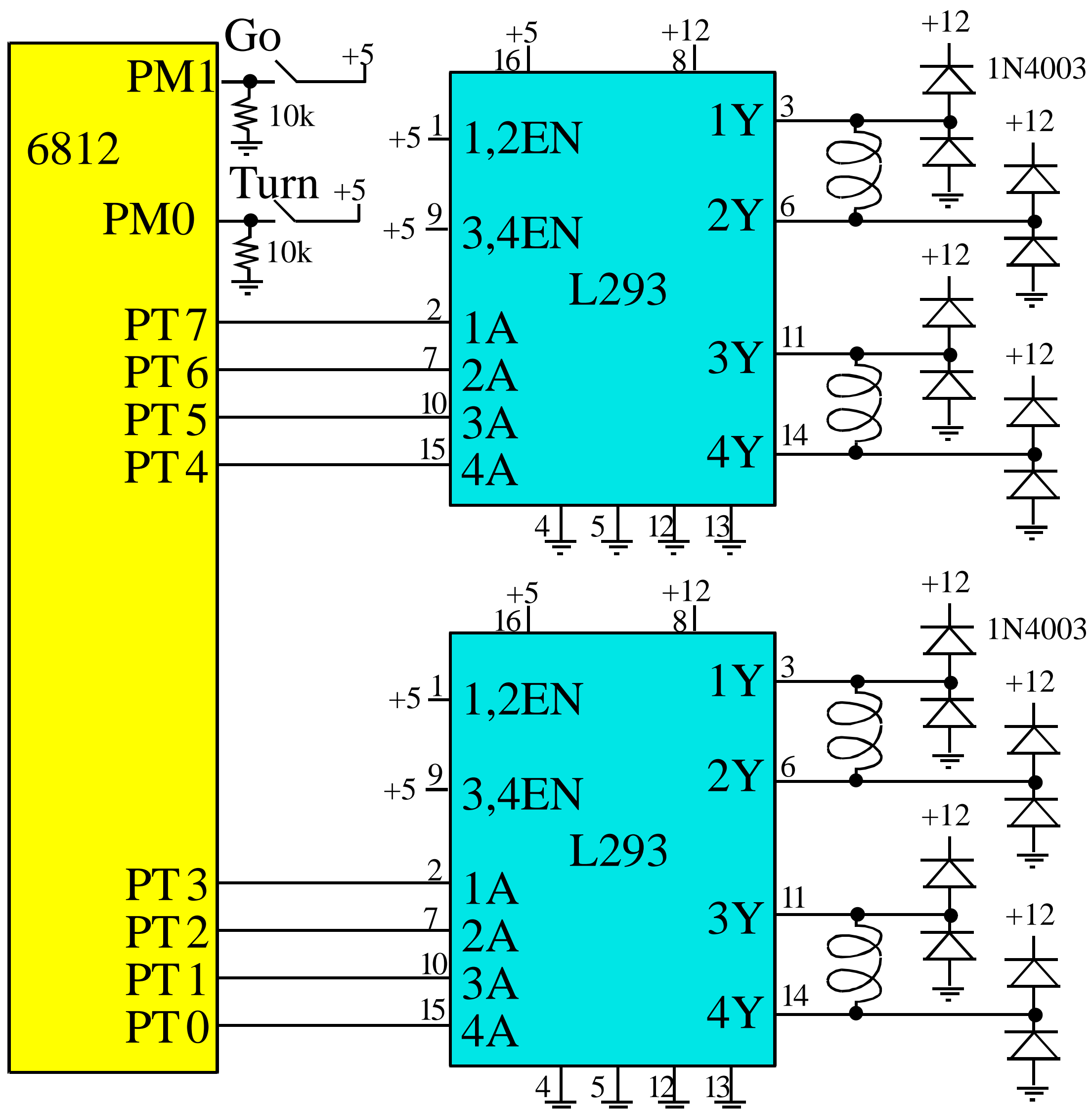
8.7. Finite state machines with statically-allocated linked structures

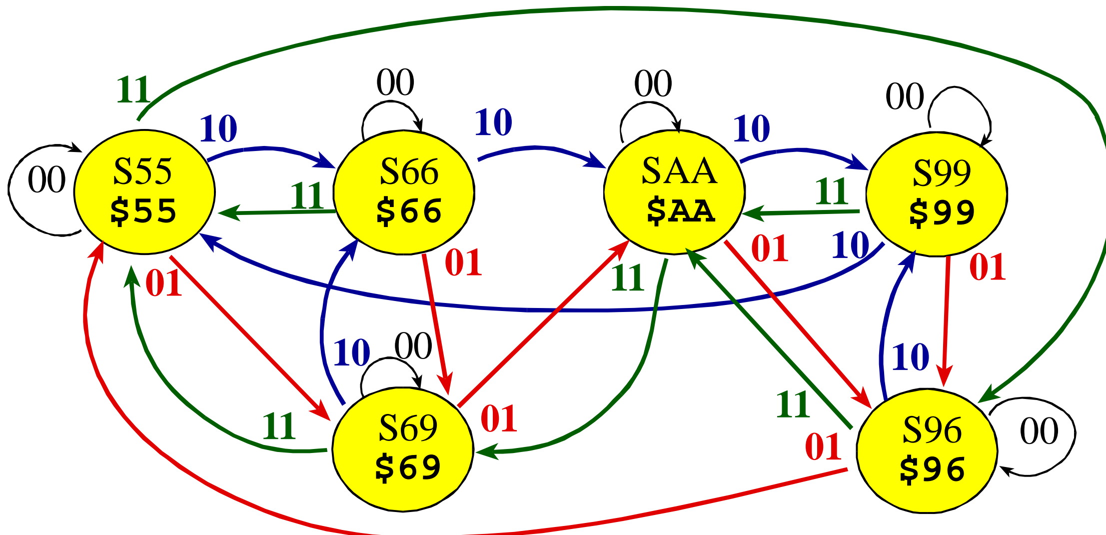
Stepper motor controller

Inputs: Go and Turn
Outputs: two 4-wire bipolar stepper motors



Bipolar stepper motor interface using an L293 driver

```
// Port M bits 1-0 are inputs
// =00 Stop
// =10 Go    (55,66,AA,99)
// =01 RTurn(55,69,AA,96)
// =11 LTurn(55,96,AA,69)
// Port H bits 7-0 are outputs to steppers
```




```
const struct State {
  unsigned char out;        // command
  const struct State *next[4];};
typedef const struct State StateType;
StateType *Pt;
#define S55 &fsm[0]
#define S66 &fsm[1]
#define SAA &fsm[2]
#define S99 &fsm[3]
#define S69 &fsm[4]
#define S96 &fsm[5]
StateType fsm[6]={
  {0x55,{S55,S69,S66,S96}},  // S55
  {0x66,{S66,S69,SAA,S69}},  // S66
```

```
  {0xAA,{SAA,S99,S99,S69}},  // SAA
  {0x99,{S99,S69,S55,SAA}},  // S99
  {0x69,{S69,SAA,S66,S55}},  // S69
  {0x96,{S96,S55,S99,SAA}}}; // S96
```

This stepper motor FSM has two input signals four outputs.

```
void main(void){
unsigned char Input;
  Timer_Init();
  DDRT = 0x0ff;
  DDRM = 0;
  Pt = S55;    /* initial state */
  while(1){    /* never quit */
    PTM = Pt->out;    /* stepper drivers */
    Timer_Wait(2000); /* 0.25ms wait */
    Input = PTM&0x03;
    Pt = Pt->next[Input];
  }
}
```

Write in assembly
** RAM variables*

```
    org  $3800
Pt  rmb  2  pointer to current state
```

** ROM constants*

```
     org  $4000
out  equ  0     8-bit output
next equ  1     4 pointers to next state
S55  fcb $55                   8-bit output
```

```
      fdb S55,S69,S66,S96   next for each in
S66   fcb $66
      fdb S66,S69,SAA,S69
SAA   fcb $AA
      fdb SAA,S99,S99,S69
S99   fcb $99
      fdb S99,S69,S55,SAA
S69   fcb $69
      fdb S69,SAA,S66,S55
S96   fcb $96
      fdb S96,S55,S99,SAA
```

** ROM program*

```
Main lds  #$4000
     bsr  Timer_Init  ; activate TCNT
     movb #$FF,DDRT   ; PT7-PT0 stepper
     movb #$00,DDRM   ; PM1=CCW, PM0=CW
     movw #S55,Pt     ; initial state
loop ldx  Pt
     movb out,X,PTT   ; step motor
     ldd  #2000
     bsr  Timer_Wait  ; wait 0.25ms
     ldaa PTM         ; read inputs
     anda #$03        ; just CCW,CW
     lsla             ; 0,2,4,6
     leax next,X      ; list of pointers
     ldx  A,X         ; next depends on in
     stx  Pt
     bra  loop
```

8.7.3. Traffic light controller

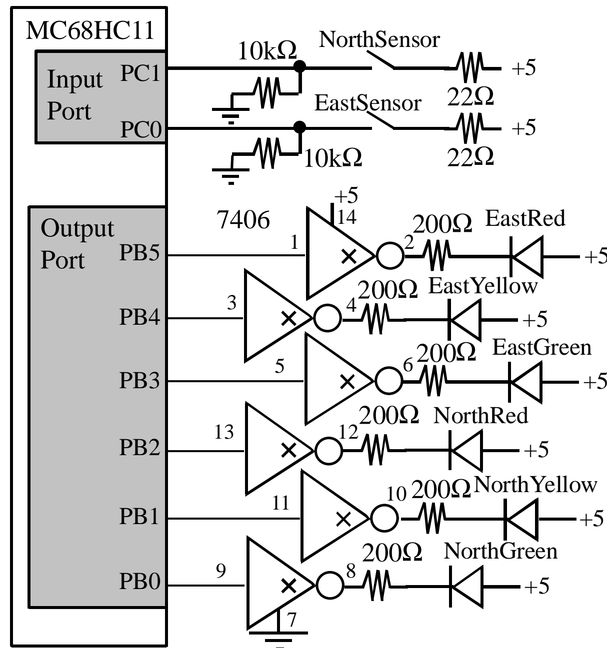


Figure 8.19. A simulated traffic intersection interfaced to a Motorola MC68HC11.

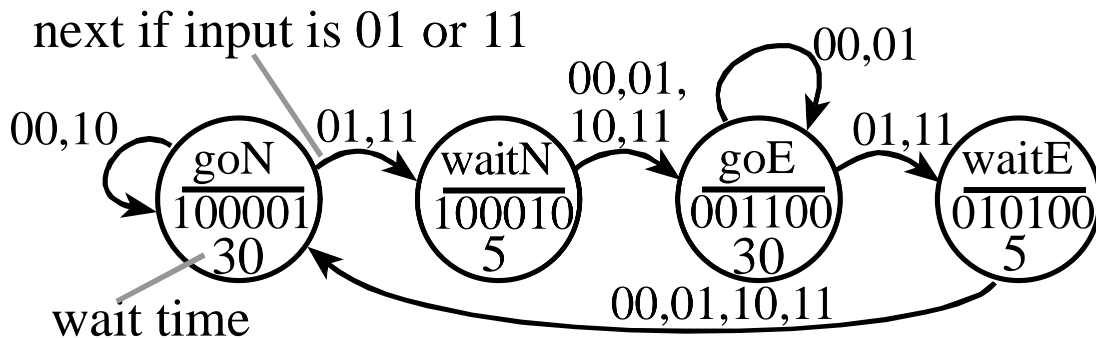


Figure 8.20. This Moore FSM controls traffic in our intersection.

```
/* Port C bits 1,0 are sensor inputs,
   Port B bits 5-0 are LED outputs */
const struct State {
  unsigned char Out; //Output to Port B
```

```
  unsigned short Time; //sec to wait
  const struct State *Next[4];}; // Next if
input=00,01,10,11*/
typedef const struct State StateType;
#define goN   &fsm[0]
#define waitN &fsm[1]
#define goE   &fsm[2]
#define waitE &fsm[3]
StateType fsm[4]={
 {0x21,30,{goN,waitN,goN,waitN}}, // goN
 {0x22, 5,{goE,goE,goE,goE}},     // waitN
 {0x0C,30,{goE,goE,waitE,waitE}}, // goE
 {0x14, 5,{goN,goN,goN,goN}}};    // waitE
StateType *Pt;  // Current State
void main(void){
  unsigned char Input;
  DDRB = 0xFF;  // outputs to traffic light
  DDRC = 0xFC;  // PC1 car on north
  Pt = goN;     // PC0 car on east
  while(1){
    PORTB = Pt->Out;        // Perform output
    Wait1sec(Pt->Time);     // Time to wait
    Input = PORTC&0x03;     // 00,01,10,11
    Pt = Pt->Next[Input]; // next
  }
}
```

Program 8.49. Traffic light controller.

To add more complexity
(e.g., put a red/red state after each yellow state),
we simply increase the size of the `fsm[]` structure

define the `Out`, `Time`, and `Next` pointers

To add more output signals
(e.g., walk light),
use more of `Out` field.
could increase the precision of the `Out` field

To add two input lines
(e.g., walk button),
increase the size of `Next[8]`.
size = 2**(number of inputs)

Lab 8.2. Traffic Light Controller

This lab has these major objectives:

- The usage of linked list data structures;
- Create a segmented software system;
- an input-directed traffic light controller.

Description

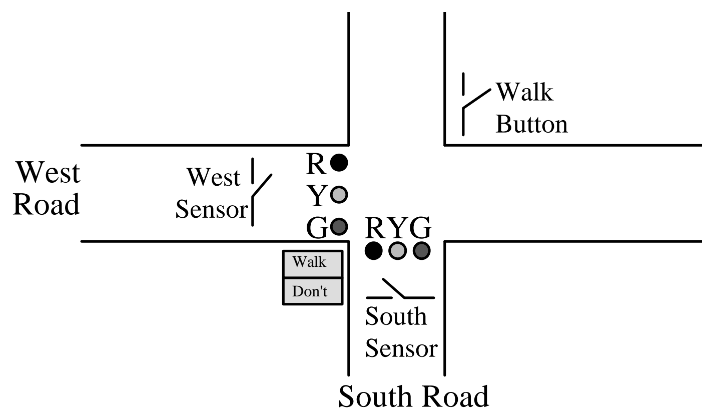


Figure 8.28. Traffic Light Intersection.

Part a) Build an I/O system with the appropriate names and colors on the lights and switches.
Part b) Design a finite state machine that implements a good traffic light system. Include a graphical picture of your finite state machine showing the various states, inputs, outputs, wait times and transitions.
Part c) Write the assembly code that implements the traffic light control system. There is no single, “best” way to implement your traffic light. However, your scheme must be segmented into RAM/EEPROM/ROM and you must use a linked-list data structure. There should be a 1-1 mapping from the FSM states and the linked list elements. A “good” solution has about 10 to 20 states in the finite state machine, and provides for input dependence. Your software will be graded on the Number of Accidents, Maximum Wait Time, and Average Wait Time. For example, if there are no cars currently on the roads and a new car approaches a red light, then the lights should change quickly to allow this car to proceed. On the other hand, if there are many cars going North/South and one car approaches East/West, it may not be efficient to quickly change the lights.

Typically in real applications using an embedded system, we put the executable instructions into the ROM. We then ask Motorola to make us 1000’s of microcomputers with our executable program in ROM. We then program the finite state machine linked list data structure into the nonvolatile EEPROM. A good implementation will allow minor changes to the finite machine (adding states, modifying times, removing states, moving transition arrows, changing the initial state) simply by changing the linked list controller (easy to change EEPROM in a real microcomputer), without changing the executable instructions (can not change the ROM of a real microcomputer).

Obviously, if we add another input sensor or output light, it may be necessary to update the executable part of the software and re-assemble. *Hint: can you change the initial state of your FSM without modifying the ROM?*