CMPE-013/L

Introduction to "C" Programming

Max Lichtenstein

What is the result of accessing free()'d data?

free does not rec. delete or clear data

```
int * bird;
bird = malloc(sizeof(int));
if(bird != NULL){
    *bird = 1973;
}

free(bird);

printf("bird's contents after free: %d", *bird);

while(1);
return 0;
```

A: 0 ✓ **B**: 1973 ✓

C: garbage ✓ **D**: Runtime error ✓

E: All of the above are possible

Roadmap

- Announcements, Lab5 notes
- Intro to Embedded Code
  - Interacting with the outside world
  - Special Function Registers (SFRs)
  - What embedded software needs
- Break
- Event-driven programming
  - Polling,
  - Blocking code
- State Machines (?)
- ISRs?

Announcements

- Revote on push-based reports: Overwhelming pass

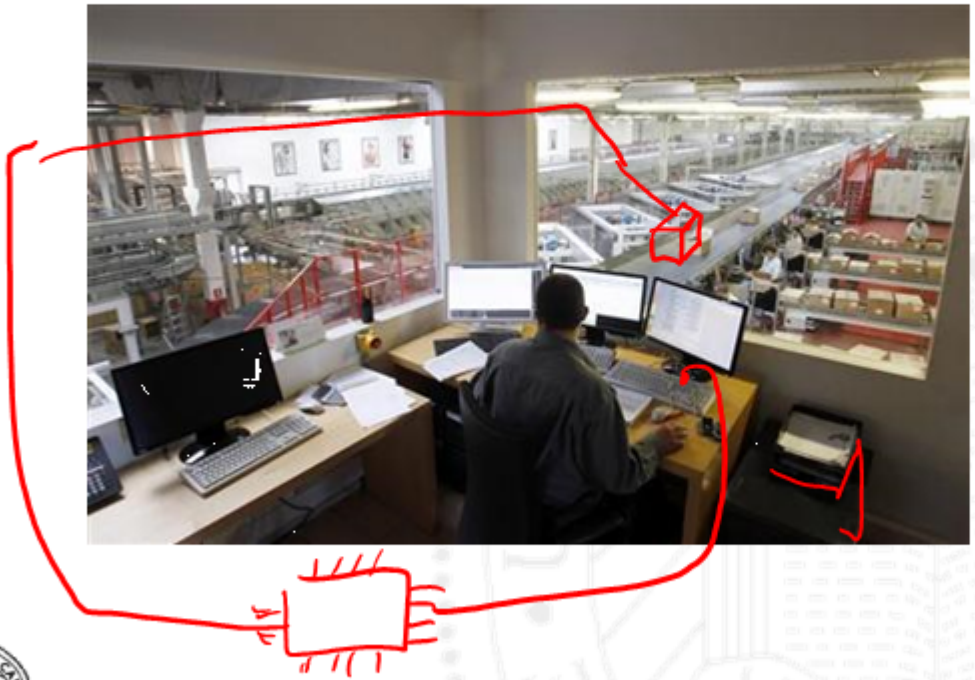
Lab 5 notes

- Accessing free()'d data
- ~~Efficiency Issues~~

  Work Smarter Not Harder
- Only ~50% of coding is coding
  - Take breaks
  - Drawing time
  - Ask

Embedded code intro

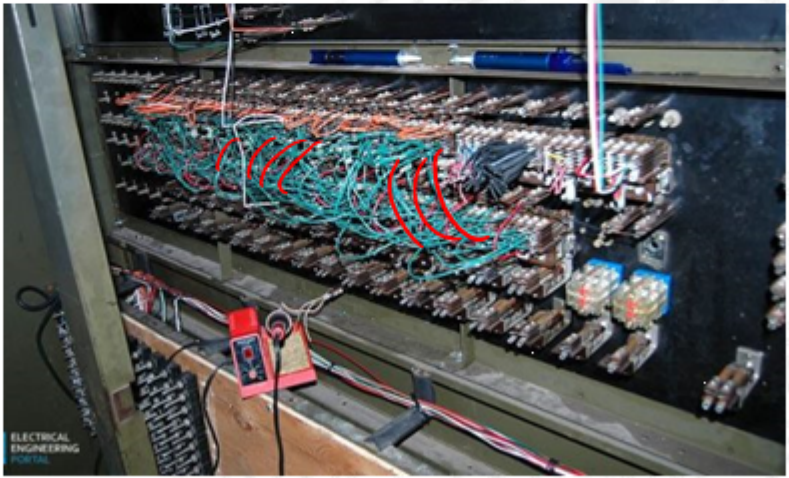
How does code interact with the world?



How does code interact with the world?

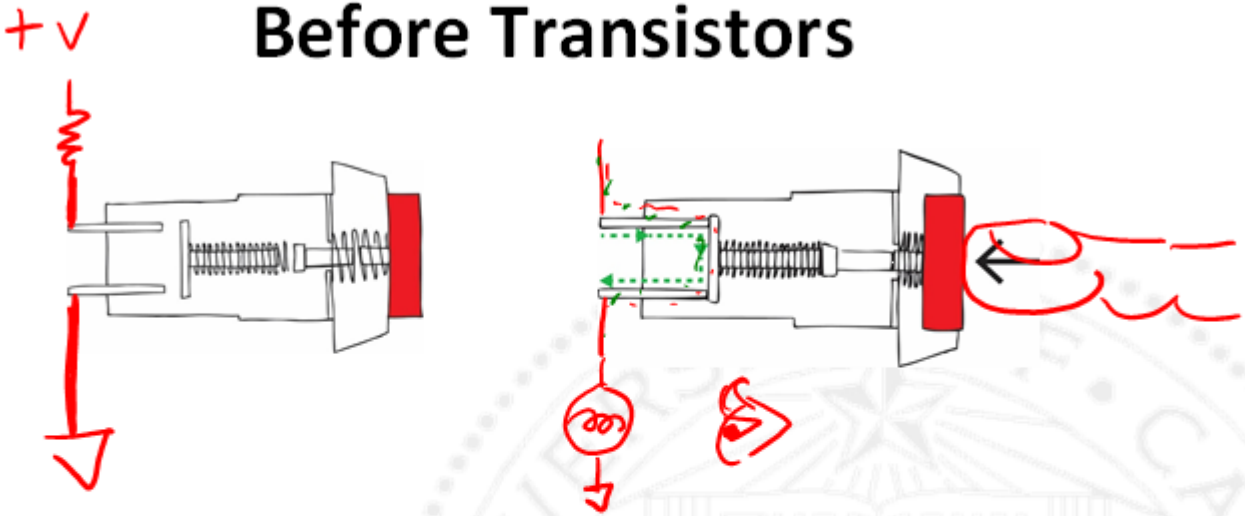
```
/*****************************************************************************
  Function: RC_Init

  Parameters
      RCpins: an unsigned short with a 1 in each position to set the pin as an RC
              servo pin, should be a bitwise OR of the #define'd RC_PORTxxx pins.

  Returns
      char: SUCCESS or ERROR

  Description
     Initializes the RC_Servo subsystem, sets each pin as a digital output, and
     sets the uptime for each pin at 1.5msec, with a period of 20msec.

  Notes: Uses TIMER4 with a rollover.

  Author: Gabriel Hugh Elkaim, 2011.12.15 16:42
 *****************************************************************************/
char RC_Init(unsigned short int RCpins) {
    char i, curPin;
    unsigned short int CurrentTime;
    dbprintf("\n\rInitializing the RC Servo Module.");
    // Check if inputs are in range, and if already initialized
    if ((RCpins == 0x0000) || (RCpins > 0x2FF0) || (RCstate != off)) {
        return ERROR;
    }
    RCstate = init;
    // Go through input and set each RC pin direction and force to low
    for (i = 0; i < RCPINCOUNT; i++) {
        curPin = (1 << i);
        if (RCpins & curPin) {
            RCpinMap[numRCPins] = i;
            numRCPins++;
            RC_upTime[i] = SERVOCENTER;
            *RC_TRISCLR[i] = rcBitsMap[i]; // Sets pin direction to output
            *RC_LATCLR[i] = rcBitsMap[i]; // Forces pin to low state
            dbprintf("\n\rEnabling pin: 0x%X", curPin);
        }
```

The old way



Max Lichtenstein

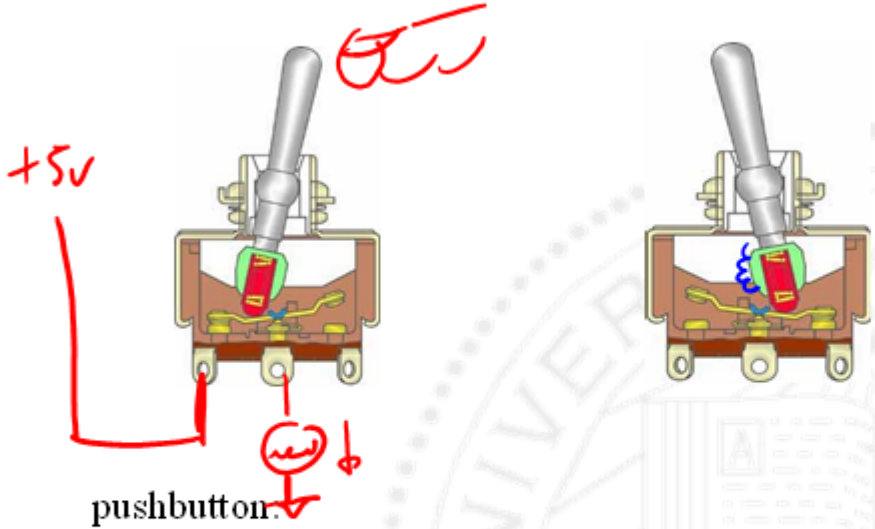
Before Transistors




```
pushbutton:
if (something presses A){
      current can flow on wire B;
}
if (something releases A) {
      current cannot flow on wire B;
}
```

Max Lichtenstein

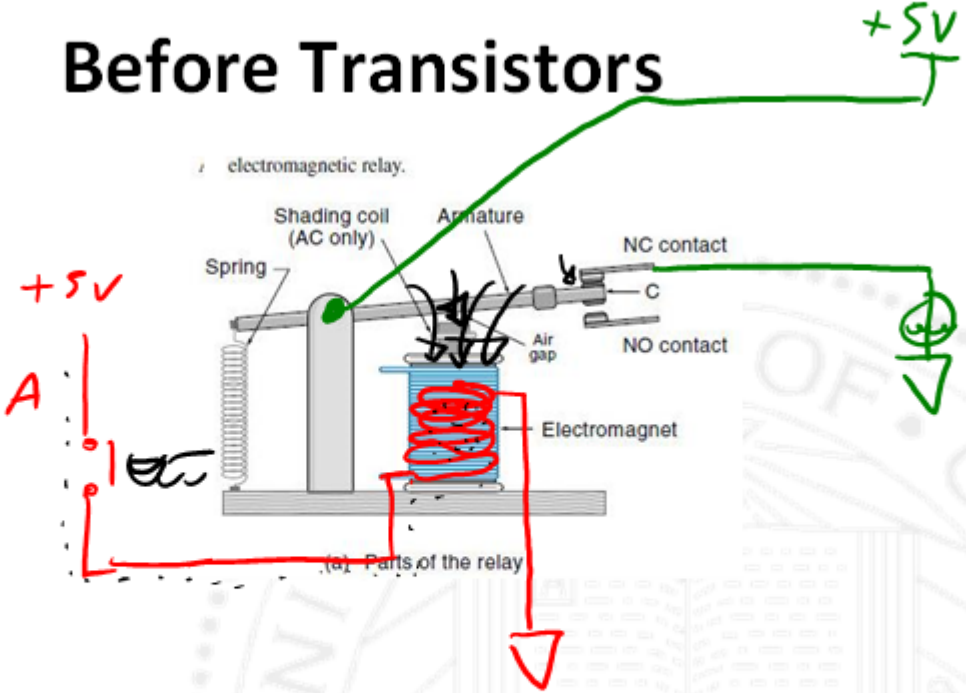
Before Transistors



pushbutton

```
if (A is switched left){
      current can flow on wire B;
}
If (A is switched right){
      current cannot flow on wire B;
}
```

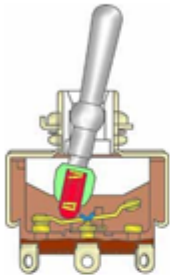
Before Transistors

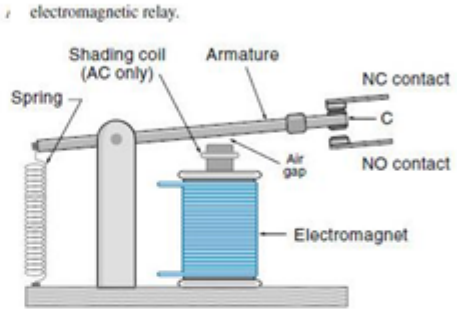


Relay:

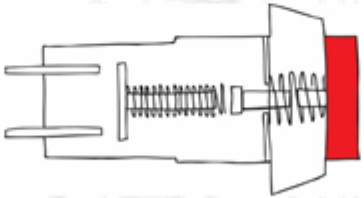
```
if (current flows on wire A){
        current can flow on wire B;
}
if (current does not flow on A){
        current cannot flow on B;
}
```

Before Transistors

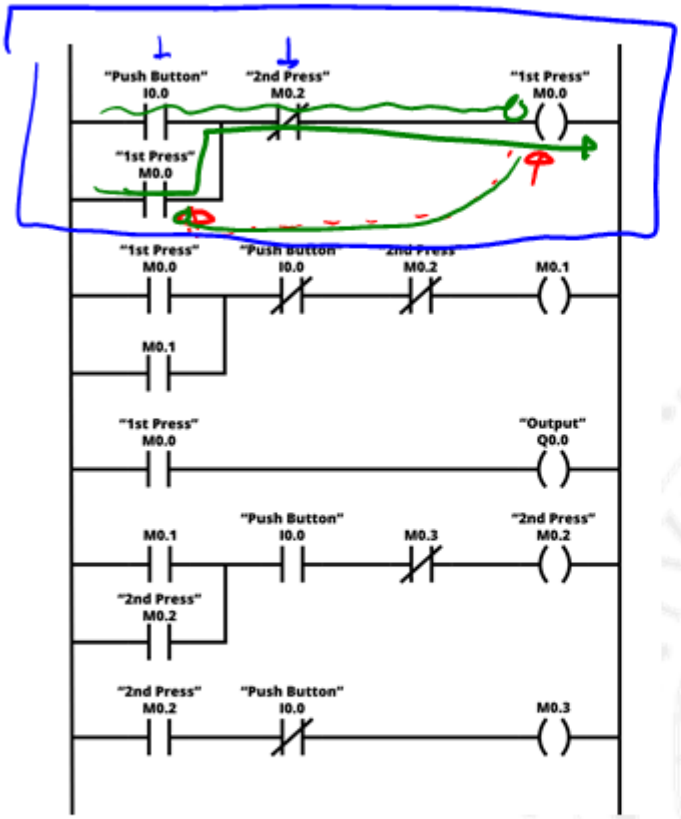




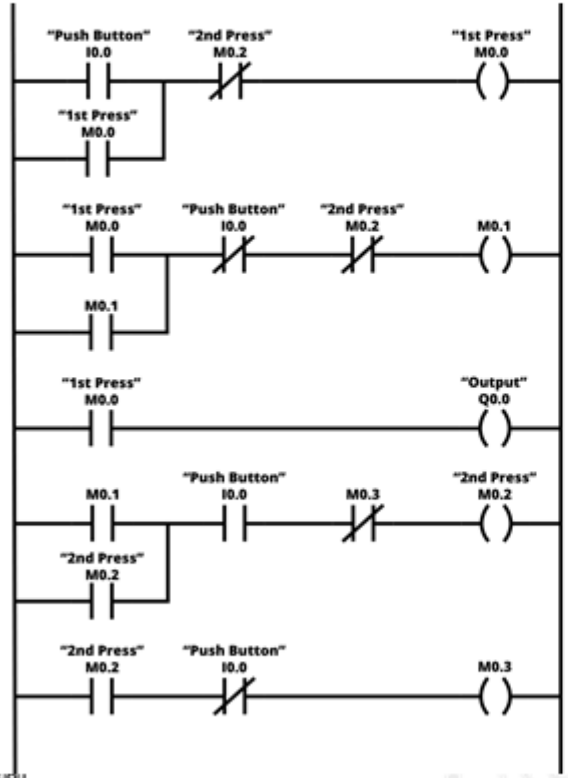
(a) Parts of the relay

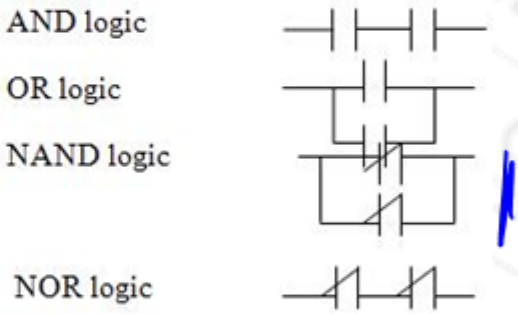


Ladder Logic



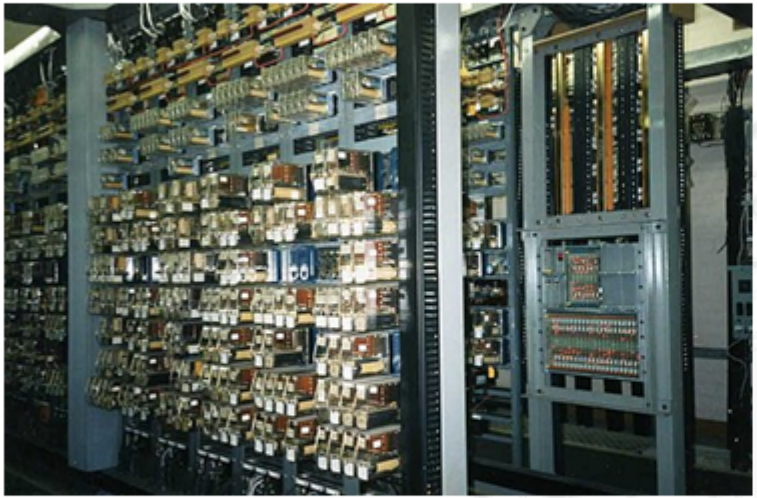
Ladder Logic





Flip flops
Memory
ALU
Control system

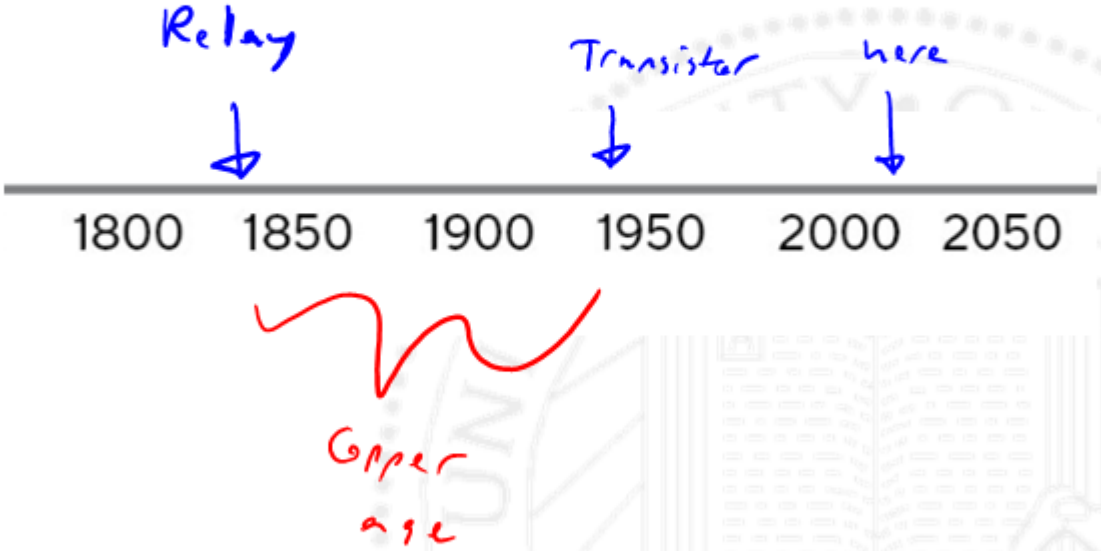
Ladder Logic



<-Turing complete!*

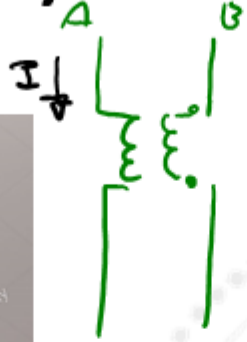
* up to resource limits

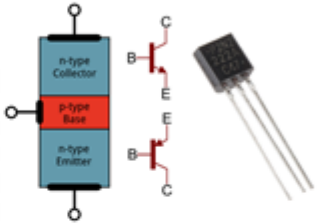
A new age



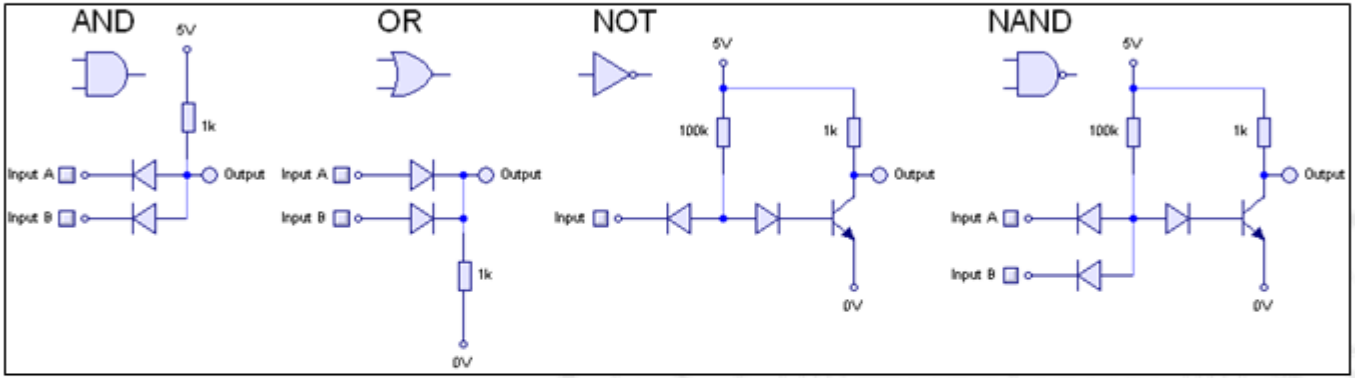
Max Lichtenstein

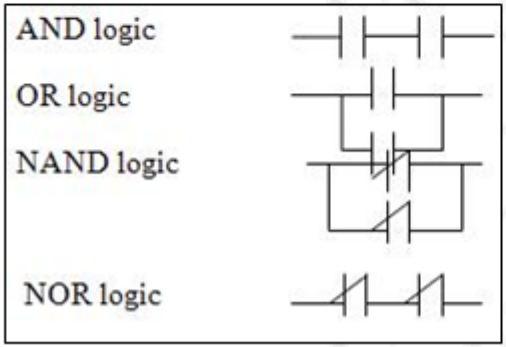
Relays vs Transistors

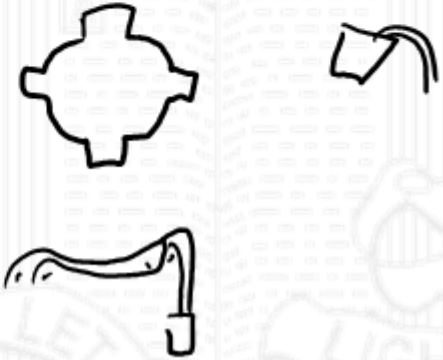




Ladder Logic vs Transistor Logic



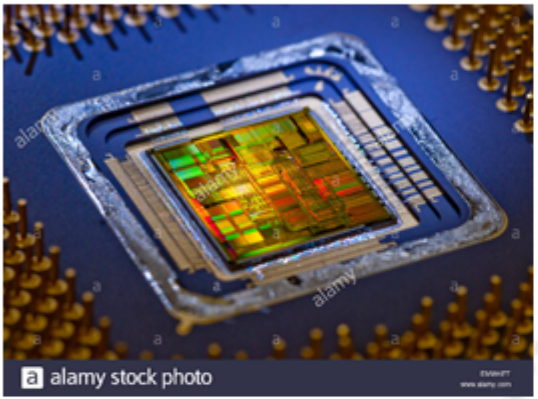




Max Lichtenstein

A new age

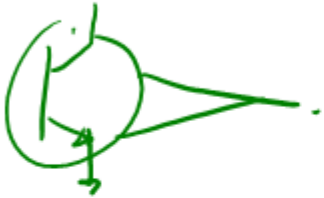
(same as the old age*)

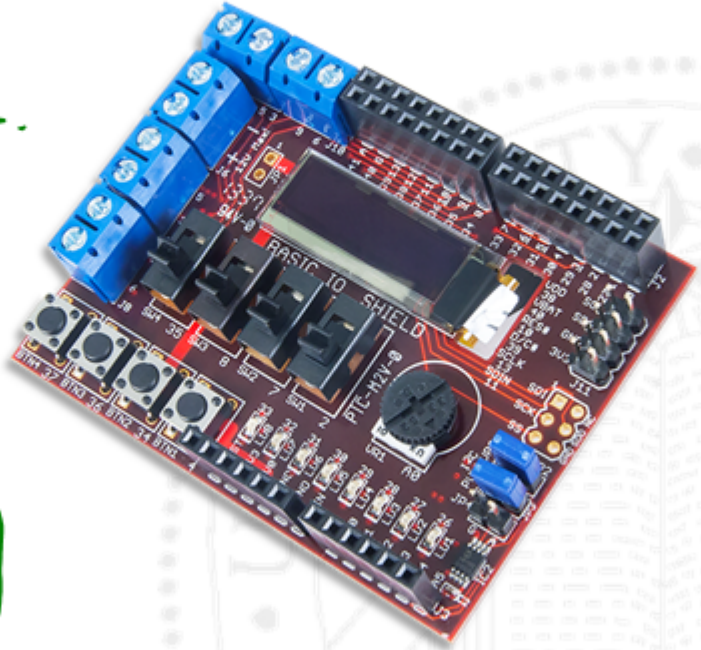


Max Lichtenstein

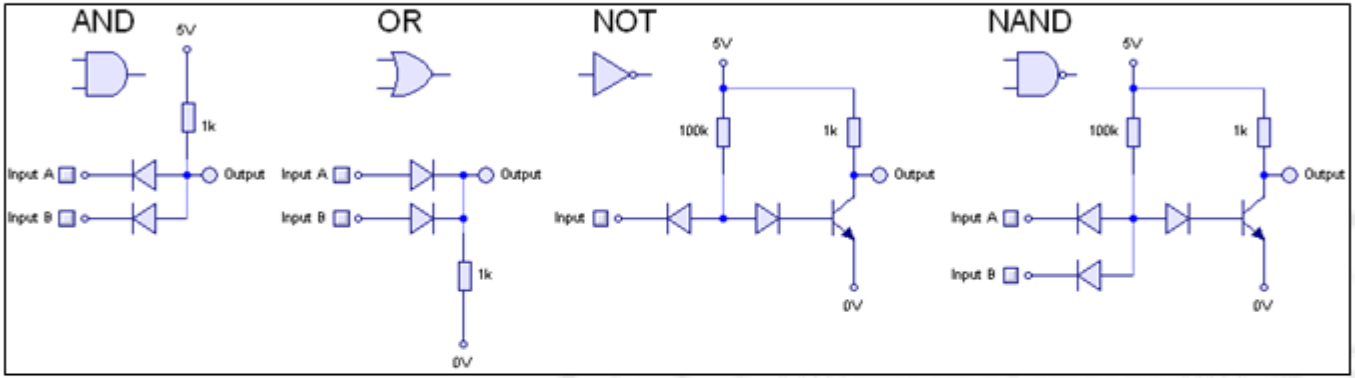
The modern way:

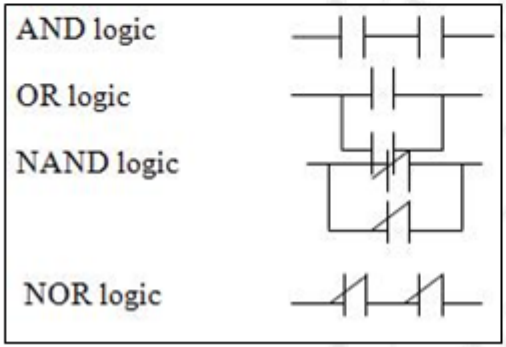
tiny voltages, big effects





Ladder Logic vs Transistor Logic





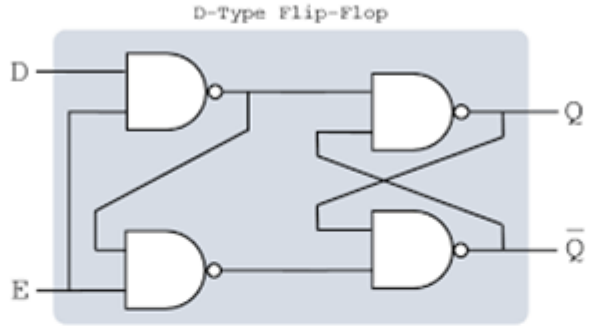
IO Example:

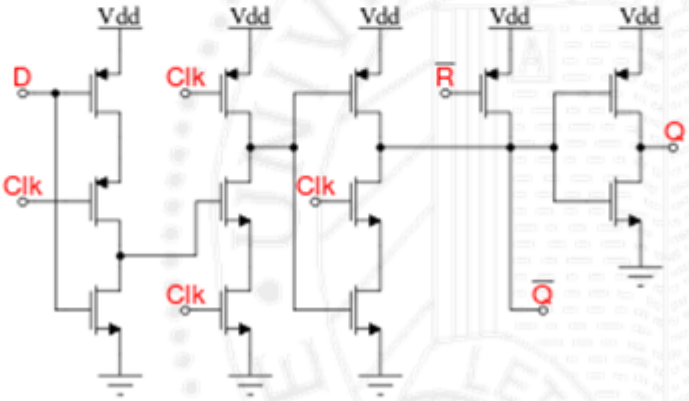
```
//Set LED1 to be output:
TRISE &= 0b11111110;
//set Button1 to be input:
TRISF |= 0b00000010;
```

SFRs

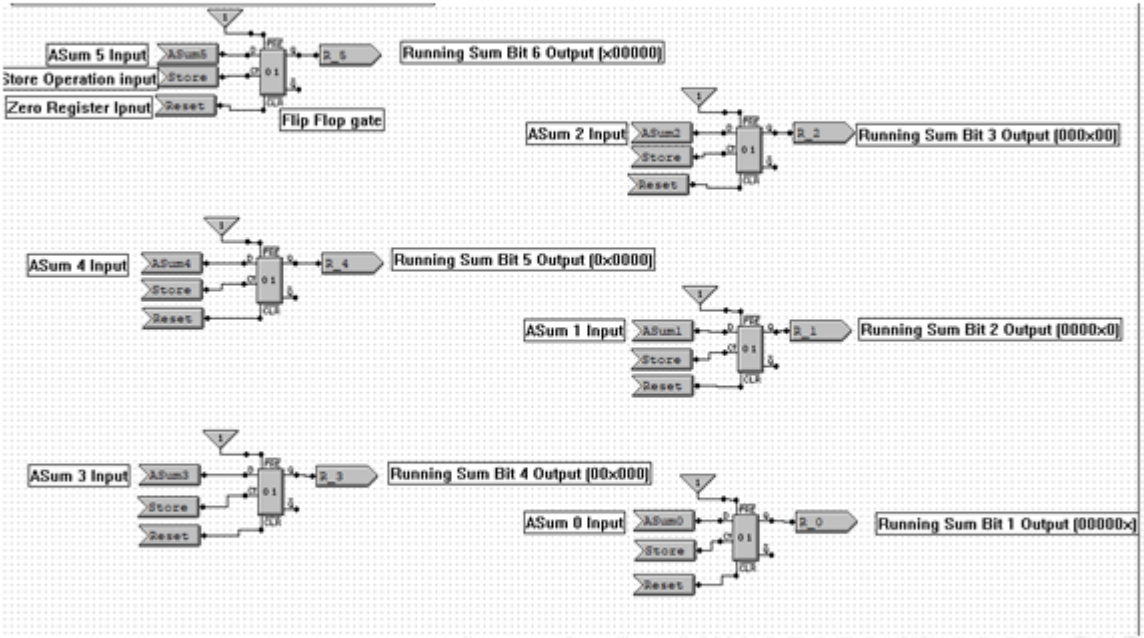
```
while (1) {
    //Set LED1 to whatever BTN1's state is:
    if ((PORTF & 0b0010) != 0) {
        LATE = 0b0001;
    } else {
        LATE = 0b0000;
    }
}
```

1s and 0s are voltages

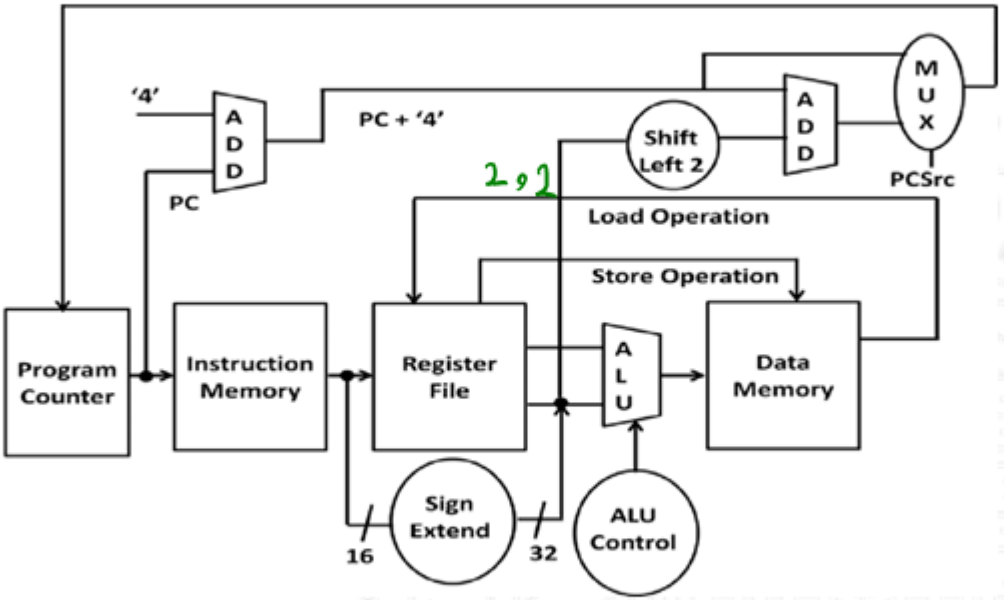




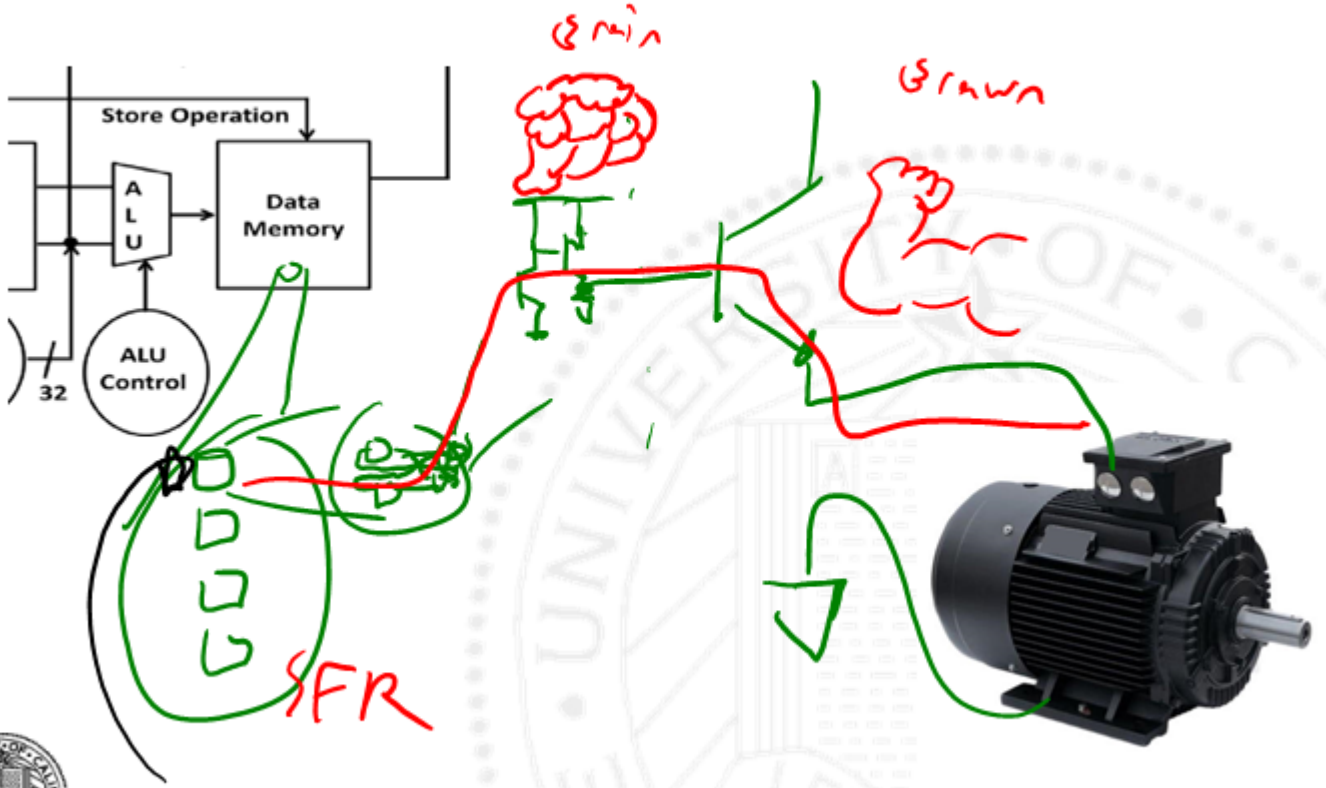
Registers' memories are voltages



Computers' memories are voltages

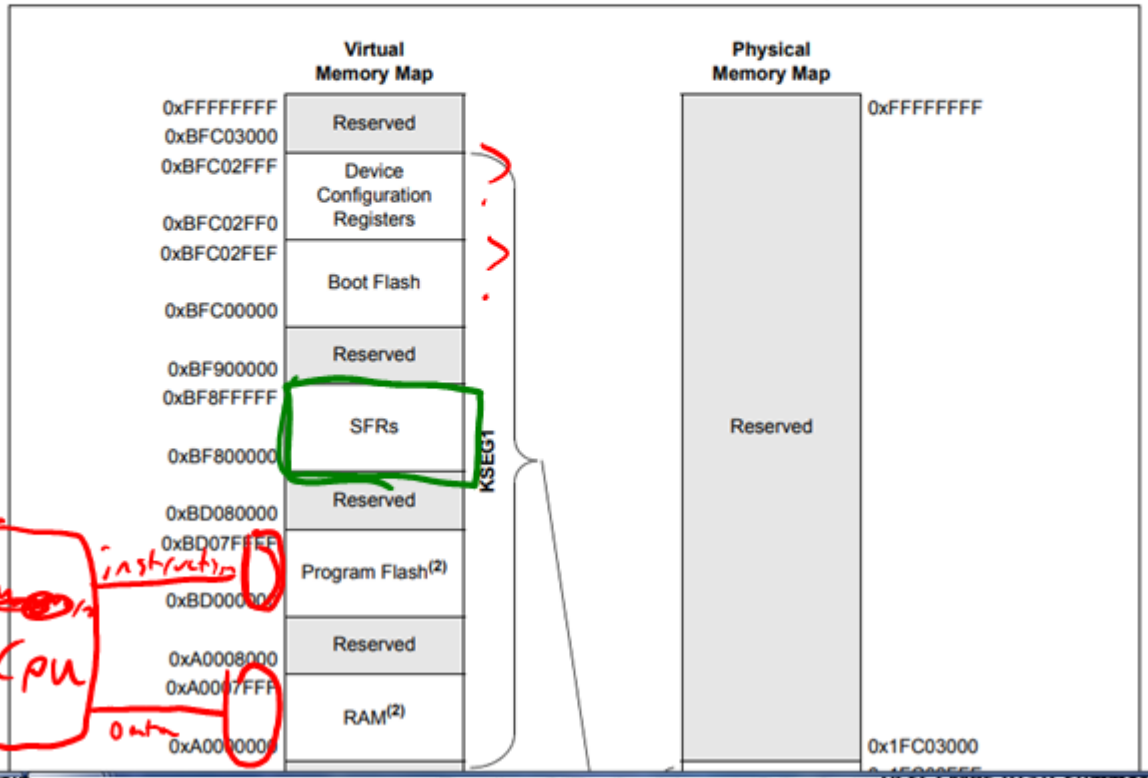


Can memory run a motor?



SFRs:

FIGURE 4-6: MEMORY MAP ON RESET FOR PIC32MX340F512H, PIC32MX360F512L, PIC32MX440F512H AND PIC32MX460F512L DEVICES(1)



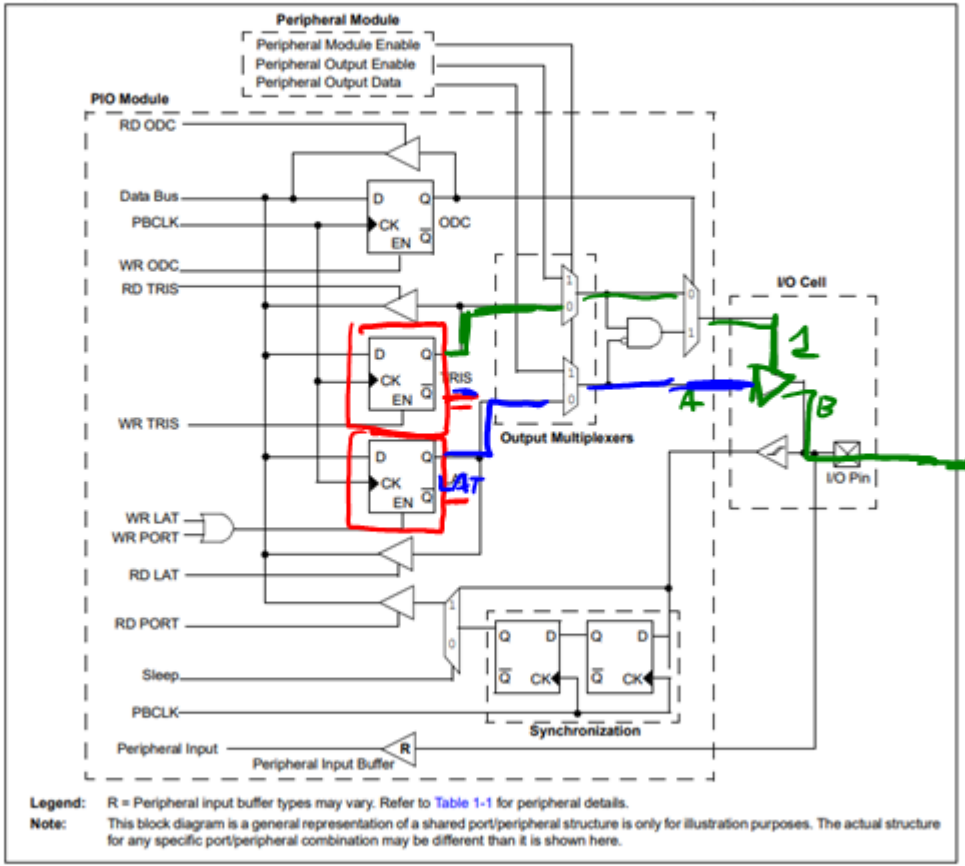
SFRs:

- Special locations of memory that can interact with the outside world
  - Or, interact with the processor's control system
- Can be accessed via variables in <xc.h>

PIC32MX340F512H

SFR example: TRIS and LAT

FIGURE 12-1: BLOCK DIAGRAM OF A TYPICAL MULTIPLEXED PORT STRUCTURE



Legend: R = Peripheral input buffer types may vary. Refer to Table 1-1 for peripheral details.

Note: This block diagram is a general representation of a shared port/peripheral structure is only for illustration purposes. The actual structure for any specific port/peripheral combination may be different than it is shown here.

0 = output

1 = input

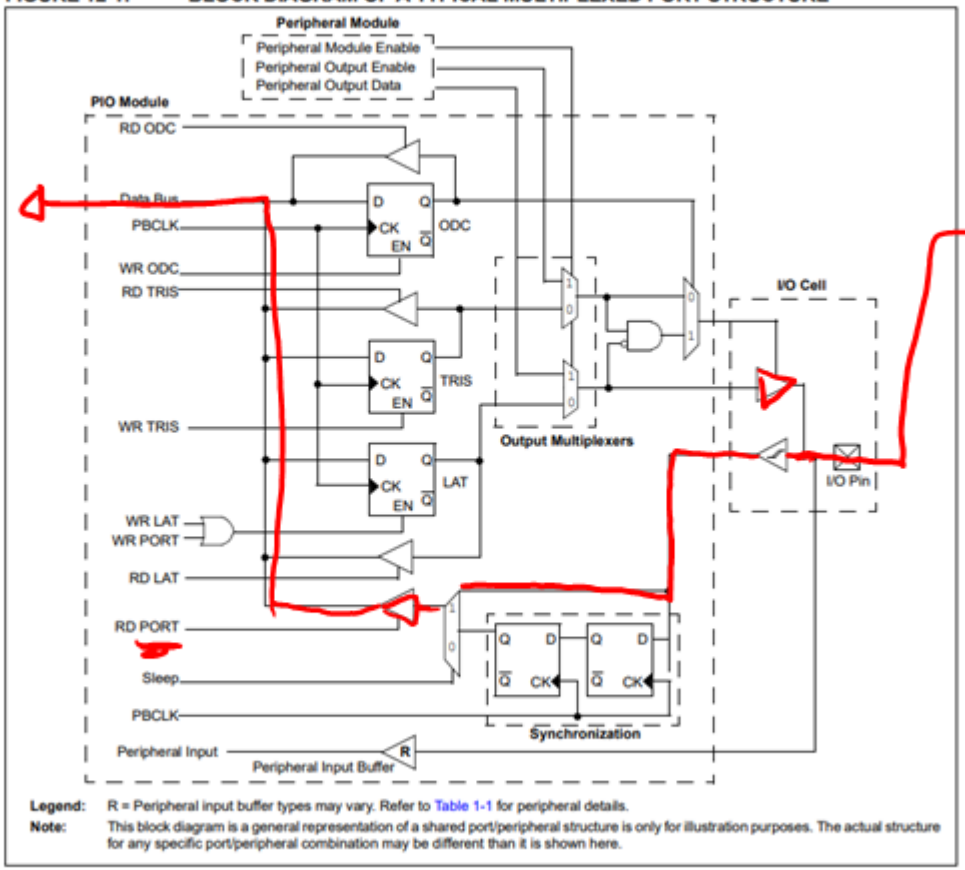
I = 1

O = 0

Max Lichtenstein

SFR example: TRIS and PORT

FIGURE 12-1: BLOCK DIAGRAM OF A TYPICAL MULTIPLEXED PORT STRUCTURE

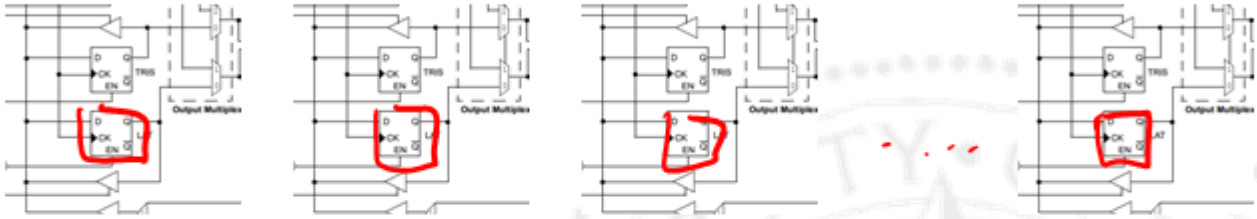


Legend: R = Peripheral input buffer types may vary. Refer to Table 1-1 for peripheral details.

Note: This block diagram is a general representation of a shared port/peripheral structure is only for illustration purposes. The actual structure for any specific port/peripheral combination may be different than it is shown here.

SFRs:

LATE Register



TriSE = 0

Example, Revisited:

```
//Set LED1 to be output:
TRISE &= 0b11111110;
//set Button1 to be input:
TRISF |= 0b00000010;

while (1) {
    //Set LED1 to whatever BTN1's state is:
    if ((PORTF & 0b0010) != 0) {
        LATE = 0b0001;
    } else {
        LATE = 0b0000;
    }
}
```

Mask

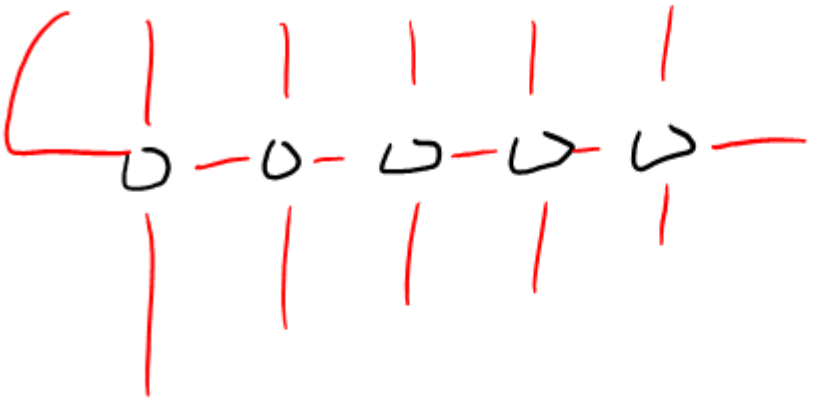
← TrisE bit 0 = 0 output

← set TrisF bit 1 = 1 input

← read portf bit 1

set output to TRisE bit 0

Max Lichtenstein



SFR documentation

TABLE 4-28:

Virtual Address (BF88_#)	Register Name	Bit Range	Bits 23/7	22/6	21/5	20/4	19/3	18/2	17/1	16/0	All Resets
6100	TRISE	31:16	—	—	—	—	—	—	—	—	0000
		15:0	TRISE7	TRISE6	TRISE5	TRISE4	TRISE3	TRISE2	TRISE1	TRISE0	00FF
6110	PORTE	31:16	—	—	—	—	—	—	—	—	0000
		15:0	RE7	RE6	RE5	RE4	RE3	RE2	RE1	RE0	xxxx
6120	LATE	31:16	—	—	—	—	—	—	—	—	0000
		15:0	LATE7	LATE6	LATE5	LATE4	LATE3	LATE2	LATE1	LATE0	xxxx
6130	ODCE	31:16	—	—	—	—	—	—	—	—	0000
		15:0	ODCE7	ODCE6	ODCE5	ODCE4	ODCE3	ODCE2	ODCE1	ODCE0	0000

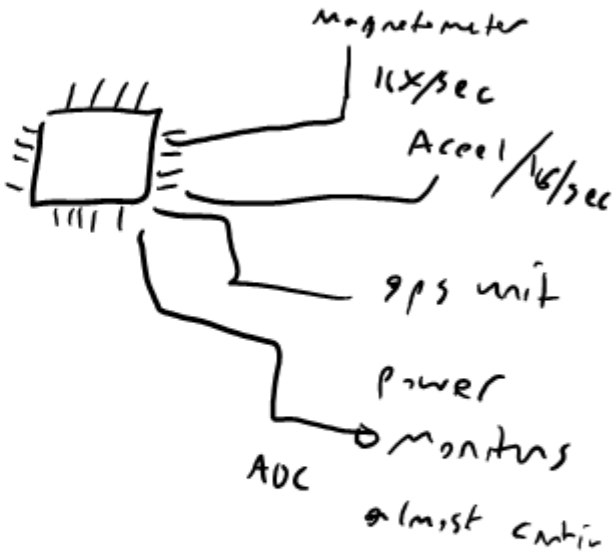
What other SFRs are there?

SO MANY!

- Internal config:
  - Speed
  - Power modes
  - Interrupts
  - Pin mapping
  - Error handling
- Internal functions:
  - Random numbers and crypto
  - Timers
  - Fast copying
- External functionality:
  - Digital IO
  - Read analog voltage
  - PWM (output "analog")
  - Manage communication interfaces
    - eg UART, I2C, SPI, USB, Ethernet
  - Count and time events
- AND MORE....

Special things in embedded world

— Speed



Piazza Poll

- We want a system that just counts the number of times someone presses a button. Will the following code work??

```
printf("How many times has the button been pressed?\n");
int button_count = 0;
while (1) {
    if (PORTF & 1) {
        button_count++;
    };
    printf("%d times\n", button_count);
}
```

A: It's fine
B: It might work under some circumstances
C: It's totally broken

BREAK

Max Lichtenstein

Piazza Poll Results

Bit Banging

- Set

Make = 1

1 ≠ 11

- Toggle

^ bitwise XOR

Set bit 1 of T1CON

T1CON 0011010

T1CON = 1

T1CON |= 0000001

Toggle bit 16 of T1CON?

T1CON ^= 0b1000 0000 0000 0000;

Max Lichtenstein

Bit Banging

- Clear
  - Masking

Clear bit 2 of T1CON;

Set to 0

T1Con &= 1111...0
= 0xFFFE

- Toggle

T1Con.bits.2 = 0;

Bit Banging

- Shift

Set bit 16 of T1con

T1conv |= (1<<16)

- Masking

Polling

- Monitoring events by reading them over and over again

Polling:

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

What's wrong with this?

Polling:

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

        for (i = 0; i < 8000000; i++); //do nothing for ~1 second
    }
```

What's wrong with ***this***?

Polling:

```
case STATE_MACHINE:
    printf("How many times has the button been pressed?\n");

    enum { PRESSED, UNPRESSED};
    int previousState = UNPRESSED;

    while (1) {
        if (BTN4 && (previousState == UNPRESSED)) {
            button_count++;
            previousState = PRESSED;  };

        if ((!BTN4) && (previousState == PRESSED)) {
            previousState = UNPRESSED;     }

        //print, but only occasionally
        i++;
        if (i % 8000000 == 0) printf("%d times\n", button_count);
    }
}
```

Surely nothing is wrong with *this?*

Max Lichtenstein

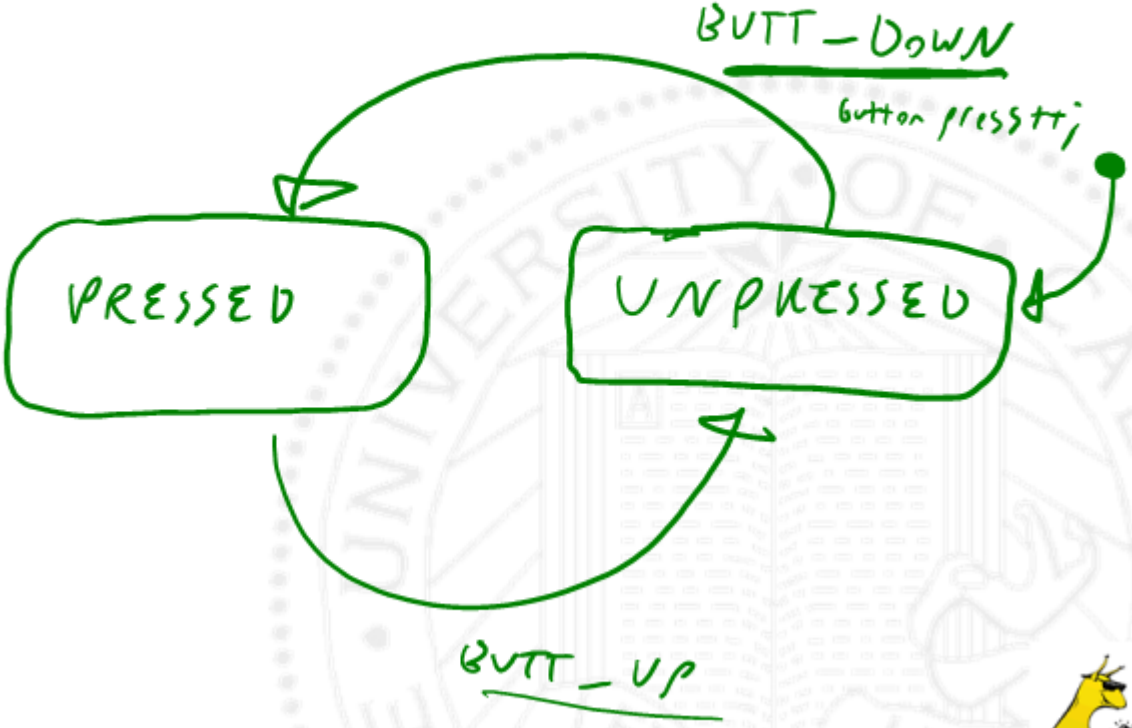
Simple State Machine

States

Events

effects

guard conditions



Max Lichtenstein

Interrupts