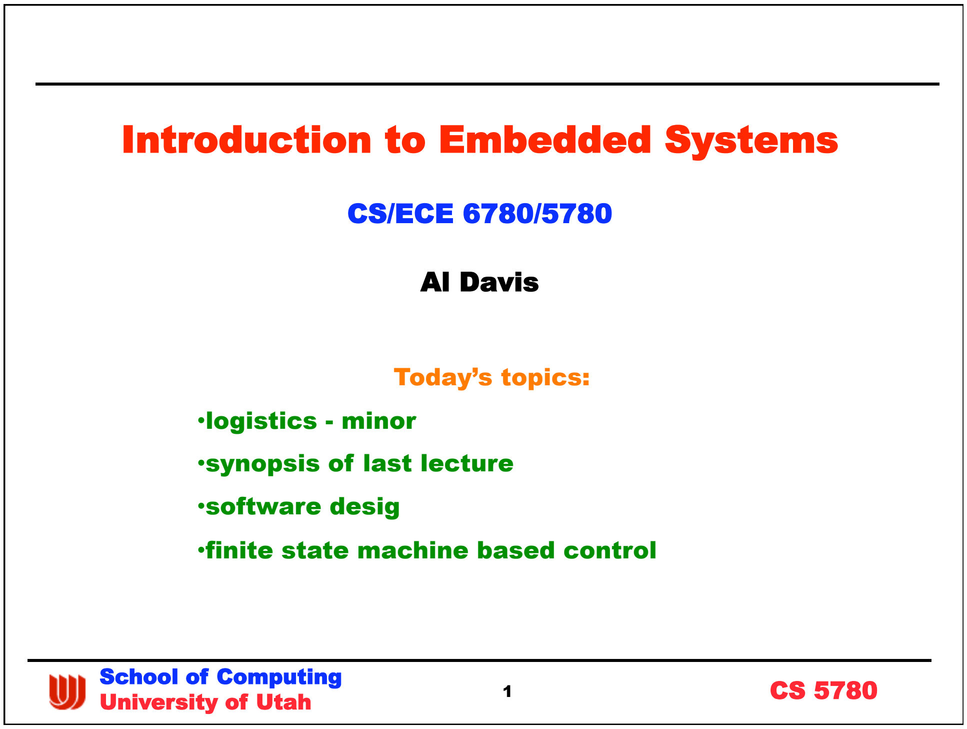


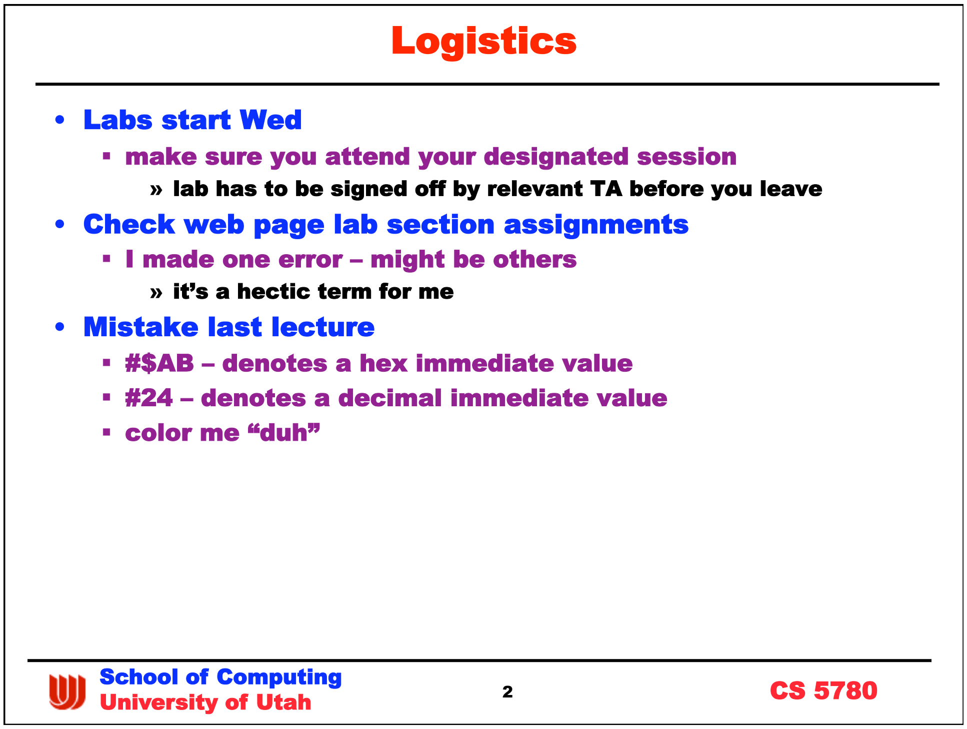
Introduction to Embedded Systems

CS/ECE 6780/5780

Al Davis

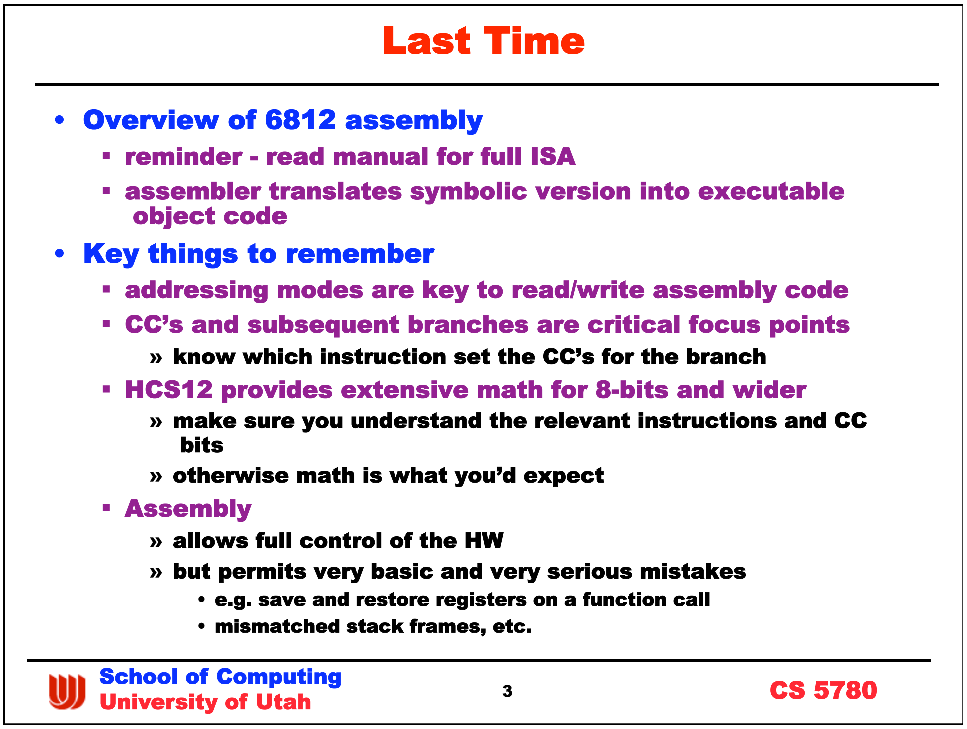
Today's topics:

- logistics - minor
- synopsis of last lecture
- software desig
- finite state machine based control



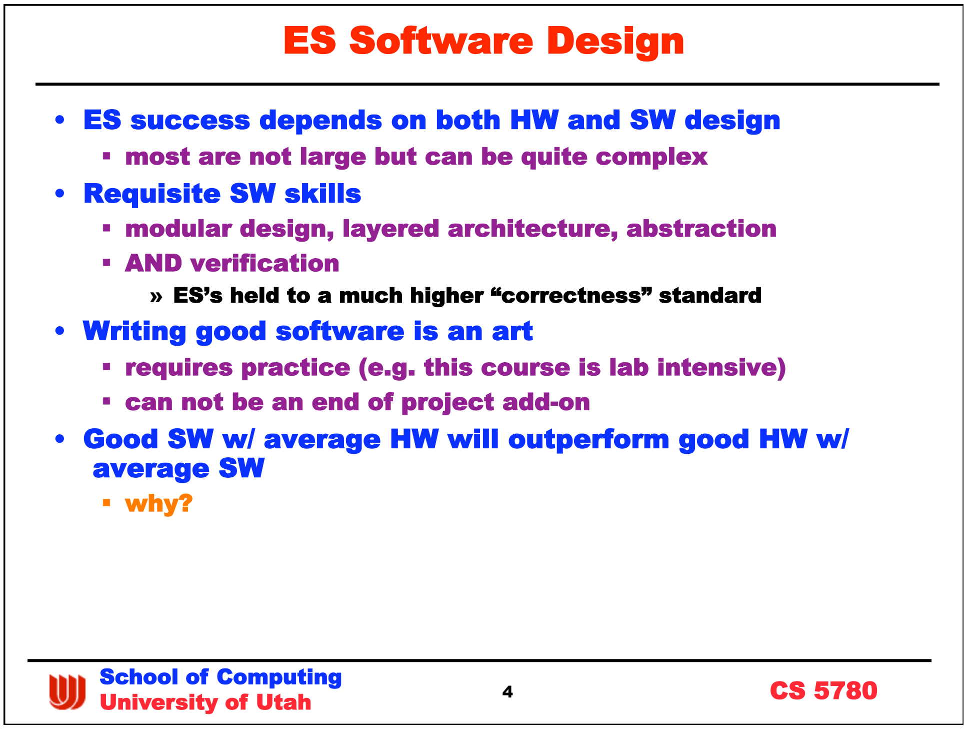
Logistics

- Labs start Wed
  - make sure you attend your designated session
    - » lab has to be signed off by relevant TA before you leave
- Check web page lab section assignments
  - I made one error – might be others
    - » it's a hectic term for me
- Mistake last lecture
  - #$AB – denotes a hex immediate value
  - #24 – denotes a decimal immediate value
  - color me "duh"



Last Time

- Overview of 6812 assembly
  - reminder - read manual for full ISA
  - assembler translates symbolic version into executable object code
- Key things to remember
  - addressing modes are key to read/write assembly code
  - CC's and subsequent branches are critical focus points
    - » know which instruction set the CC's for the branch
  - HCS12 provides extensive math for 8-bits and wider
    - » make sure you understand the relevant instructions and CC bits
    - » otherwise math is what you'd expect
  - Assembly
    - » allows full control of the HW
    - » but permits very basic and very serious mistakes
      - e.g. save and restore registers on a function call
      - mismatched stack frames, etc.



ES Software Design

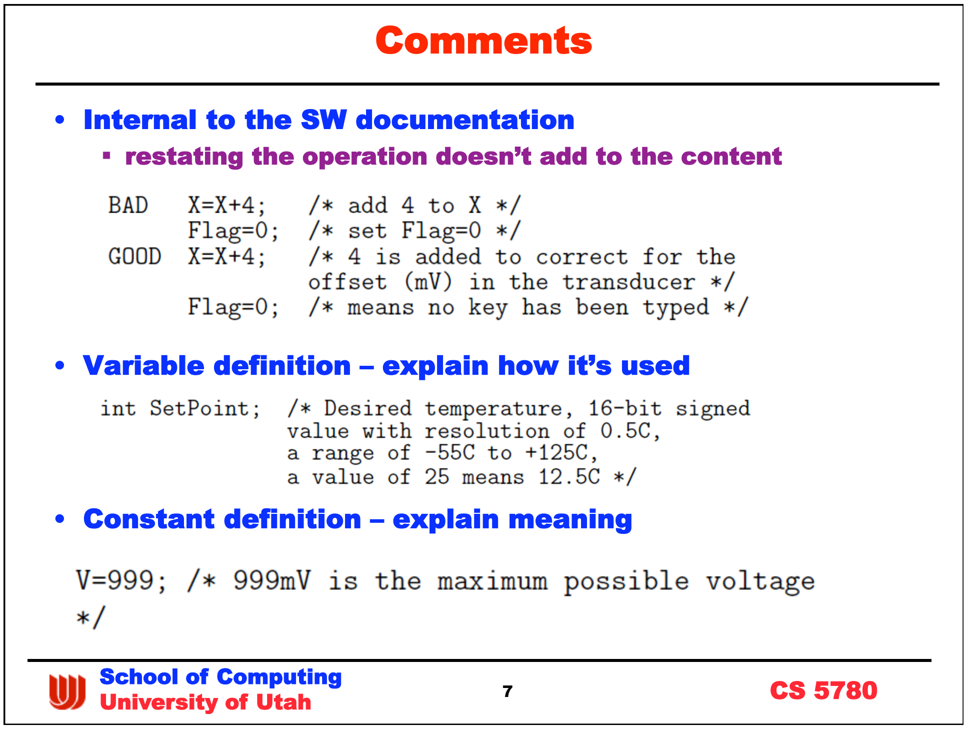
- ES success depends on both HW and SW design
  - most are not large but can be quite complex
- Requisite SW skills
  - modular design, layered architecture, abstraction
  - AND verification
    - » ES's held to a much higher "correctness" standard
- Writing good software is an art
  - requires practice (e.g. this course is lab intensive)
  - can not be an end of project add-on
- Good SW w/ average HW will outperform good HW w/ average SW
  - why?

Good SW – what does it look like?

- **Quantitative measures**
  - ***Dynamic efficiency*** **– number of CPU cycles and power**
  - ***Static efficiency*** **– RAM/ROM code/data footprint**
  - **Design constraints satisfied?**
- **Qualitative measures**
  - **Ease of debug**
  - **Ease of verification – prove correct**
  - **Ease of maintenance – enhance features**
- **Note**
  - **sacrifice clarity to enhance speed is usually a bad choice**
    - » **leads to bugs and leads to maintenance nightmares**
- **You're a good programmer when:**
  - **you can understand your code a year later**
  - **others find it relatively easy to modify your code**

SW Maintenance

- **Extremely important design phase**
  - **may persist longer than other phases**
    - » **think iTunes vs. a particular iPod platform**
- **Includes**
  - **initial bug fixes**
  - **add features**
  - **optimization**
  - **porting to new hardware**
  - **porting to new OS or run-time system**
  - **reconfigure to handle new requirements**
- **Documentation**
  - **should assist maintenance**
  - **resides in and outside of the SW itself**
    - » **caveat – programmer and external tech writer are seldom the same people**
      - **ever read the manual and then decided just to figure it out?**



Comments

- **Internal to the SW documentation**
  - **restating the operation doesn't add to the content**

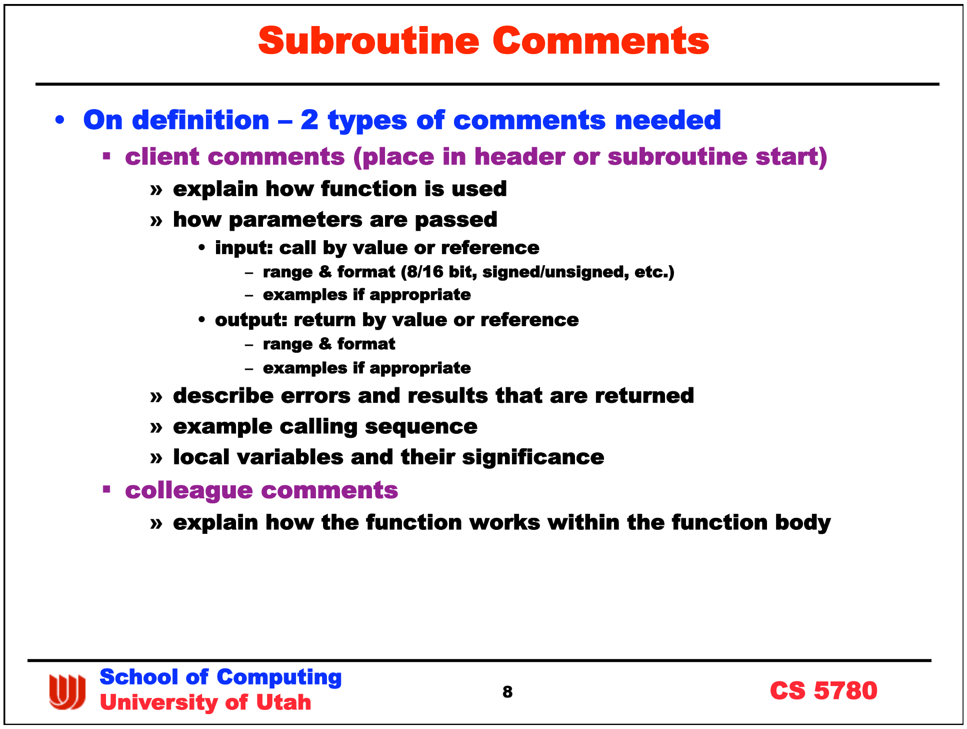
```
BAD   X=X+4;   /* add 4 to X */
      Flag=0;  /* set Flag=0 */
GOOD  X=X+4;   /* 4 is added to correct for the
               offset (mV) in the transducer */
      Flag=0;  /* means no key has been typed */
```

- **Variable definition – explain how it's used**

```
int SetPoint;  /* Desired temperature, 16-bit signed
               value with resolution of 0.5C,
               a range of -55C to +125C,
               a value of 25 means 12.5C */
```

- **Constant definition – explain meaning**

```
V=999; /* 999mV is the maximum possible voltage
*/
```



Subroutine Comments

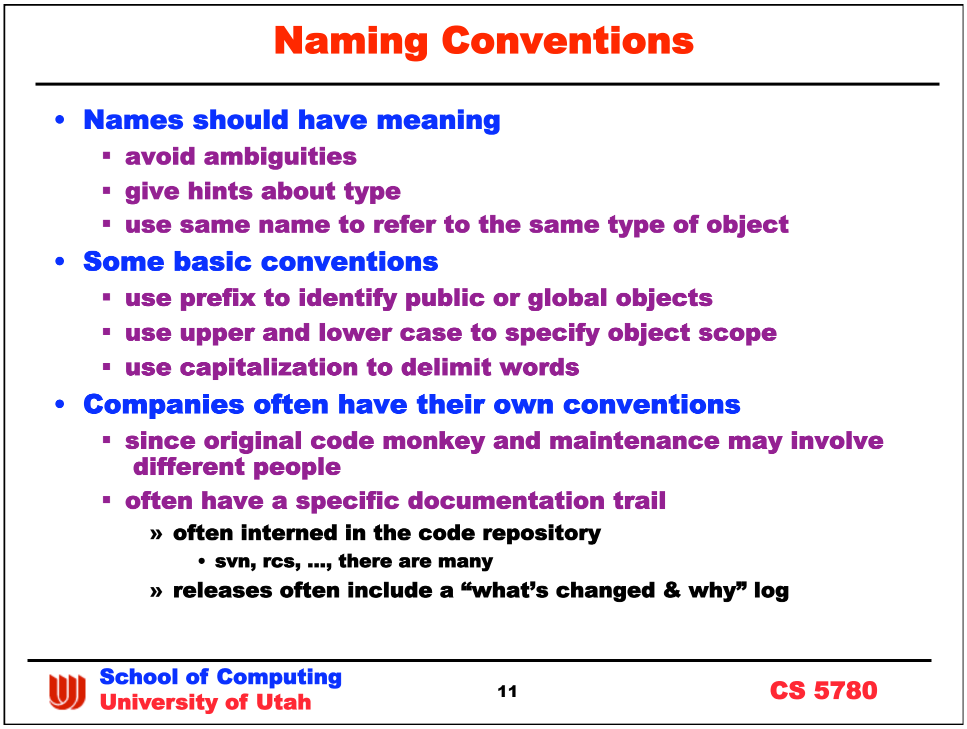
- **On definition – 2 types of comments needed**
  - **client comments (place in header or subroutine start)**
    - » **explain how function is used**
    - » **how parameters are passed**
      - **input: call by value or reference**
        - **range & format (8/16 bit, signed/unsigned, etc.)**
        - **examples if appropriate**
      - **output: return by value or reference**
        - **range & format**
        - **examples if appropriate**
    - » **describe errors and results that are returned**
    - » **example calling sequence**
    - » **local variables and their significance**
  - **colleague comments**
    - » **explain how the function works within the function body**

Self-Documenting Code

- SW written in a way that both purpose and function are self-apparent
  - use descriptive names for variables, constants, & functions
    - » remember that someone else be doing maintenance
      - 5780: this doesn’t really apply except to your lab partner
      - but might as well develop good habits and stick to it
    - » document your naming convention
  - formulate and organize your code
    - » into a well defined hierarchy of sub-components
    - » ideal
      - match problem decomposition with program structure
  - liberal use of #define or equ statements helps

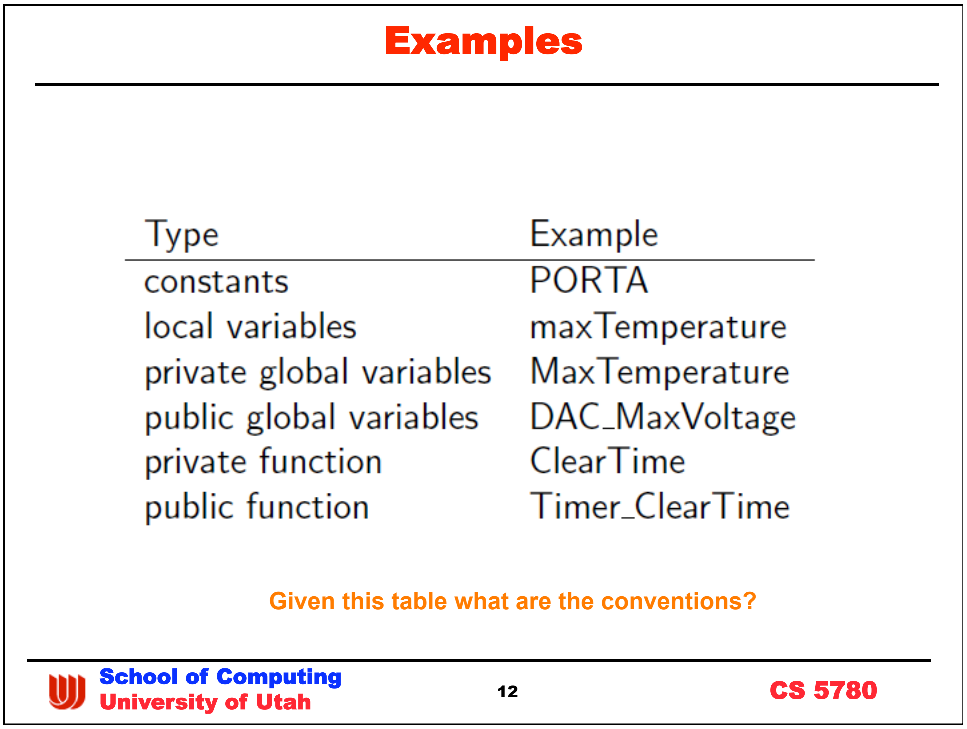
Using #define

```
// An inappropriate use of #define.
#define size 10
short data[size];
void initialize(void){ short j
   for(j=0;j<10;j++)
      data[j]=0;
};
// An appropriate use of #define.
#define size 10
short data[size];
void initialize(void){ short j
   for(j=0;j<size;j++)
      data[j]=0;
};
```



Naming Conventions

- Names should have meaning
  - avoid ambiguities
  - give hints about type
  - use same name to refer to the same type of object
- Some basic conventions
  - use prefix to identify public or global objects
  - use upper and lower case to specify object scope
  - use capitalization to delimit words
- Companies often have their own conventions
  - since original code monkey and maintenance may involve different people
  - often have a specific documentation trail
    - » often interned in the code repository
      - svn, rcs, …, there are many
    - » releases often include a "what's changed & why" log



Examples

Type	Example
constants	PORTA
local variables	maxTemperature
private global variables	MaxTemperature
public global variables	DAC_MaxVoltage
private function	ClearTime
public function	Timer_ClearTime

Given this table what are the conventions?

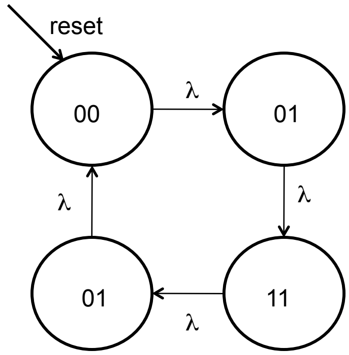
Abstraction

- A SW abstraction factors common functionality out of diverse examples
  - advantages
    - » faster to develop because some building blocks already exist
    - » easier to debug
      - due to separation of concept and implementation
    - » easier to understand
      - understand the abstraction and then see how it is implemented
    - » easier to change
      - if you understand it you know how to change it
- Finite state machine (FSM)
  - simple concept consisting of:
    - » states, inputs, outputs, and state transitions
  - FSM software is easy to understand, debug, & modify
  - works equally well in HW or SW

FSM Concepts

- When are outputs assigned?
  - by state ::= Moore machine
  - by transition ::= Mealy machine
  - equally powerful
    - » so use the version that is most intuitive for the problem at hand
    - » turn the crank procedure exists to convert Mealy ←→ Moore
- When are the inputs observed
  - when they happen → asynchronous FSM
  - based on some periodic time step → synchronous FSM
    - » we'll focus on this one since asynch FSM's have some additional complexity
    - » and most microcontrollers are synchronous
      - which makes synch FSM's an easy and natural choice
- Anybody use this for their lab1 abstraction?
  - when? == button_push

2-bit Gray Code FSM

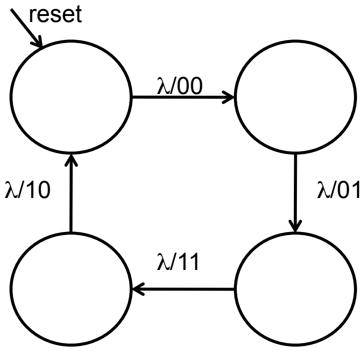


Moore or Mealy?

Where are the inputs?

What is λ ?

Mealy 2-bit Gray Counter

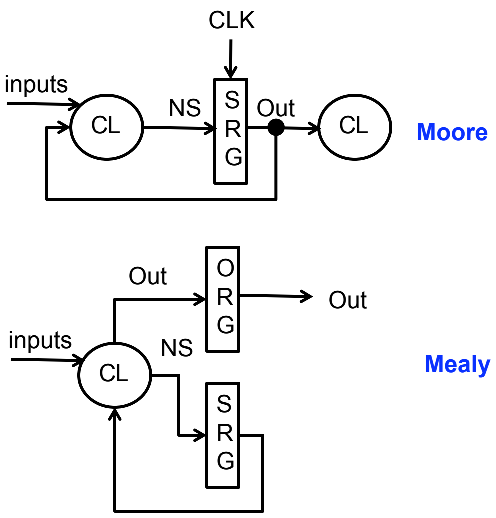


In the case where there are no inputs – Mealy vs. Moore distinction is moot.

reset – ALWAYS needs to be there since it helps to know which state is in play when you start.

Looks like we need a timer.

FSM's in HW



6812 Timer Details

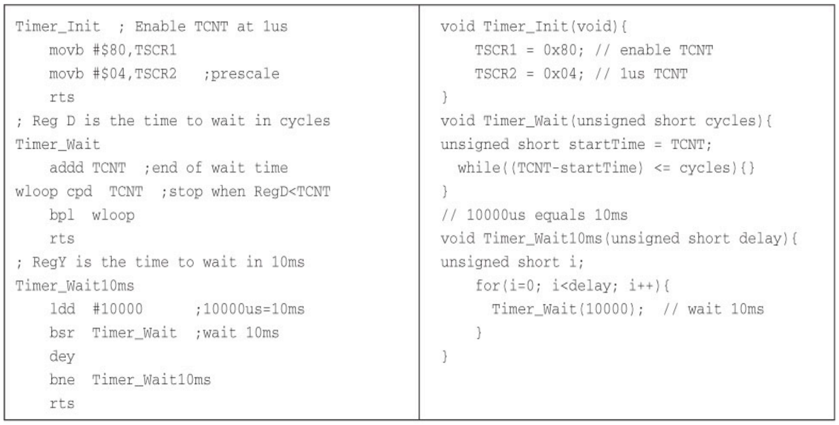
- **TCNT is a 16-bit unsigned counter**
  - **increments at a rate defined by 3 prescale bits in TSCR2**
    - » **bit 7 in TSCR1 enables use of the TCNT timer**
  - **assuming a 4 MHz E clock**

PR2	PR1	PR0	Divide by	TCNT Period	TCNT Frequency
0	0	0	1	250ns	4 MHz
0	0	1	2	500ns	2 MHz
0	1	0	4	1μs	1 MHz
0	1	1	8	2μs	500 kHz
1	0	0	16	4μs	250 kHz
1	0	1	32	8μs	125 kHz
1	1	0	64	16μs	62.5 kHz
1	1	1	128	32μs	31.25 kHz

$/2^{\text{PR-bits}}$

- **When TCNT overflows, TOF flag in TFLG2 register is set**
  - » **causes an interrupt if the TOI bit in TSCR2 is set.**

Implementing a Time Delay



```
Timer_Init   ; Enable TCNT at 1us
    movb #$80,TSCR1
    movb #$04,TSCR2    ;prescale
    rts
; Reg D is the time to wait in cycles
Timer_Wait
    addd TCNT   ;end of wait time
wloop cpd  TCNT   ;stop when RegD<TCNT
    bpl  wloop
    rts
; RegY is the time to wait in 10ms
Timer_Wait10ms
    ldd  #10000      ;10000us=10ms
    bsr  Timer_Wait  ;wait 10ms
    dey
    bne  Timer_Wait10ms
    rts
```

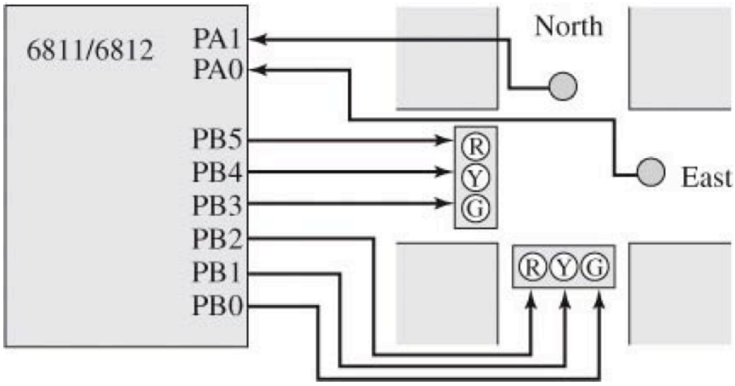
```
void Timer_Init(void){
    TSCR1 = 0x80; // enable TCNT
    TSCR2 = 0x04; // 1us TCNT
}
void Timer_Wait(unsigned short cycles){
unsigned short startTime = TCNT;
  while((TCNT-startTime) <= cycles){}
}
// 10000us equals 10ms
void Timer_Wait10ms(unsigned short delay){
unsigned short i;
    for(i=0; i<delay; i++){
      Timer_Wait(10000);  // wait 10ms
    }
}
```

Program 2.10
Timer functions that implement a time delay.

Note modular (maybe taken too far)

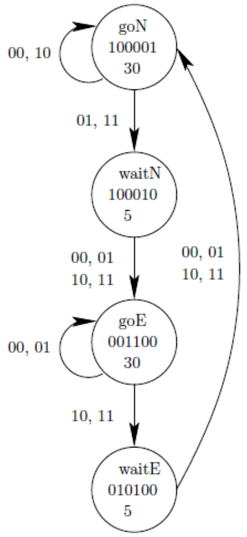
Traffic Light

- **2 one-way roads**



other ways to do it?

Moore FSM & State Table



Input vector <N,E>

Output vector <RE,YE,GE,RN,YN,GN>

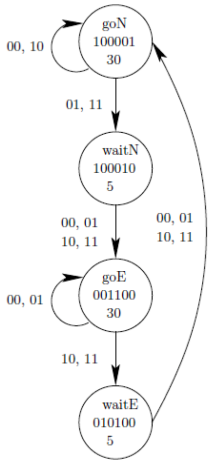
	No cars	Car E	Car N	Car N,E
goN	goN	waitN	goN	waitN
waitN	goE	goE	goE	goE
goE	goE	goE	waitE	waitE
waitE	goN	goN	goN	goN

For every input next_state must be spec'd

Implementing the FSM in SW

- **Required**
  - **initialize timer (in this case)**
  - **initial state specified**
    - » **entry point or go there on some reset signal**
  - **build FSM controller**
    - » **implementation options**
      - **mess of jumps (OK for simple state machine – bad for complex)**
        - – **bigger code footprint**
        - – **con: complete rewrite if FSM changes**
      - **table based**
        - – **bigger data footprint**
      - **pointer based**
        - – **even bigger data footprint but will run faster**
        - – **in this case you don't really care**
      - **maybe some others?**

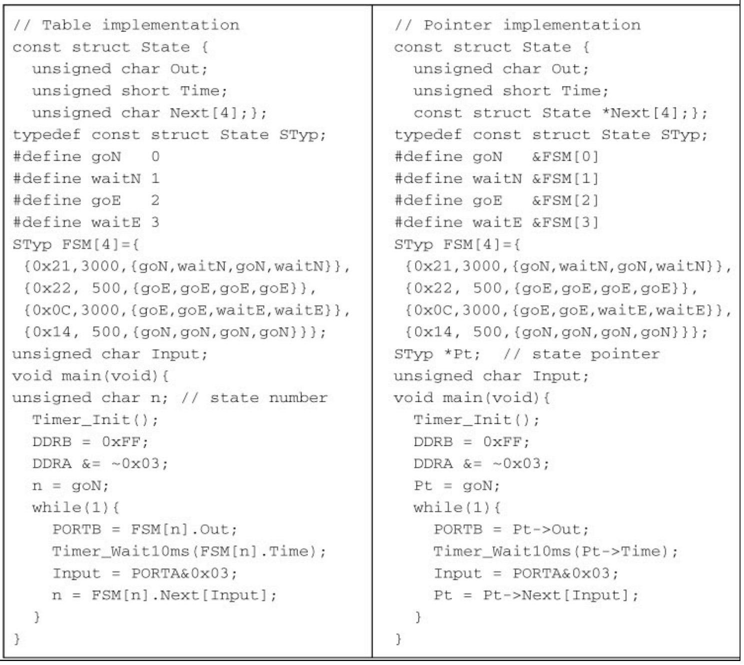
Goto Pseudocode



```
goN:
   output = 0x21;
   wait(30);
   if (E==0) goto goN;
waitN:
   output = 0x22;
   wait(5)
goE:
   output = 0x0C;
   wait(30);
   if (N==0) goto goE;
waitE:
   output = 0x14;
   wait(5);
   goto goN;
```

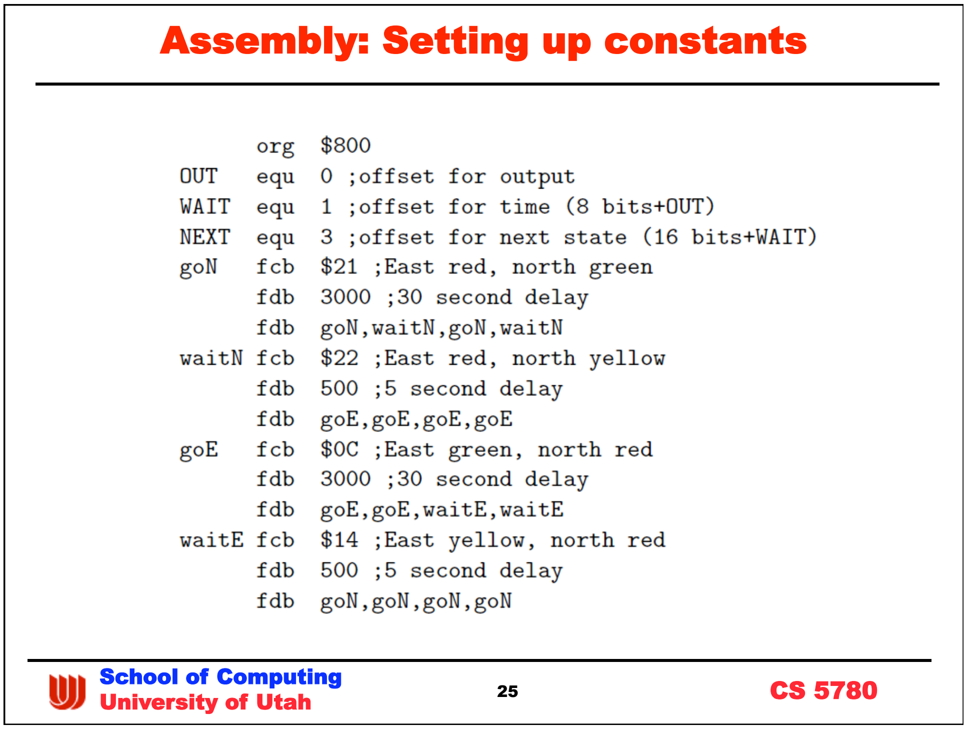
Table vs. Pointer Implementation in C

Program 2.11
Two 6812
C implementations of a
Moore FSM.



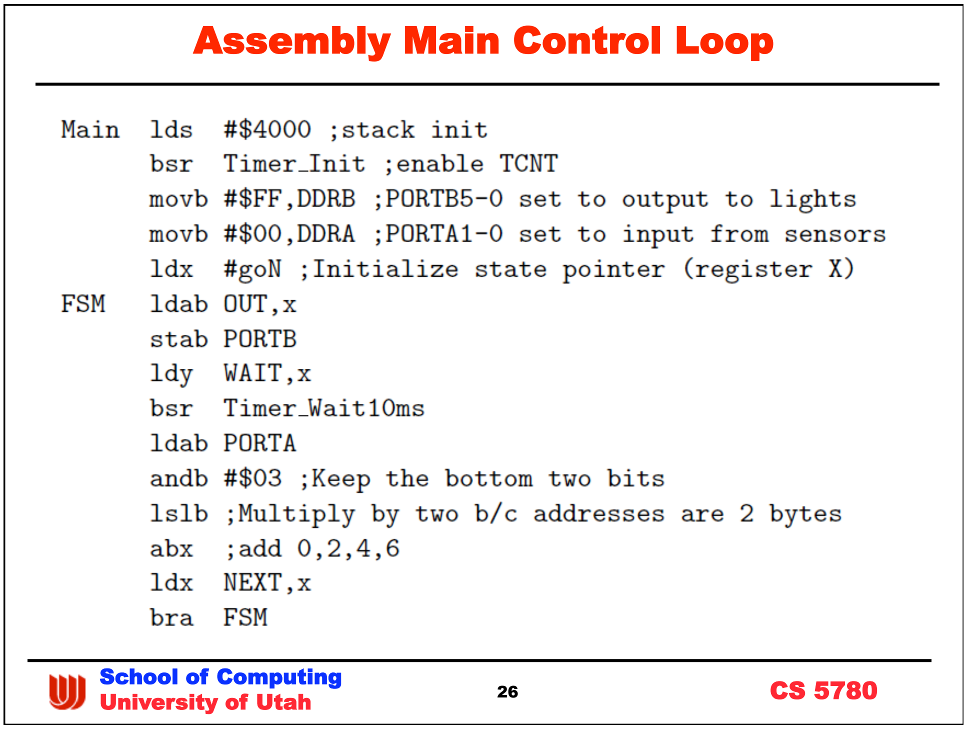
```
// Table implementation
const struct State {
  unsigned char Out;
  unsigned short Time;
  unsigned char Next[4];};
typedef const struct State STyp;
#define goN   0
#define waitN 1
#define goE   2
#define waitE 3
STyp FSM[4]={
 {0x21,3000,{goN,waitN,goN,waitN}},
 {0x22, 500,{goE,goE,goE,goE}},
 {0x0C,3000,{goE,goE,waitE,waitE}},
 {0x14, 500,{goN,goN,goN,goN}}};
unsigned char Input;
void main(void){
unsigned char n; // state number
  Timer_Init();
  DDRB = 0xFF;
  DDRA &= ~0x03;
  n = goN;
  while(1){
    PORTB = FSM[n].Out;
    Timer_Wait10ms(FSM[n].Time);
    Input = PORTA&0x03;
    n = FSM[n].Next[Input];
  }
}
```

```
// Pointer implementation
const struct State {
  unsigned char Out;
  unsigned short Time;
  const struct State *Next[4];};
typedef const struct State STyp;
#define goN   &FSM[0]
#define waitN &FSM[1]
#define goE   &FSM[2]
#define waitE &FSM[3]
STyp FSM[4]={
 {0x21,3000,{goN,waitN,goN,waitN}},
 {0x22, 500,{goE,goE,goE,goE}},
 {0x0C,3000,{goE,goE,waitE,waitE}},
 {0x14, 500,{goN,goN,goN,goN}}};
STyp *Pt;  // state pointer
unsigned char Input;
void main(void){
  Timer_Init();
  DDRB = 0xFF;
  DDRA &= ~0x03;
  Pt = goN;
  while(1){
    PORTB = Pt->Out;
    Timer_Wait10ms(Pt->Time);
    Input = PORTA&0x03;
    Pt = Pt->Next[Input];
  }
}
```



Assembly: Setting up constants

```
      org  $800
OUT   equ  0 ;offset for output
WAIT  equ  1 ;offset for time (8 bits+OUT)
NEXT  equ  3 ;offset for next state (16 bits+WAIT)
goN   fcb  $21 ;East red, north green
      fdb  3000 ;30 second delay
      fdb  goN,waitN,goN,waitN
waitN fcb  $22 ;East red, north yellow
      fdb  500 ;5 second delay
      fdb  goE,goE,goE,goE
goE   fcb  $0C ;East green, north red
      fdb  3000 ;30 second delay
      fdb  goE,goE,waitE,waitE
waitE fcb  $14 ;East yellow, north red
      fdb  500 ;5 second delay
      fdb  goN,goN,goN,goN
```



Assembly Main Control Loop

```
Main  lds  #$4000 ;stack init
      bsr  Timer_Init ;enable TCNT
      movb #$FF,DDRB ;PORTB5-0 set to output to lights
      movb #$00,DDRA ;PORTA1-0 set to input from sensors
      ldx  #goN ;Initialize state pointer (register X)
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03 ;Keep the bottom two bits
      lslb ;Multiply by two b/c addresses are 2 bytes
      abx  ;add 0,2,4,6
      ldx  NEXT,x
      bra  FSM
```

Partial Memory Map

```
      org  $0800
OUT   equ  0
WAIT  equ  1
NEXT  equ  3
goN   fcb  $21
      fdb  3000
      fdb  goN,waitN,
           goN,waitN
waitN fcb  $22
      fdb  500
      fdb  goE,goE,
           goE,goE
goE   fcb  $0C
```

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Execution

```
    ldx  #goN
FSM ldab OUT,x
    stab PORTB
    ldy  WAIT,x
    bsr  Timer_Wait10ms
    ldab PORTA
    andb #$03
    lslb
    abx
    ldx  NEXT,x
    bra  FSM
```

RegX	08 00
RegY	XX XX
AccB	XX

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Execution

```
        ldx  #goN
FSM     ldab OUT,x  ;0800+0
        stab PORTB
        ldy  WAIT,x
        bsr  Timer_Wait10ms
        ldab PORTA
        andb #$03
        lslb
        abx
        ldx  NEXT,x
        bra  FSM
```

RegX	08 00
RegY	XX XX
AccB	21

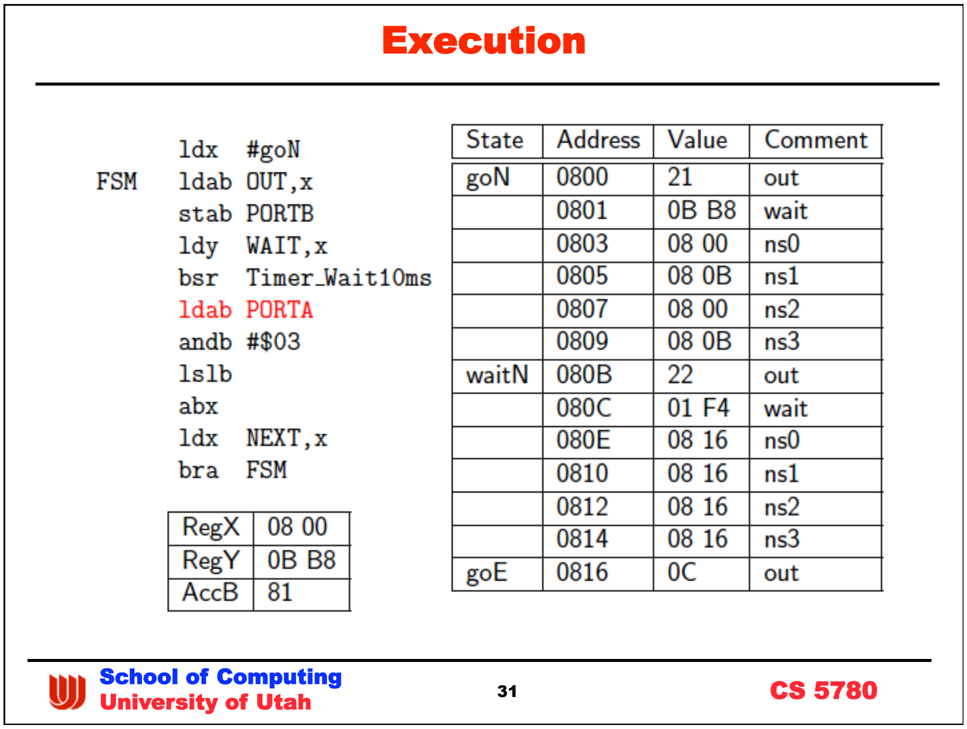
State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Execution

```
        ldx  #goN
FSM     ldab OUT,x
        stab PORTB
        ldy  WAIT,x  ;0800+1
        bsr  Timer_Wait10ms
        ldab PORTA
        andb #$03
        lslb
        abx
        ldx  NEXT,x
        bra  FSM
```

RegX	08 00
RegY	0B B8
AccB	21

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

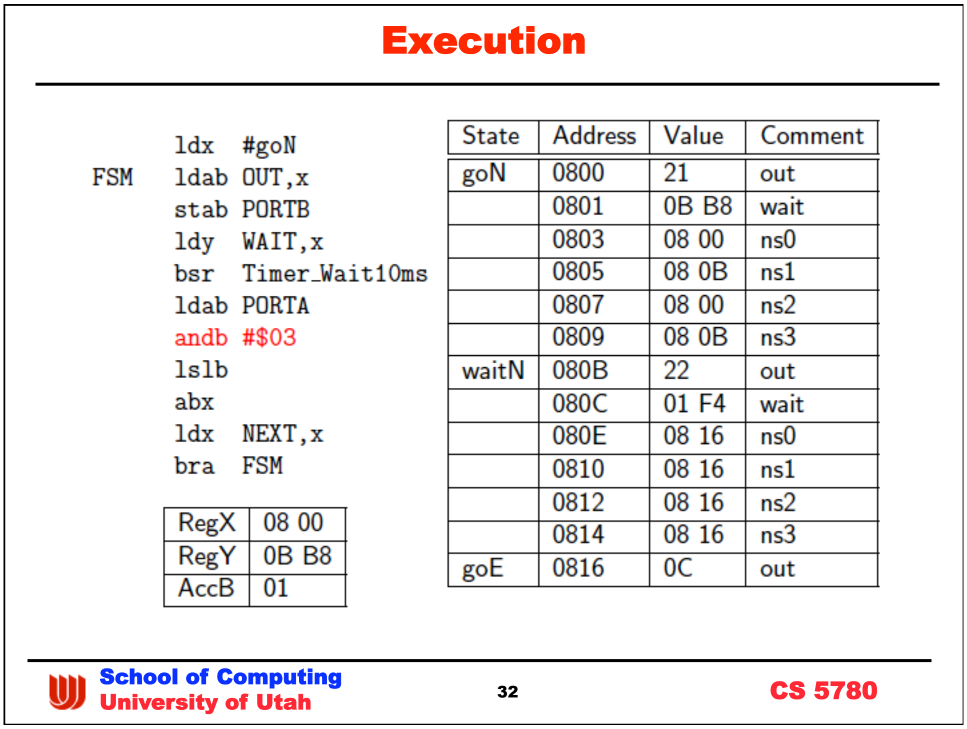


Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 00
RegY	0B B8
AccB	81

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

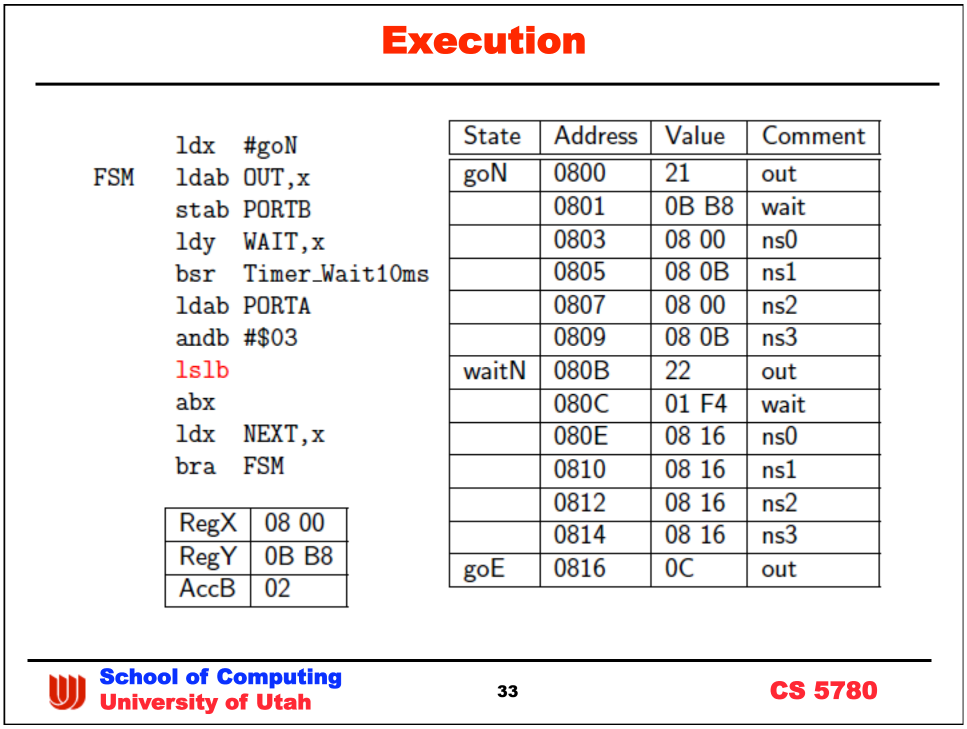


Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 00
RegY	0B B8
AccB	01

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

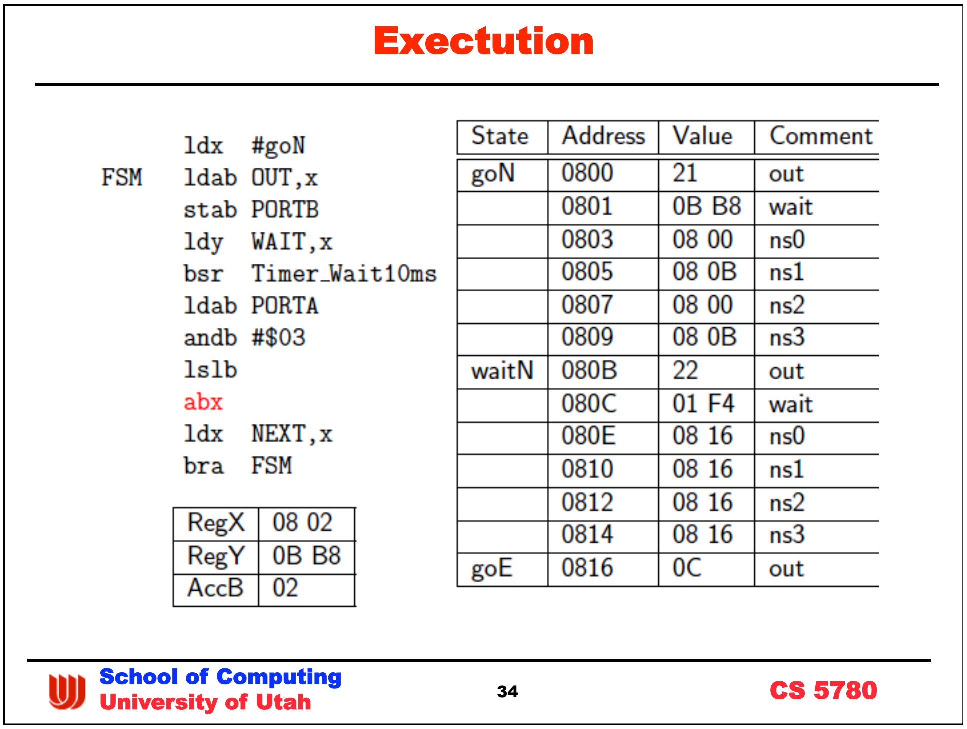


Execution

```
        ldx  #goN
FSM     ldab OUT,x
        stab PORTB
        ldy  WAIT,x
        bsr  Timer_Wait10ms
        ldab PORTA
        andb #$03
        lslb
        abx
        ldx  NEXT,x
        bra  FSM
```

RegX	08 00
RegY	0B B8
AccB	02

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out



Exectution

```
        ldx  #goN
FSM     ldab OUT,x
        stab PORTB
        ldy  WAIT,x
        bsr  Timer_Wait10ms
        ldab PORTA
        andb #$03
        lslb
        abx
        ldx  NEXT,x
        bra  FSM
```

RegX	08 02
RegY	0B B8
AccB	02

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

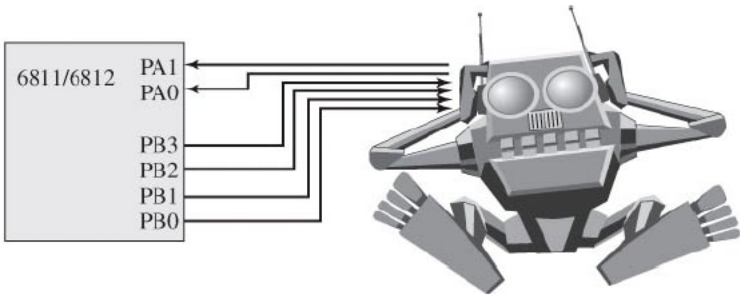
Executioon

```
        ldx  #goN
FSM     ldab OUT,x
        stab PORTB
        ldy  WAIT,x
        bsr  Timer_Wait10ms
        ldab PORTA
        andb #$03
        lslb
        abx
        ldx  NEXT,x ;0802+3
        bra  FSM
```

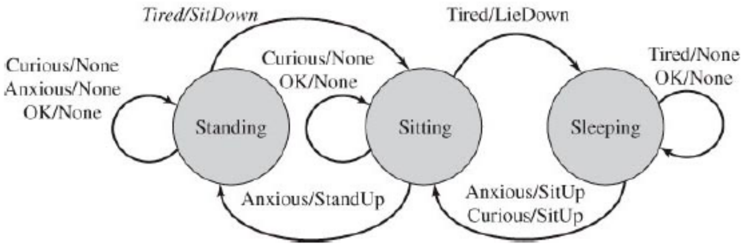
RegX	08 0B
RegY	0B B8
AccB	02

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Weary Robot Interface



Mealy FSM for a Robot Controller



Difference – output depends on transition so 2 tables: next_state & output
go through the example in your text

Concluding Remarks

- **FSM's are your friend**
  - **nice abstraction mechanism**
    - » **works well for both HW & SW control design**
    - » **context aware transfer function model of the world**
      - **context = state**
      - **transfer function = CL model = output response to input stimulus**
      - **context + transfer function models → FSM**
  - **multiple implementation styles**
    - » **choice depends on problem/environment constraints**
      - **code or data footprint**
      - **execution speed**
    - » **if no critical constraints**
      - **then pick the implementation strategy that max's clarity**
      - **usually transparency between abstract FSM (state graph) and code**
- **Other FSM benefits**
  - **turn the crank minimization procedures**
    - » **developed for HW designs also apply to SW – NICE!**
      - **any intro book on digital design will show you how**
- **What what should you do if your FSM gets huge?**