CMPE-013/L

Introduction to "C" Programming

Max Lichtenstein

Piazza poll: Which Interrupt Service Routine(s) are valid?

```
//A ================================================:
void __ISR(_TIMER_1_VECTOR, IPL4AUTO) t1(void) {
    INTClearFlag(INT_T1);

        myTime++;
}

//B ================================================:
int __ISR(_TIMER_1_VECTOR, IPL4AUTO) t1(void) {
    INTClearFlag(INT_T1);

        myTime++;
        return myTime;
}

//C ================================================:
void __ISR(_TIMER_1_VECTOR, IPL4AUTO) t1(int timeSkip) {
    INTClearFlag(INT_T1);

        myTime+=timeSkip;
        return myTime;
}
```

Roadmap

- Announcements, Notes
- Macros
- Interrupts
- Break
- Reactive (Event-driven) Programming
  - Synchronous vs Asynchronous SMs
- States and Events
  - States vs Events
  - State Machines
    - As models for the external world
    - As code design

Announcements

- Lab 6, 7 due dates
  - 6 due Tuesday,
  - 7 out Monday and due Monday

Piazza Notes: :(

Summary
(100 characters or less)

WTF is wrong with my code?!?!?!

Details
use plain text editor

Edit ▾ Insert ▾ View ▾ Format ▾ Table ▾

B *I* code tt markdown Help

```
long int i;
// whichTest = BASIC;
// whichTest = BLOCKING_DELAY;
// whichTest = LATCH;
whichTest = STATE_MACHINE;
switch (whichTest) {
case BASIC:
printf("How many times has the button been pressed?\n");
while (1) {
if (BTN4) {
button_count++;
```

Posting Options

- ☐ Make this an announcement (note appears on the course page)
- ☐ Send email notifications immediately (bypassing students' email preferences, if necessary)

Piazza Notes: :)

Summary
(100 characters or less)

WTF is wrong with my code?!?!?!

Details
use plain text editor

Edit ▾ Insert ▾ View ▾ Format ▾ Table ▾

B *I* | code | markdown | Help

```
whichTest = STATE_MACHINE;

switch (whichTest) {
    case BASIC:
        printf("How many times has the button been pressed?\n");
        while (1) {
            if (BTN4) {
                button_count++;
            };
            printf("%d times\n", button_count);
```

Posting Options

- ☐ Make this an announcement (note appears on the course page)
- ☐ Send email notifications immediately (bypassing students' email preferences, if necessary)

Notes on lab grades

- Code that doesn't compile generally receives 0 points
  - Submit early and often!
  - Most code is for others

.h is a Contract

- Extra credit "cushion"
  - Grades shouldn't exceed 100%
  - Your past grades may have changed recently

Lab 06 Notes

- Bad .h file
  - `ButtonsCheckEvents(uint8_t button_states)`
- Missing semicolon in lab manual
- Uno32 IO board schematic
  https://reference.digilentinc.com/_media/chipkit_shield_basic_io_shield:chipkit_basic_io_sch.pdf

Example test for LEDs:

```
LEDS_INIT();
LEDS_SET(0xCC);
int i;
for (i = 0; i < 10000000; i++);
LEDS_SET(0xDD);
for (i = 0; i < 10000000; i++);
LEDS_SET(0);
for (i = 0; i < 10000000; i++);
LEDS_SET(0xFF);
```

Buttons Test:

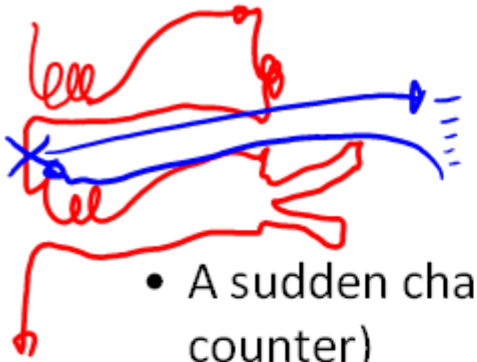
```
ButtonsCheckEvents(0b1101);

Init Timer
while1:
  PrintButtonEvents(BUTTON_STATE1)
```

Datasheet / SFR notes:

You'll learn more when you need it

Interrupts



Interrupts

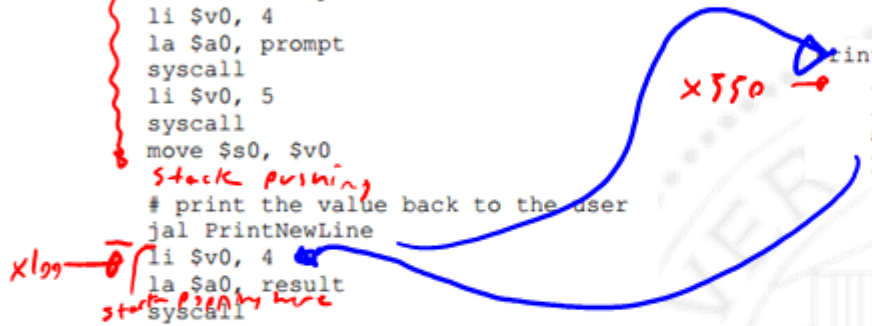
- A sudden change in your "code vector" (ie, program counter)
  - Externally caused (by processor)
  - Handled by an Interrupt Service Routine (ISR)
    - Basically, a function that the microcontroller calls

- Caused when the Microcontroller's hardware when:
  - Detects a change in the outside world
  - Internal timer went off
  - Internal process completed
  - Certain errors

Subroutines (functions) in MIPS32:

```
.text
main:
    # read an input value from the user
    li $v0, 4
    la $a0, prompt
    syscall
    li $v0, 5
    syscall
    move $s0, $v0
    # print the value back to the user
    jal PrintNewLine
    li $v0, 4
    la $a0, result
    syscall
    li $v0, 1
    move $a0, $s0
    syscall

PrintNewLine:
    li $v0, 4
    la $a0, __PNL_newline
    syscall
    jr $ra
```

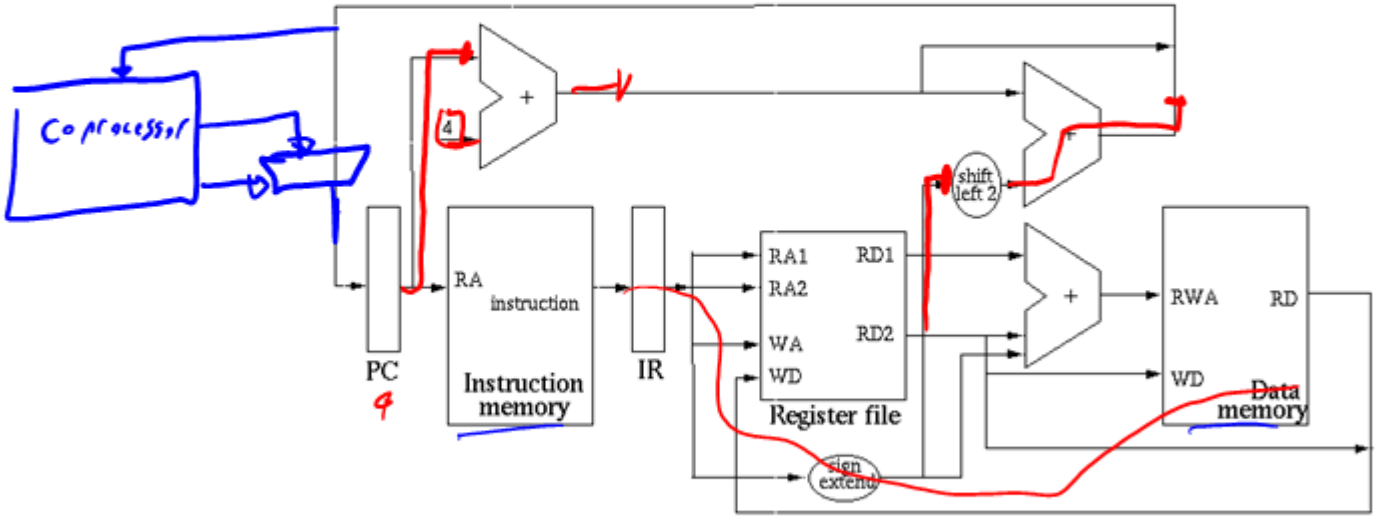


$ra = x100

$pc = x559

- From Charles Kann, *Introduction to MIPS Assembly Language Programming*

Interrupts in Hardware:



Interrupt Configuration:

The Interrupts Controller module consists of the following Special Function Registers (SFRs):

- **INTCON: Interrupt Control Register**

  This register controls the interrupt vector spacing, Single Vector or Multi-Vector modes, Interrupt Proximity, and external Interrupt edge detection.

- **PRISS: Priority Shadow Select Register**

  This register controls which CPU shadow register set is used by which interrupt priority, or in Single Vector mode, whether a shadow register set is used.

- **INTSTAT: Interrupt Status Register**

  This read-only register provides the status of the interrupt priority level or vector that is presented to the CPU.

- **IPTMR: Interrupt Proximity Timer Register**

  This register controls a timing window in which interrupts are held off from being presented to the CPU.

- **IFSx: Interrupt Flag Status Register**

  These registers contain the flags that indicate the status of the various interrupt sources, and also clears those interrupts sources.

- **IECx: Interrupt Enable Control Register**

  These registers contain the flags that enable or disable the interrupt sources from triggering an interrupt of the CPU.

- **IPCx: Interrupt Priority Control Register**

  These registers control the priority and sub-priority levels of the various interrupt sources.

- **OFFx: Interrupt Vector Address Offset Register**

  These registers contain the offset from EBASE that the CPU will jump to when the corresponding interrupt occurs.

Max Lichtenstein

```
void __ISR(_TIMER_1_VECTOR, IPL4AUTO) Timer1Handler(void) {
    // Clear the interrupt flag.
    INTClearFlag(INT_T1);

    // If we've exceeded the timer trigger count, trigger a timer event.
    if (++timerData.value > SWITCH_STATES()) {
        timerData.event = true;
        timerData.value = 0;
    }
}
```

What happens when an interrupt occurs?

```
void __ISR(_TIMER_1_VECTOR, IPL4AUTO) Timer1Handler(void) {
0x9D0026F8: RDPGPR SP, SP
0x9D0026FC: MFC0 K1, EPC
0x9D002700: MFC0 K0, SRSCtl
0x9D002704: ADDIU SP, SP, -120
0x9D002708: SW K1, 116(SP)
0x9D00270C: MFC0 K1, Status
0x9D002710: SW K0, 108(SP)
0x9D002714: SW K1, 112(SP)
0x9D002718: INS K1, ZERO, 1, 15
0x9D00271C: ORI K1, K1, 4096
0x9D002720: MTC0 K1, Status
0x9D002724: SW V1, 28(SP)
0x9D002728: SW V0, 24(SP)
0x9D00272C: LW V1, 108(SP)
0x9D002730: ANDI V1, V1, 15
0x9D002734: BNE V1, ZERO, 0x9D002780
0x9D002738: NOP
0x9D00273C: SW RA, 92(SP)
0x9D002740: SW S8, 88(SP)
0x9D002744: SW T9, 84(SP)
0x9D002748: SW T8, 80(SP)
0x9D00274C: SW T7, 76(SP)
0x9D002750: SW T6, 72(SP)
0x9D002754: SW T5, 68(SP)
0x9D002758: SW T4, 64(SP)
```

```
x9D002774: SW A1, 36(SP)
x9D002778: SW A0, 32(SP)
x9D00277C: SW AT, 20(SP)
x9D002780: NOP
x9D002784: MFLO V0
x9D002788: SW V0, 100(SP)
x9D00278C: MFHI V1
x9D002790: SW V1, 96(SP)
x9D002794: ADDU S8, SP, ZERO
    // Clear the interrupt flag.
    INTClearFlag(INT_T1);
x9D002798: ADDIU A0, ZERO, 8
x9D00279C: JAL INTClearFlag
x9D0027A0: NOP
```

Piazza poll: Which Interrupt Service Routine(s) are valid?

```
//A ================================================:
void __ISR(_TIMER_1_VECTOR, IPL4AUTO) t1(void) {
    INTClearFlag(INT_T1);

        myTime++;
}

//B ================================================:
int __ISR(_TIMER_1_VECTOR, IPL4AUTO) t1(void) {
    INTClearFlag(INT_T1);

        myTime++;
        return myTime;
}

//C ================================================:
void __ISR(_TIMER_1_VECTOR, IPL4AUTO) t1(int timeSkip) {
    INTClearFlag(INT_T1);

        myTime+=timeSkip;
        return myTime;
}
```

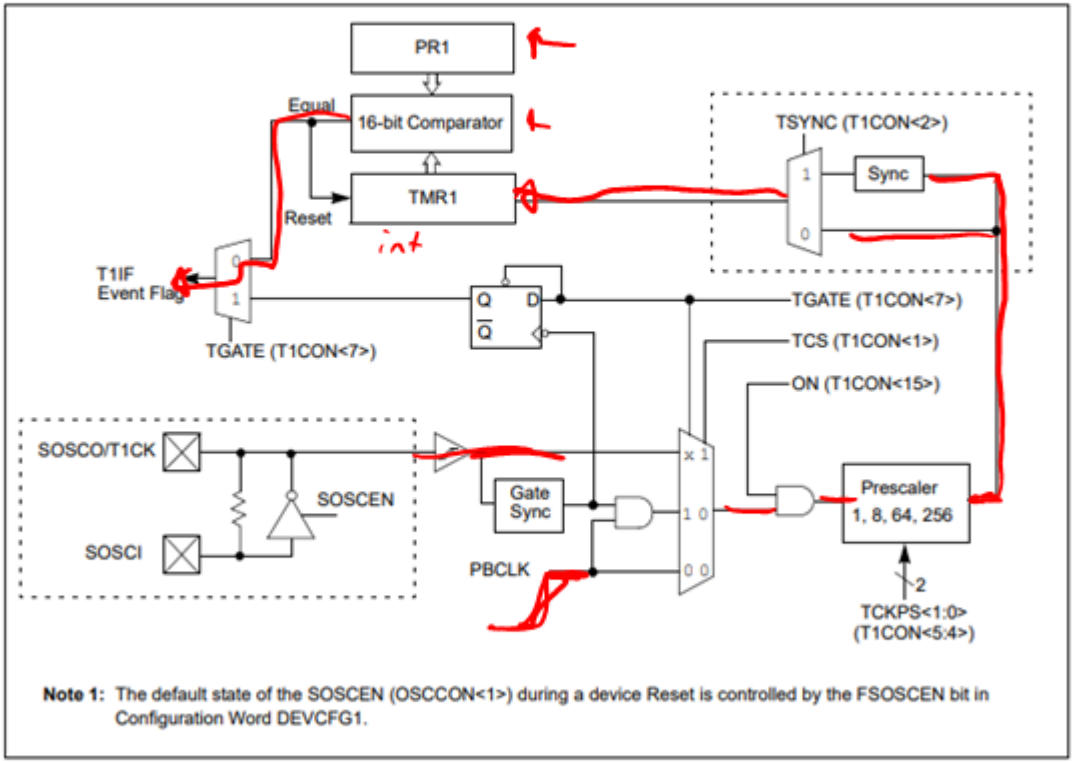
Max Lichtenstein

```
int __ISR(_TIMER_1_VECTOR) __ Timer1Handler(){
    time++;
    return time;
}
```

```
while(1);
  int startTime;
    runSlowProcess()
  print("%d", startTime - time);
```

Timer ISRs:

FIGURE 13-1: TIMER1 BLOCK DIAGRAM(1)



Note 1: The default state of the SOSCEN (OSCCON<1>) during a device Reset is controlled by the FSOSCEN bit in Configuration Word DEVCFG1.

Timer SFR documentation

TABLE 4-7: TIMER1-5 REGISTERS MAP(1)

Virtual Address (BF80_#)	Register Name	Bit Range	Bits															All Resets
			31/15	30/14	29/13	28/12	27/11	26/10	25/9	24/8	23/7	22/6	21/5	20/4	19/3	18/2	17/1	16/0	
0600	T1CON	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	ON	—	SIDL	TWDIS	TWIP	—	—	—	TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—	0000
0610	TMR1	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	TMR1<15:0>																0000
0620	PR1	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	PR1<15:0>																FFFF

- Diagram, complete register map:
  - In the PIC32MX3xx datasheet

Timer SFR documentation

Register 14-1: T1CON: Type A Timer Control Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	U-0	R/W-0	R/W-0	R-0	U-0	U-0	U-0
	ON(1)	—	SIDL	TWDIS	TWIP	—	—	—
7:0	R/W-0	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	U-0
	TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 31-16 **Unimplemented:** Read as '0'

bit 15 **ON:** Timer On bit(1)
- 1 = Timer is enabled
- 0 = Timer is disabled

bit 14 **Unimplemented:** Read as '0'

bit 13 **SIDL:** Stop in Idle Mode bit
- 1 = Discontinue operation when device enters Idle mode
- 0 = Continue operation when device enters Idle mode

bit 12 **TWDIS:** Asynchronous Timer Write Disable bit
- 1 = Writes to TMR1 are ignored until pending write operation completes
- 0 = Back-to-back writes are enabled (Legacy Asynchronous Timer functionality)

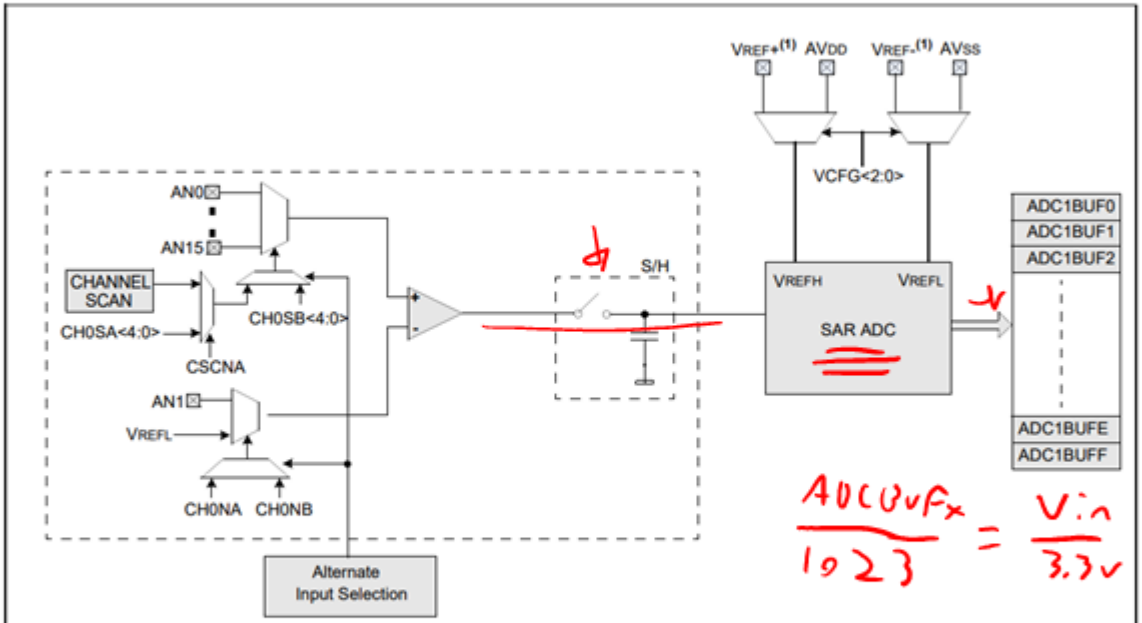
bit 5-4 **TCKPS<1:0>:** Timer Input Clock Prescale Select bits
- 11 = 1:256 prescale value
- 10 = 1:64 prescale value
- 01 = 1:8 prescale value
- 00 = 1:1 prescale value

- Register Descriptions:
  - In the PIC32 Family Reference Manual
    - (google for this)

Max Lichtenstein

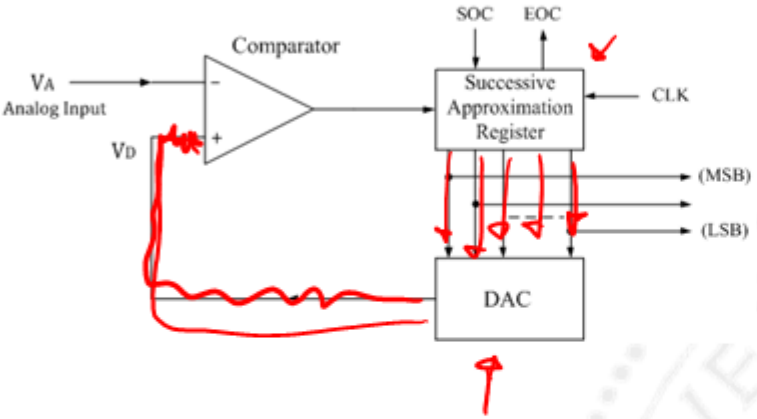
ADC ISR:

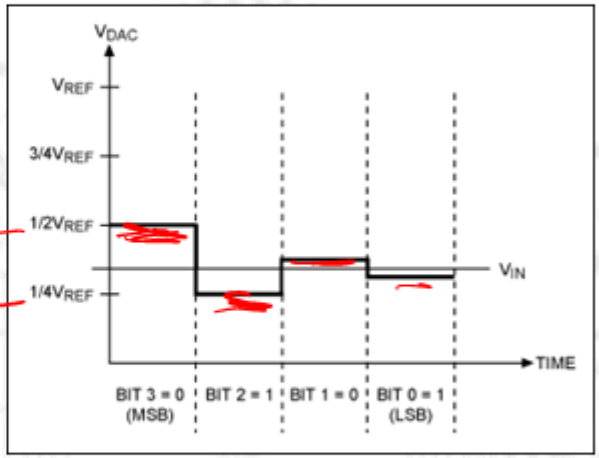
FIGURE 22-1: ADC1 MODULE BLOCK DIAGRAM



Note 1: VREF+, VREF- inputs can be multiplexed with other analog inputs.

ADC: Successive Approximation





Top speed:

Up to 1000 kilo samples per second (ksps) conversion speed

Kind of slow

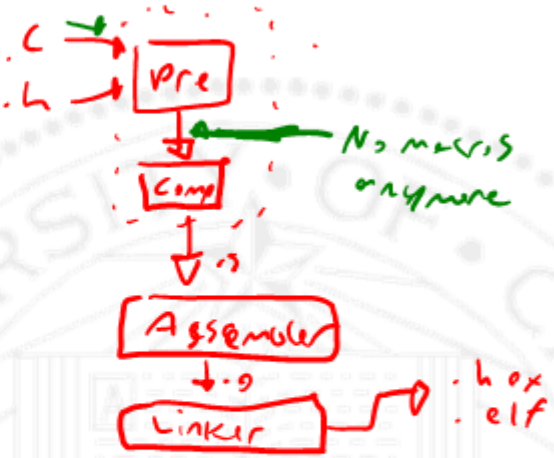
Other ISRs of interest

- Change Notification:
  - When a digital input goes from 0 to 1, or vice versa
  - Useful for encoders or warning sensors
- Comparator:
  - When voltage A exceeds voltage B
- UART, I2C, SPI USB:
  - Data received
  - Transmission complete

Macros

Macros

- Tell the pre-processor to do a simple find-and-replace
  - Easy to read
  - Efficient
- Can take arguments
  - Indifferent to type!
- Can be super confusing
  - Compiler can't give good error messages because it cannot "see" the macro
  - So use MPLABX's View Macro Expansion!
    - (right-click on .c code, Navigate -> View Macro Expansion)



Macros

Board.h

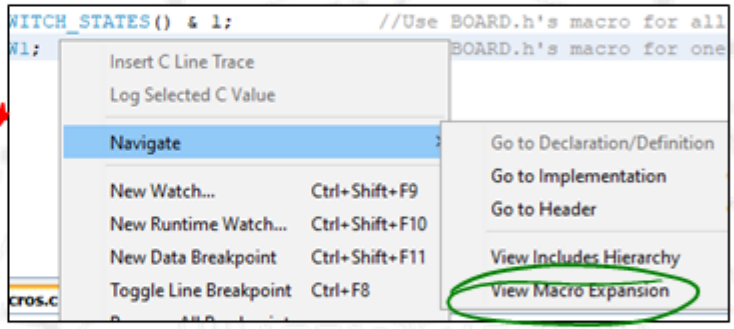
```
#define SW1 PORTDbits.RD8
#define SWITCH_STATES() ((PORTD >> 8) & 0x0F)
```

main()?

```
//hard to read:
switch1_state = (PORTD >> 8) & 1;
switch1_state = PORTDbits.RD8;

//Easy to read:
switch1_state = SWITCH_STATES() & 1;
switch1_state = SW1;
```

union in xc8.



Compiler sees:

```
//hard to read:
switch1_state = (PORTD >> 8) & 1;
switch1_state = PORTDbits.RD8;

//Easy to read:
switch1_state = ((PORTD >> 8) & 0x0F) & 1;
switch1_state = PORTDbits.RD8;
```

Macros with Arguments

```
#define FP_DELTA 0.001
#define EQUAL_WITHIN_DELTA(x, y) ((x - y) < FP_DELTA) && ((y - x) < FP_DELTA)

float a = 3.000001;
 int b = 3;
if (EQUAL_WITHIN_DELTA(a, b)) {
```

```
float a = 3.000001;
 int b = 3;
 if ((( a - b ) < 0.001 ) && (( b - a ) < 0.001 )) {
```

Max Lichtenstein

Macros vs Functions

```
void InitializeLEDsToPattern(char x)
{
    //Set LEDs to outputs:
    TRISE = 0x00;
    //set them to off by default:
    LATE = x;
}

#define INIT_LEDS_TO_PATTERN(x) {LATE = 0x00; TRISE = x;}

int main(void){

    //Initialize LEDs via function:
    InitializeLEDsToPattern(0b00110011);

    //Initialize LEDs via macro:
    INIT_LEDS_TO_PATTERN(0b11001100);
```

Discussion: What is 7*7 ?

```
#define   SQR(x)  (x*x)          ((x)*(x))

    printf("5+2 = 7, 7*7 = %d", SQR(5 + 2));
```

SQR(5+2)

(5+2 * 5+2)

(5+2*5+2)

5 + 10 + 2

Max Lichtenstein

#define if

Break

Event-Driven Programming

Event-Driven Programming

- So far, you've used code that:
  - runs once, at startup
  - Runs forever in a while loop

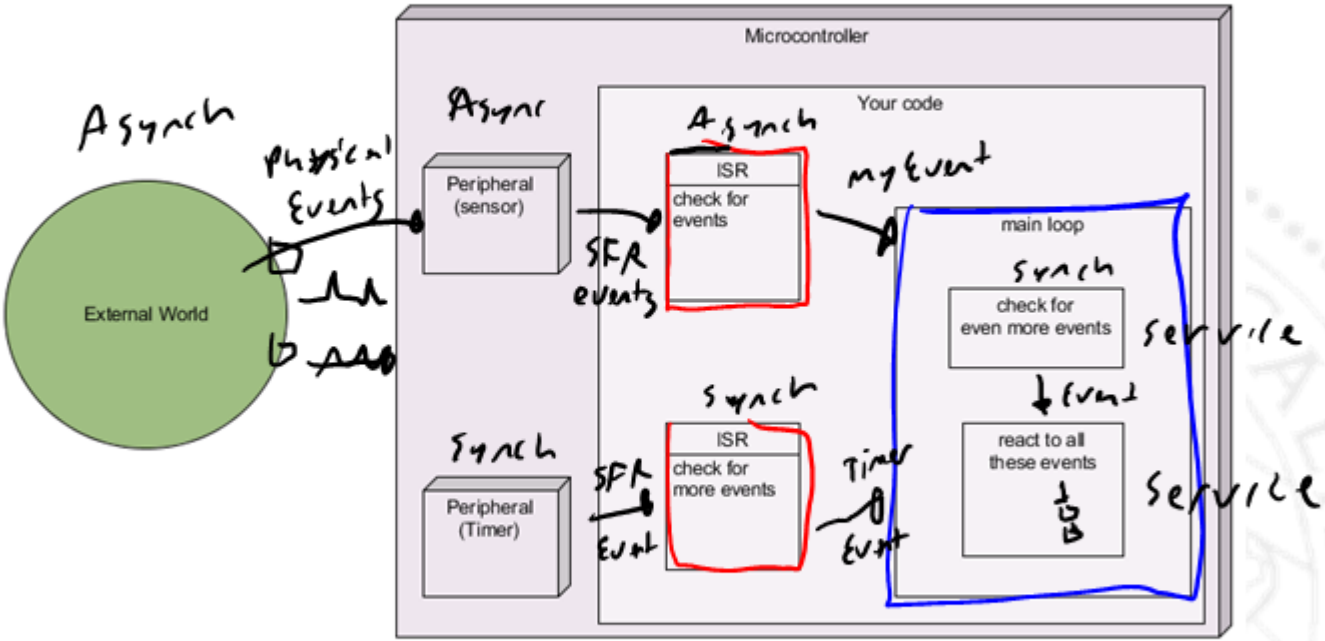
- But some code should run only at specific times:
  - Periodic sensor monitoring ✓
  - React to warnings ✓
  - Take advantage of opportunities ✓
  - Respond to requests ✓

Events / Services

Asynchronous vs Synchronous

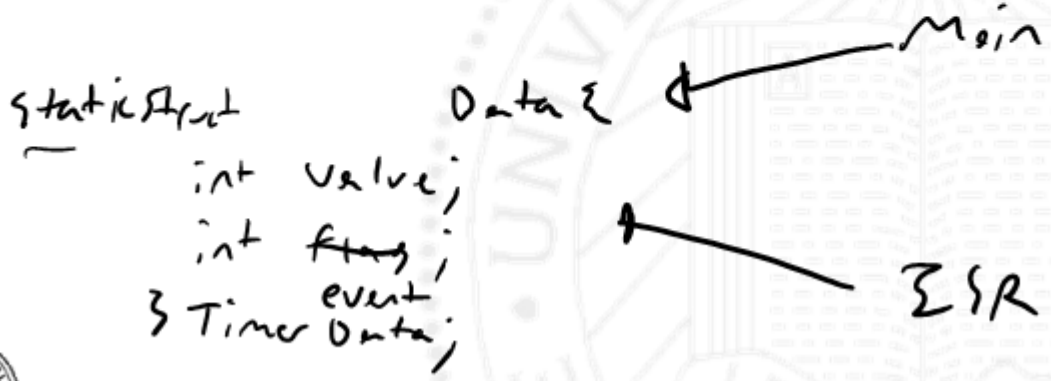
- Synchronous:
  - Code runs at periodic intervals
  - Or over a sequence of inputs (eg chars in a string)
- Asynchronous:
  - State machine is run when an event occurs
    - Is "fed" an event
    - Called by some other process
- You use both in this class!

Events in your code



In code

- Events are signaled with high-scope variables
  - Flags, for specialized event checkers
    - Event checker sets flag to 1
    - Main loop polls flag
      - Reacts if appropriate, then sets flag to 0



Piazza Poll: Which processes are synchronous?

```
static int event = 0;

int main(void)
{
    while(1){
        event = runProcessA();
        if(event){
            runProcessB(event);
        }
    }
}

void __ISR(_TIMER_1_VECTOR, IPL4AUTO) Timer1Handler(void){
    eventA = runProcessC();
}

void __ISR(_CN_1_VECTOR, IPL4AUTO) ChangeNotification1Handler(void){
    eventA = runProcessD();
}
```

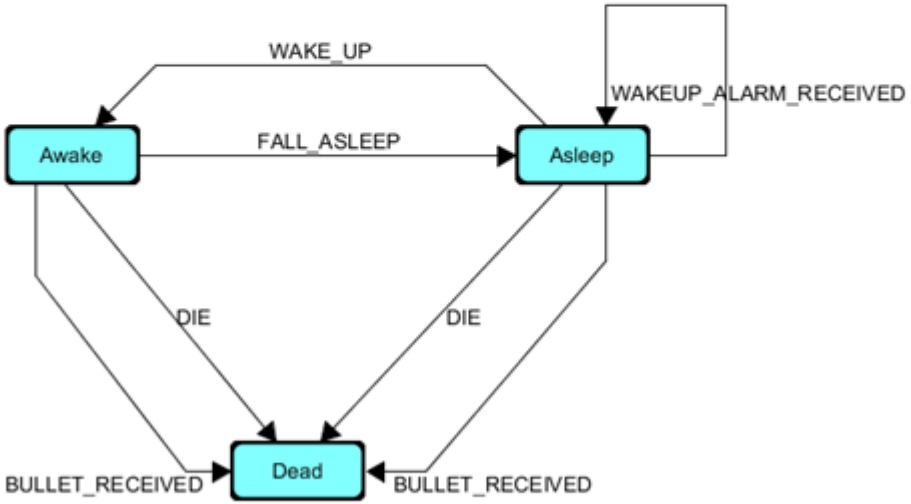
States and Events

States vs Events

- State:
  - Temporary but not instantaneous property of a system
    - For example: Sleeping, Awake, Dead
- Events:
  - Instantaneous
    - Can be internally caused changes in state:
      - For example: WAKE_UP, FALL_ASLEEP, DIE
    - Or momentary occurrences that don't necessarily cause changes in state:
      - For example, WAKEUP_ALARM_RECEIVED
    - Or external causes of changes:
      - For example: BULLET_RECEIVED

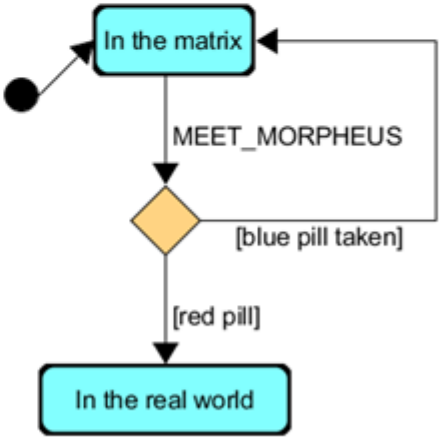
State Machine:

- A set of states, linked by a set of transitions (generally events):



State Machine Diagrams

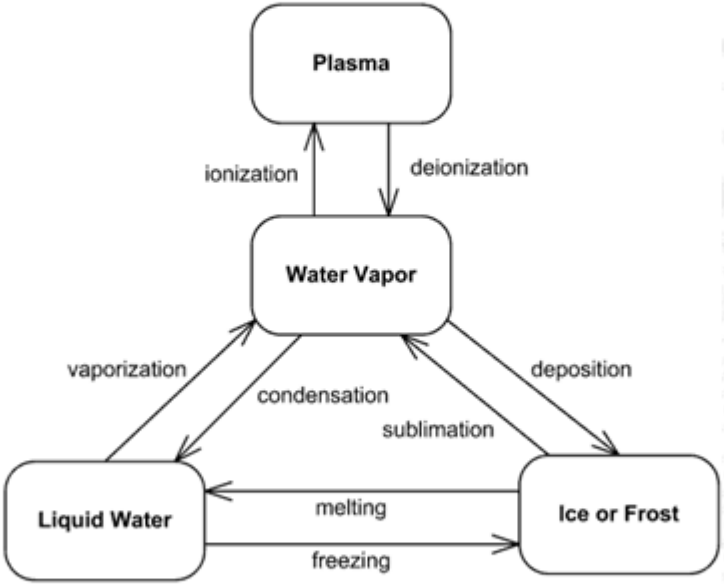
- A way of drawing state machines
- Standardized in Unified Modeling Language notation (UML)
  - States are bubbles
    - Starting state denoted by dot
  - Transitions are arrows
  - EVENTS are written on arrows
    - Can have / associated actions
  - Conditional transitions are diamonds
    - Take path of true [guard condition]



State Machines as Models

- Model of the physical world:
  - Used to abstract (simplify) a complex system
    - Example: Button is up, button is down
- Software Design Model
  - Used to plan, document, and build code
    - Or circuits, or machines
    - Example: Button event checker from last lecture
  - Easy to observe and debug (SO USEFUL)

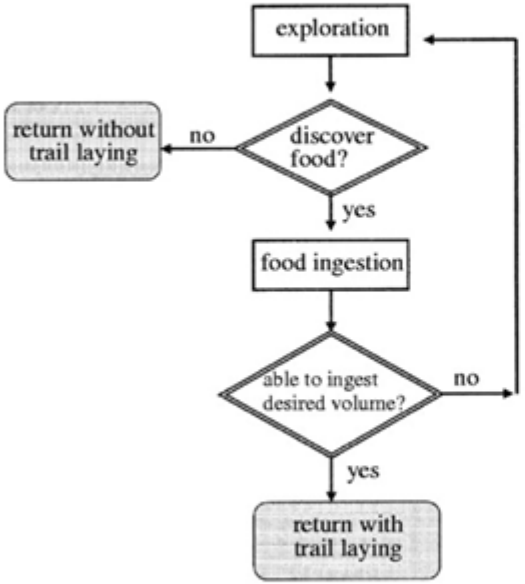
Physical Models



https://www.uml-diagrams.org/examples

Max Lichtenstein

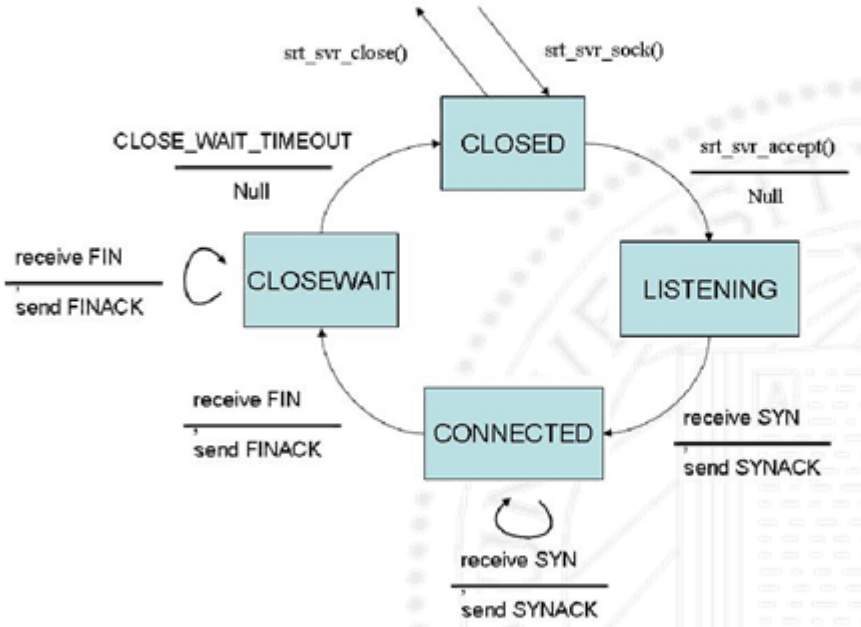
Behavioral Models



The principles of collective animal behaviour
D.J.T Sumpter Phil. Trans. R. Soc. B 29 January 2006

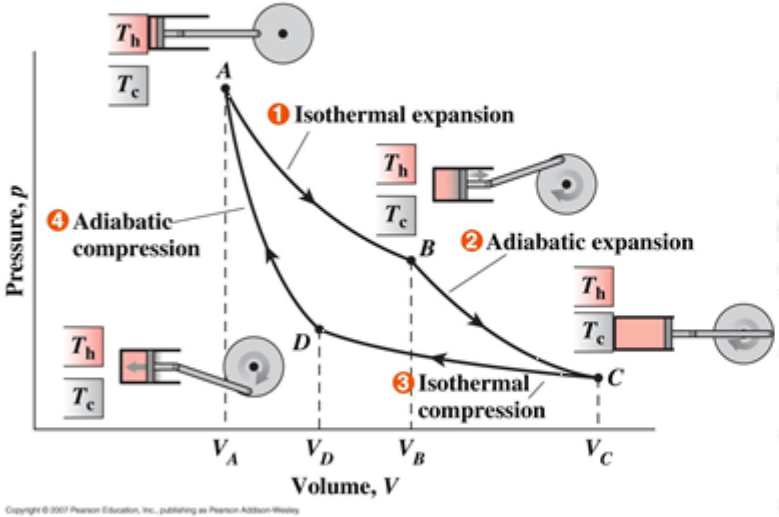
Max Lichtenstein

Code Design Models

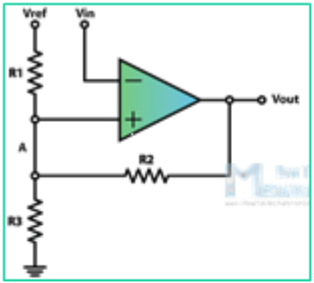


http://www.cs.dartmouth.edu/~campbell/cs60/srt.html

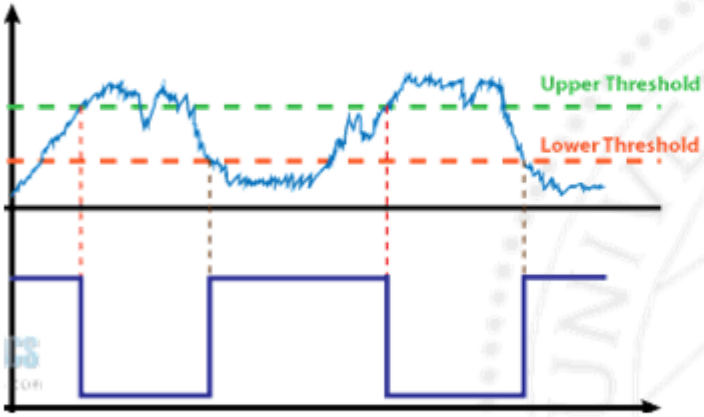
Physical Design Models



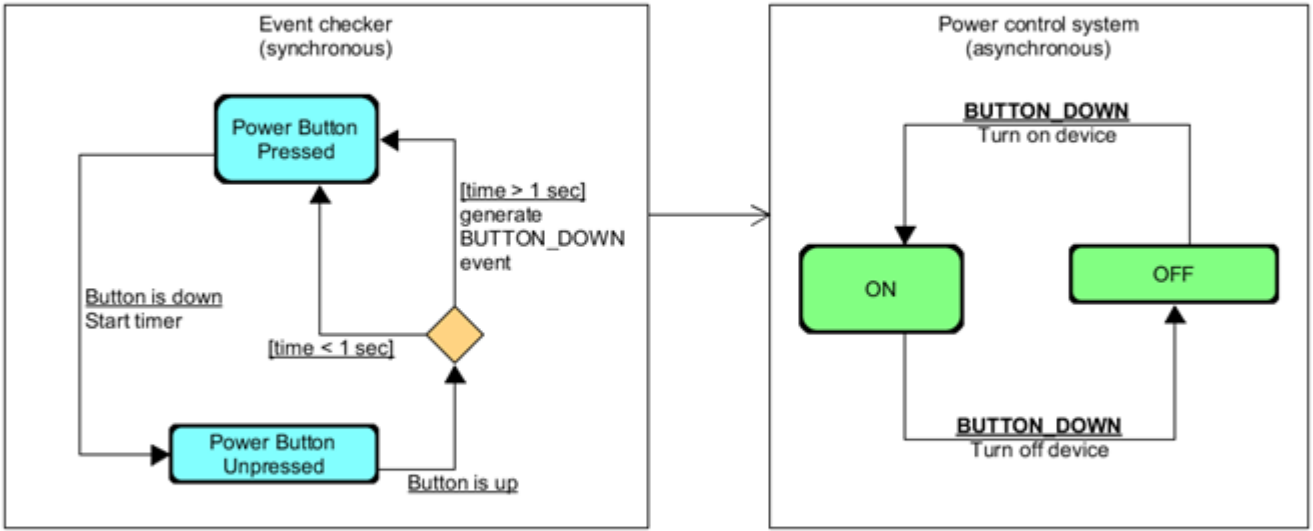
Max Lichtenstein



Electrical Design Models

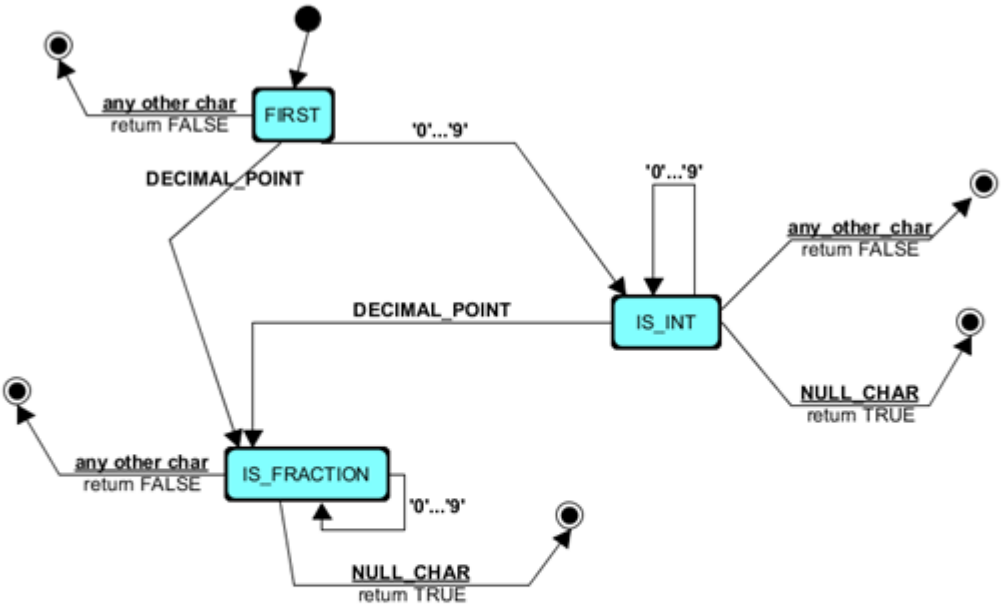


Asynchronous vs Synchronous SMs



State machines can both
generate **and** *react to* events!

Piazza Poll: IsNumber(char * token)



This state machine operates over the chars in a string to determine if the string represents a number.

Which token(s) will return "FALSE"?

A: "3.14159" B: "192.168.1.1"

C: "-.000" D: "12.5%"