EE 109 Unit 4

Microcontrollers (Arduino) Overview

BIT FIDDLING

Using software to perform logic on individual (or groups) of bits

4.2 BIT FIDDLING

• Suppose we want to place the binary value 00111010 into a char variable, v [i.e. char v;]
  – We could convert to decimal on our own (58_{10})
    v = 58;
  – All compilers support hexadecimal using the _____ prefix
    v = 0x3a;
  – Our Arduino compiler supports binary using the _____ prefix
    v = 0b00111010;
• Important note: Compilers convert EVERYTHING to equivalent __________. The 3 alternatives above are equivalent because the compiler will take all 3 and place 00111010 in memory.
  – Use whichever base makes the most sense in any given situation
  – It is your (the programmer’s) ________...compiler will end up converting to binary once it is compiled

4.3 Numbers in Other Bases in C/C++

• Suppose we want to place the binary value 00111010 into a char variable, v [i.e. char v;]
  – We could convert to decimal on our own (58_{10})
    v = 58;
  – All compilers support hexadecimal using the _____ prefix
    v = 0x3a;
  – Our Arduino compiler supports binary using the _____ prefix
    v = 0b00111010;
• Important note: Compilers convert EVERYTHING to equivalent __________. The 3 alternatives above are equivalent because the compiler will take all 3 and place 00111010 in memory.
  – Use whichever base makes the most sense in any given situation
  – It is your (the programmer’s) ________...compiler will end up converting to binary once it is compiled

4.4 Modifying Individual Bits

• Suppose we want to change only a single bit (or a few bits) in a variable [i.e. char v;] without changing the other bits
  – Set the LSB of v to 1 w/o affecting other bits
    • Would this work? v = 1;
  – Set the upper 4 bits of v to 1111 w/o affecting other bits
    • Would this work? v = 0xf0;
  – Clear the lower 2 bits of v to 00 w/o affecting other bits
    • Would this work? v = 0;
  – ____!!! Assignment changes _________ bits in a variable
• Because the smallest unit of data in C is a byte, manipulating individual bits requires us to use BITWISE LOGICAL OPERATIONS.
  – Use ______ operations to clear individual bits to 0
  – Use ______ operations to set individual bits to 1
  – Use XOR operations to invert bits
  – Use AND to isolate a bit(s) value from the others in the register
Bitwise Logical Operations

- ANDs can be used to control whether a bit passes changed or a '0' is produced (i.e. AND's can force a bit to _____)
- ORs can be used to control whether a bit passes unchanged or a '1' is produced (i.e. OR's can force a bit to _____)
- XORs can be used to control whether a bit passes unchanged or is inverted/flipped

![Logic Gates Diagram]

<table>
<thead>
<tr>
<th>Ctrl</th>
<th>Bit</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ctrl</th>
<th>Bit</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

0 AND x = ___
1 AND x = ___
x AND x = ___

0 OR x = ___
1 OR x = ___
x OR x = ___

0 XOR x = ___
1 XOR x = ___
x XOR x = ___

Logical Operations

- Logic operations on numbers means performing the operation on each pair of bits

| 0xF0 AND 0x3C | 1111 0000 AND 0011 1100 |
| 0xF0 OR 0x3C | 1111 0000 OR 0011 1100 |
| 0xF0 XOR 0x3C | 1111 0000 XOR 0011 1100 |

Logical Operations

- The C language has two types of logic operations
  - Logical and Bitwise
  - Logical Operators (&&, ||, !)
    - Operate on the logical value of a FULL variable (char, int, etc.) interpreting that value as either True (non-zero) or False (zero)
      - char x = 1, y = 2, z;
      - Result is z = ____;
      - Why?
  - Bitwise Logical Operators (&, |, ^, ~)
    - Operate on the logical value of INDIVIDUAL bits in a variable
      - char x = 1, y = 2, z;
      - Result is z = ____;
      - Why?

Logical Operations

- Bitwise logic operations are often used for "bit fiddling"
  - Change the value of a bit in a register w/o affecting other bits
  - C operators: & = AND, | = OR,
    ^ = XOR, ~ = NOT
  - Examples (Assume an 8-bit variable, v)
    - Clear the LSB to '0' w/o affecting other bits
      - v = v & 0xfe; or equivalently
      - v = v & ~(0x01);
    - Set the MSB to '1' w/o affecting other bits
      - v = v | 0x80;
    - Flip the LS 4-bits w/o affecting other bits
      - v = v ^ 0x0f;
4.9 Changing Register Bits

- Bitwise logic operations can be used to change the values of individual bits in registers without affecting the other bits in the register.
  - Set bit 0 of v to a ‘1’
    \[ v = v | ________; \]
  - Clear the 4 upper bits in v to ‘0’s
    \[ v = v & ________; \]
  - Flip bits 4 and 5 in v
    \[ v = v ^ ______________; \]

4.10 Checking Register Bits

- To check for a given set of bits we use a bitwise-AND to isolate just those bits
  - The result will then have 0’s in the bit locations not of interest
  - The result will keep the bit values of interest

- Examples
  - Check if bit 7 of v = '1'
    \[ \text{if (v & 0x80 == 0x80) \{ code \} \text{ or } \text{if (v & 0x80) \{ code \} } \]
  - Check if bit 2 of v = '0'
    \[ \text{if (v & 0x04 == 0x00) \{ code \} \text{ or } \text{if (! (v & 0x04)) \{ code \} } \]
  - Check if bit 2:0 of v = "101"
    \[ \text{if ( (v & 0b00000111) == 0b00000101) \{ code \} } \]
  - Check if bit 5-4 of v = "01"
    \[ \text{if ( (v & 0x30) == 0x10) \{ code \} } \]

4.11 Short Notation for Operations

- In C, assignment statements of the form
  \[ x = x \text{ op } y; \]
- Can be shortened to
  \[ x \text{ op}= y; \]
- Example:
  \[ x = x + 1; \text{ can be written as } x += 1; \]
- The preceding operations can be written as
  \[ v| = 0x01; \]
  \[ v & = 0x0f; \]
  \[ v ^= 0b00110000; \]
4.13 Arduino Uno

- The Arduino Uno is a microcomputer development board based on the Atmel ATmega328P 8-bit processor.
- Most microcomputer manufacturers (Atmel, Freescale, etc.) produce small PC boards with their chips on them for engineers to experiment with and hopefully generate sales of the product.

4.14 Arduino Uno

- Arduino
  - An Italian company
  - They make numerous boards with different processors
  - Hardware and software are open source.
  - Very popular with hobbyists, due in a large part to their low cost.

4.15 Arduino Uno

- What’s on an Arduino Uno board?
  - Atmel ATmega328P microcontroller
  - 16MHz oscillator (i.e. clock signal generator)
  - USB interface
  - Power connector (can also be powered if connected to USB)
  - Connectors for I/O lines D0 – D13
  - I/O lines A0 – A5

4.16 Arduino Uno

- Arduino Unos can be stacked with "shield" boards to add additional capabilities (Ethernet, wireless, D/A, LCDs, sensors, motor control, etc.)
### ARDUINO PORTS AND PINS

**Flashback to Week 1**

- Recall the computer interacts with any input or output (I/O) device by simply doing reads/writes to the memory locations (often called registers) in the I/O interfaces...
- The Arduino has many of these I/O interfaces all connected via the data bus

---

**Atmel ATmega328P**

- The Arduino Uno is based on an Atmel ATmega328P 8-bit microcontroller
  - 32kb of FLASH ROM
  - ______ bytes of RAM
  - ___ I/O lines
  - 3 timer/counters
  - Serial/SPI/I²C interfaces
  - A/D converter

**Where Does It All Go**

The program you write and compile on your laptop is downloaded into the microcontroller on the UNO board.

The code resides in the FLASH memory while the CPU fetches one instruction at a time and executes it. Data sits in the RAM (SRAM).

Your program controls external inputs and outputs primarily through PORTs B, C, and D which effectively control the values of the I/O pins.
**Digital I/O Example**

This program...
- Checks if the button is being pressed (i.e. the value on Port B bit 7 is '1').
- If so, it sets the value on Port B bit 5 to '1' (which is a high voltage) and connects to an LED to make it light up.
- Otherwise it sets PB5 to '0' (low voltage) and the LED does NOT light up.

Software code to control the microcontroller

```c
#include <avr/io.h>

int main()
{
    while(true){
        if(PortB[7] == 1)
            PortB[5] = 1;
        else
            PortB[5] = 0;
        return 0;
    }
}
```

Disclaimer: This code & connections are an approximation and should not just be copied.

Main Point: What happens to the hardware (button and LED) is controlled by the software.

**Arduino Digital I/O**

- ATmega328P has 23 pins on the chip that can be connected to other devices (switches, LEDs, motors, etc.)
  - Other members of the ATmega family may have more or less lines.
  - The Arduino Uno can make use of only 20 of these lines.
- Each pin can be used as a digital input or a digital output
  - **For output pins**: Your code determines what value ('1' or '0') appears
  - **For input pins**: Your code senses/reads what value another device is putting on the pin.

**Arduino Port/Pin Mapping**

- Since computers usually deal with groups of 8-bits (a.k.a. a byte), all of the 20 I/O pins are split into three I/O ports (B, C and D)
  - The avr-gcc software (SW) and the Arduino hardware use different names to refer to the bits within each port.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PortB[0]</td>
<td>DIG8</td>
<td>PortC[0]</td>
<td>AN0</td>
<td>PortD[0]</td>
<td>DIG0</td>
<td>PortD[0]</td>
<td>DIG0</td>
<td>PortD[0]</td>
<td>DIG0</td>
<td>PortD[0]</td>
<td>DIG0</td>
</tr>
</tbody>
</table>

Main Point: Each pin has a name the software uses (Portx) and a name used on the Arduino circuit board (Anx or DIGx).

**Arduino Digital I/O**

- The I/O ports (i.e. groups of pins) are the ____________ between your software program and the physical devices connected to the chip.
  - Your program is responsible for managing these ports (groups of I/O pins) in order to make things happen on the outside.
- Most I/O pins in a port can be directly controlled by your software for "__________" OR be used for other specific HW functionality integrated on chip
  - PORTC0 can be used as a digital I/O OR as the Analog-to-Digital Conversion input: ADC0
  - PORTD0 can be used as digital I/O OR the serial communication receive input: RXD
- We will discuss these other HW functions later...focus on digital I/O.
4.25 INPUT AND OUTPUT DEVICES

How to connect LEDs and Switches/Pushbuttons

What Do we Do Now

- Great! We have this Arduino microcontroller with all these pins…what should we connect to them?
  - Outputs: LED's, LCD screens, wired/wireless communication
  - Inputs: Buttons, Switches, temperature sensors, rotary encoders, etc.

- We'll start simple and try to attach an ________ (output) and a pushbutton/_______ (input)

(Light-Emitting) Diodes

- The simplest output we can control is an LED (Light-emitting diode) which is like a tiny light bulb
- An LED glows ('on') when the voltage across it is greater than ________ and is 'off' otherwise
- Voltage/Current Relationship:
  - For a resistor, current flowing through a resistor is proportional to the voltage across it (I = 1/R * V)
  - For an LED, current grows exponentially with voltage ______________
  - Since a small change in voltage could cause a large increase in current and possibly blow-out the LED, we need to limit current with a resistor
- LEDs are ________ meaning they only work in one orientation (longer leg must be at higher voltage)

LED Connection Approaches

- Below are some options for connecting an LED
- We need a series ___________ to limit current
  - Choose value based on amount of current you want
  - Amount of current will determine brightness of LED
  - i = V1/R1 = (Vs-VLED) / R1
  - Usually R1 is a few hundred ohms (330 or 470 ohms)

No current limitation...BAD

Choose resistor to limit current

Doesn't matter where resistor is placed as long as it is in series

An Arduino output will serve as our voltage source that can be either '0' (0V) or '1' (5V)
### LED Connection Approaches

- When letting a digital output control an LED, the value (i.e. '0' or '1') that causes the LED to light up depends on how the circuit is wired.
- Note: Gates can often "______" (take in) more current than they can "______" (push out), so option 2 may be preferred...but let's not worry about this now...let's use option 1.

<table>
<thead>
<tr>
<th>Option 1</th>
<th>Option 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED is on when gate outputs '1'</td>
<td>LED is on when gate outputs '0'</td>
</tr>
</tbody>
</table>

**Main Point:** LED's should always be connected in series with a current-limiting resistor.

### Switches and Pushbuttons

- Switches and pushbuttons can be in one of two configurations: __________ or __________
  - Switches can be opened or closed and then stay in that position until changed.
  - Pushbuttons are open by default and require you to push them to close the circuit (they then open when you release).

**Important Note 1:**
- When open a SW/PB looks like an ________ resistance (no current can flow).
- When closed a SW/PB looks like a __________ and no voltage drops across it.

**Important Note 2:**
- SW or PBs don’t produce digital 0’s or 1’s __________, they control what voltage (PWR/GND) is connected to your device.

### Power & Ground Connections

- Easy mistake when you’re just learning to wire up circuits:
  - Wire the inputs & outputs but ________ to connect power and ground.
- All circuits and chips require a connection to a power source and ground.
  - Gates
  - Switches
  - Buttons

**Actual connection…**

### Connecting a Switch

- Switches do not produce a 0 (GND) or 1 (VDD) by itself.
- Option 1: Attach one side to GND and the other side to the device.
  - When the switch=open, nothing is connected to the device (a.k.a. “___________”).
  - A floating input may sometimes appear as zero, and other times as a one.
  - We need the inputs to logic gates to be in either the 0 or 1 state...not floating.

**Option 2:**
- SW open => Input = __________
- SW closed => Direct wire from both VDD and GND to input = __________ Circuit = unknown voltage and possibly ________ current flow...BAD!!!
Using a Pull-up Resistor

- Solution: Put GND on the far side and a "pull-up" resistor at the input side
  - "Pull-up resistor" used to hold the input _______ unless something is forcing it to a zero
  - SW open => Arduino input looks like infinity. Resistance in series with Rp. Thus _______ through Rp and thus ____ voltage drop across Rp...input = VDD = 1
  - SW closed => Direct wire from GND to input...input = GND = 0...Also current flowing from Vdd to GND is __________ by Rp preventing a short circuit.
  - Usually Rp is large (10k ohms) to limit current

To calculate Vin:

\[ Vin = Vdd - V_{RP} \]

Vin = ____________

Input = ____________ since in series with inf. resistance of Arduino input

Thus, Vin = _____

Main Point: Buttons & switches should have GND connected to one side & a pull-up resistor on the other

Alternative Switch/Resistor Connections

- Consider the options to connect PWR & GND to a SW/PB
- Note: A gate input "looks like" an inf. resistance

Option 1

Option 2

Option 3

Option 4

Preferred

Less Preferred

(just take our word)

Overview

- In the next few slides you will learn
  - What your software needs to do to setup the pins for use as digital inputs and/or outputs
  - To set bits (to 1) and clear bits (to 0) using bitwise operations (AND, OR, NOT) to control individual I/O pins
  - How to do it in a readable syntax using shift operators (<<, >>)

Don't be worried if it doesn't make sense the first time...listen, try to make sense of it, and ask a lot of questions.
Controlling I/O Ports

- Each port (B, C, and D) has 3 registers in the μC associated with it that control the operation:
  - Each bit in the register controls something about the corresponding I/O bit.
  - ___________ Register (DDRB, DDRC, DDRD)
  - ___________ Register (PORTB, PORTC, PORTD)
  - ___________ Register (PINB, PINC, PIND)

- You'll write a program that sets these bits to 1's or 0's as necessary

**Register 1: Data Direction Register**

- DDRx (Data direction register) \(x = \{B, C, D\}...DDR_{B}, DDR_{C}, DDR_{D}\)
  - Controls whether pins on the chip act as inputs or outputs.
  - Example: If DDRB[5] = 0 -> PB5 (Port B bit 5 = DIG13 pin) will be used as ______
  - Example: If DDRB[5] = 1 -> PB5 (Port B bit 5) will be used as __________
  - All I/O lines start out as inputs when the μC is reset or powered up.

**Register 2: PORT Register**

- PORTx (Primarily used if port X is configured as an output)
  - When a pin is used as an output (DDRx[n] = 1), the corresponding bit in PORTx[n] determines the value/voltage of that pin.
  - E.g. By placing a '1' in port B bit 5, pin PB5 will output a ________ voltage

**Register 3: PIN Register**

- PINx[n] (Used if PORT is configured as an input)
  - When a bit is an input (DDx[n]=____), getting the bit from PINx[n] reflects the current value at the corresponding input pin.
  - The action of referencing PINx causes all the signals to be acquired.
  - if(PIND == 0x00) // check if all the signals coming into port D are 0's
    - char val = PINB; // read and save all 8 signals coming into port B in a variable 'val'.
  - Programs must read the ___________ bits in the PIN register, but can then use bitwise logical operations to check individual bits.
  - If a port is an input but has no signal connected to it, it will “float” and could be read as either zero or one.

**Main Point:** For pins configured as outputs, the values you put in the PORT register will be the output voltages.

**Main Point:** For pins configured as inputs, referencing the PINx register samples the input voltages at all the pins.
Review of Accessing Control Registers in C

- Control registers have names and act just like variables in a C program
- To put values into a control register you can assign to them like any C variable or perform bitwise operations
  - DDRD = 0xff; // 0b11111111 or 255
  - DDRB = 255;
  - PORTD |= 0xc0; // 0b11000000 or 192
  - PORTD |= 0b01110000;
- To read the value of a control register you can write expressions with them
  - unsigned char myvar = PIND; // grabs all 8-inputs on the port D
  - myvar = PINB & 0x0f; // you will see this grabs just the lower 4 inputs

Practice: Changing Register Bits

- Use your knowledge of the bitwise logic operations to change the values of individual bits in registers without affecting the other bits in the register.
  - Set DDRB, bit 3 to a '1'
    - DDRB |= 0b___________;
  - Clear the 2 upper bits in PORTC to '0's
    - PORTC &= 0x________________;
  - Flip bits 7 and 1 in DDRC
    - DDRC ^= 0b_______________;
  - Check if PIND, bit 4 = '1'
    - if (________________) { code }

Review

- To use a pin(s) as output:
  - If you want to use Port B, bit 5 as an output, make DDRB, bit 5 = 1
    - Ex. DDRB |= 0b00100000;
  - Then perform operations on PORTB register to place the desired output value into bit 5
    - Ex. PORTB |= 0b00100000; // make pin on B5 = Vdd (5V)
**Blinking an LED**

- Hardware and software to make an LED connected to D7 blink

```c
#include<avr/io.h>
#include<util/delay.h>
int main() {
  // Init. D7 to output
  DDRD |= 0x00;
  // Repeat forever
  while(1){
    // PD7 = 1 (LED on)
    PORTD |= 0x01;
    _delay_ms(500); // PD7 = 0 (LED off)
    PORTD &= ~0x01;
  }
  // Never reached
  return 0;
}
```

**Turning an LED on/off with PB**

- Hardware to turn an LED connected to D7 on/off when pressing a pushbutton connected to D4

```c
#include<avr/io.h>
int main() {
  // Init. D7 to output
  DDRD |= 0x80;
  // All pins start as input // on reset, so no need to // clear DDRD bit 4
  // Repeat forever
  while(1){
    // Is PD4 pressed?
    if( (PIND & 0x10) == 0){
      // PD7 = 1 (LED on)
      PORTD |= 0x80;
    } else {
      // PD7 = 0 (LED off)
      PORTD &= ~0x80;
    }
  }
  // Never reached
  return 0;
}
```

**Turning on an LED from a Button**

- Note: When the button is pressed a __ is produced at the PD4 input

```c
#include<avr/io.h>
int main() {
  // Init. D7 to output
  DDRD |= 0x00;
  // Repeat forever
  while(1){
    // PD7 = 1 (LED on)
    PORTD |= 0x01;
    _delay_ms(500); // PD7 = 0 (LED off)
    PORTD &= ~0x01;
  }
  // Never reached
  return 0;
}
```

**Pull Up Resistors**

- Adding and wiring pull-up resistors for input buttons can be time consuming...
- Thankfully, each Arduino input bit has an ___________ optional “pull-up resistor” associated with it.
  - If the pull-up is enabled, in the absence of an input signal, the input bit will be “pulled” up to a logical one.
  - The pull-up has no effect on the input if an active signal is attached.
Enabling Pull Up Resistors

- When DDRX[n] is '0' (i.e. a pin is used as input), the value in the ___________ register determines whether the internal pull-up is enabled
  - Remember, the PORT register is normally used when a pin is an output, but here its value helps enable the internal pull-up resistor

Using Internal Pull-up Resistors

- Let's simplify our wiring and use the internal pull-up resistors

Turning on an LED from a Button

- Note: When the button is pressed a '0' is produced at the PD4 input

```
#include<avr/io.h>

int main()
{
  // Init. D7 to output
  DDRD |= 0x80;
  // Enable pull-up on PD4
  PORTD |= 0x80;

  // Repeat forever
  while(1)
  {
    // Is PD4 pressed?
    if( (PIND & 0x10) == 0 )
    {
      // PD7 = 1 (LED on)
      PORTD |= 0x80;
    }
    else
    {
      // PD7 = 0 (LED off)
      PORTD &= ~(0x80);
    }
  }
  // Never reached
  return 0;
}
```
### Code Read-ability Tip #1

- Try to replace hex and binary constants with shifted constants

```c
#include<avr/io.h>
int main()
{
    // Init. D7 to output
    DDRD |= (1 << DD7);
    // Enable pull-up on PD4
    PORTD |= (1 << PD4);
    // Repeat forever
    while(1)
    {
        // Is PD4 pressed?
        if( (PIND & (1 << PD4)) == 0)
        {
            // PD7 = 1 (LED on)
            PORTD |= (1 << PD7);
        }
        else
        {
            // PD7 = 0 (LED off)
            PORTD &= ~(1 << PD7);
        }
    }
    // Never reached
    return 0;
}
```

```
#include<avr/io.h>
int main()
{
    // Init. D7 to output
    DDRD |= (1 << 7);
    // Enable pull-up on PD4
    PORTD |= (1 << 4);
    // Repeat forever
    while(1)
    {
        // Is PD4 pressed?
        if( (PIND & (1 << 4)) == 0)
        {
            // PD7 = 1 (LED on)
            PORTD |= (1 << 7);
        }
        else
        {
            // PD7 = 0 (LED off)
            PORTD &= ~(1 << 7);
        }
    }
    // Never reached
    return 0;
}
```

This syntax tells us we are putting a '1' in bit 7 (DD7) or bit 4 (PD4)...

We will teach you what all this means in the next slides...

### Shift Operations

- In C, operators '<<' and '>>' are the shift operators
  - `<<` = __________ shift
  - `>>` = __________ shift

- Format: data `<< bit_places_to_shift_by`
- Bits shifted out and dropped on one side
- Usually (but not always) 0's are shifted in on the other side

\[
x = x >> 2;
\]

(Right Shift by 2 bits)

<table>
<thead>
<tr>
<th>Original x</th>
<th>x Shifted by 2 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 1 1 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

\[
x = x << 2;
\]

(Left Shift by 2 bits)

<table>
<thead>
<tr>
<th>Original x</th>
<th>x Shifted by 2 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 1 0 1 0</td>
</tr>
</tbody>
</table>

0's shifted in...

0's shifted in...

### Another Example

- To get a 1 in a particular bit location it is easier to shift the constant 1 some number of places than try to think of the hex or binary constant

\[
0x1
\]

\[
0 0 0 0 0 0 0 1 = +1
\]

\[
1 << 3
\]

0's shifted in...

Suppose we want a 1 in bit location 3. Just take the value 1 and shift it 3 spots to the left

\[
0x1
\]

\[
0 0 0 0 0 0 0 1 = +1
\]

\[
1 << 5
\]

0's shifted in...

Suppose we want a 1 in bit location 5. Shift 1 5 spots to the left. Easier than coming up with 0x20...

### #define macros

- Can be used for simple find/replace scenarios
  - `#define find_pat replace_pat`
- Makes constants more readable and easier to change (if you have the same constant in 10 places and you realize you need to change it, just change the one `#define` statement)

```c
#define MAX_VALUE 100
int counter = MAX_VALUE;
```

Original Code

Compiler sees...

```c
int counter = 100;
```

Original Code

Compiler sees...

```c
#define DD7 7
#define PD4 4
...
DDRD |= (1 << DD7);
PORTD |= (1 << PD4);
```

Original Code

Compiler sees....
Register Bit Names

- Symbolic names for the positions of bits in each register are defined as constants in <avr/io.h>
  - `#include <avr/io.h>`
- Each PORTx register has its own constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Port Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTB</td>
<td>PB5 PB4 PB3 PB2 PB1 PB0</td>
</tr>
<tr>
<td>PORTC</td>
<td>PC5 PC4 PC3 PC2 PC1 PC0</td>
</tr>
<tr>
<td>PORTD</td>
<td>PD7 PD6 PD5 PD4 PD3 PD2 PD1 PD0</td>
</tr>
</tbody>
</table>

- All DDRx registers share the same constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Port Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDRB,C,D</td>
<td>DD7 DD6 DD5 DD4 DD3 DD2 DD1 DD0</td>
</tr>
</tbody>
</table>
- All PINx registers share the same constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Port Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINB,C,D</td>
<td>PIN7 PIN6 PIN5 PIN4 PIN3 PIN2 PIN1 PIN0</td>
</tr>
</tbody>
</table>

Putting it All Together

- Values for working with bits can be made using the ‘<<’ shift operator
  - OK:  `PORTB |= 0x20;`       Better:  `PORTB |= (1 << PB5);`
  - OK:  `DDRD |= 0x04;`       Better:  `DDRD |= (1 << 2);`
- This makes the code more readable and your intention easier to understand...
- More examples
  - `DDRC |= (1 << DD5)` set DDRC, bit 5
  - `PORTB ^= (1 << PB2)` invert PORTB, bit 2
  - `PORTD &= (1 << PD3)` clear PORTD, bit 3 <- WRONG!
    
    Why?

Clearing Bits...A Common Mistake

- When using the ‘&=’ operation to clear bits, remember to invert the bits.
- This won’t work to clear PD3 to ‘0’
  - `PORTD &= (1 << PD3);`
  - is the same as
  - `PORTD &= 0b00001000;`
  - which clears ___________ bit PD3
- Use the ‘~’ operator to complement the bits.
  - `PORTD &= ~(1 << PD3);`
  - is the same as
  - `PORTD &= 0b11110111;`
  - and now PD3 gets cleared.

Defining Your Own Symbolic Names

- You can make your own more meaningful symbolic names
  - `#define LEDBIT       (1 << PB6)`
  - `#define CLKBIT       (1 << PC3)`
  - `PORTB |= LEDBIT;`
  - `PORTC |= CLKBIT;`
  - `PORTC &= ~CLKBIT;`
- Can combine multiple bits into one defined value
  - `#define MULTIBITS   ((1 << PB3) | (1 << PB4) | (1 << PB5))`
    
    is the same as
    - `#define MULTIBITS   0b00111000` 00001000 00010000 1 << PB3 1 << PB4 1 << PB5 00111000
    - `PORTD &= MULTIBITS;`
COPYING BITS

Copying Multiple Bits

• Suppose we want to copy a portion of a variable or register into another BUT __________ affecting the other bits

• Example: Copy the lower 4 bits of X into the lower 4-bits of PORTB...but leave the upper 4-bits of PORTB UNAFFECTED

• Assignment __________ work since it will overwrite ALL bits of PORTB
  — PORTB = x; // changes all bits of PORTB

Copying Into a Register

• Solution...use these steps:

  • Step 1: Define a _____ that has 1’s where the bits are to be copied
    #define MASKBITS 0x0f

  • Step 2: _________ those bits in the destination register using the MASK
    _______ &= ~MASKBITS

  • Step 3: Mask the appropriate field of x and then _______ it with the destination, PORTB
    PORTB |= (______________);

  x = 01000011

Do We Need Step 2...Yes!!!

• Can’t we just do step 1 and 3 and OR the bits of x into PORTB
  #define MASKBITS 0x0f
  PORTB |= (x & MASKBITS);

• ____ , because what if the destination (PORTB) already had some _____ where wanted 0's to go...

• ...Just OR'ing wouldn't change the bits to _______

• That's why we need step 2
  • Step 2: Clear those bits in the destination register using the MASK
    PORTB &= ~MASKBITS;
Copying To Different Bit Locations

- What if the source bits are in a different location than the destination
  - Ex. Copy lower 4 bits of x to upper 4 bits of PORTB
- Step 1: Define a mask that has 1's where the bits are to be copied
  ```c
  #define MASKBITS 0xf0
  ```
- Step 2: Clear those bits in the destination register using the MASK
  ```c
  PORTB &= ~MASKBITS
  ```
- Step 3: ______ the bits of x to align them appropriately, then perform the regular step 3
  ```c
  PORTB |= ((______) & MASKBITS);
  ```

DEBOUNCING SWITCHES

Counting Presses

- Consider trying to build a system that counted button presses on PC2 (increment once per button press)
- We can write code to check if the button is pressed (==0) and then increment 'cnt'
- But remember, your code executes extremely ______...what will happen?

Waiting Through a Press

- Consider trying to build a system that counted button presses on PC2 (increment once per button press)
- We can write code to check if the button is pressed (==0) and then increment 'cnt'
- But remember, your code executes extremely fast...what will happen?
Interfacing Mechanical Switches/Buttons

- Mechanical switches and buttons do not make solid, steady contact immediately after being pressed/changed.
- For a short (few ms) time, “__________” will ensue and can cause spurious SW operation (one press of a button may look like multiple presses).
- Need to “debounce” switches with your software.
  - Usually waiting around _______ from the first detection of a press will get you past the bouncing and into the stable period.

Waiting Through a Press

- Consider trying to build a system that counted button presses on PC2 (increment once per button press).
- We can write code to check if the button is pressed (==0) and then increment `cnt`.
- But remember, your code executes extremely fast...what will happen?

What's Your Function

- Because there is a fair amount of work to do just to recognize a button press, you may want to extract those to functions you can call over and over again.

```c
#include<avr/io.h>
int main() {
    PORTC |= (1 << PC2); int cnt = 0; while(1) {
        char pressed = (PINC & 0x04); if( pressed == 0 ) {
            _delay_ms(5); while( (PINC & 0x04) == 0 ) {} _delay_ms(5); cnt++; }
        else return 0; }
    return 0;
}
```

```c
#include<avr/io.h>
char pc2Pressed() {
    char pressed = (PINC & 0x04); if( pressed == 0 ){
        _delay_ms(5); while( (PINC & 0x04) == 0 ) {} _delay_ms(5); return 1; }
    else return 0; }
int main() {
    PORTC |= (1 << PC2); int cnt = 0; while(1) {
        if( pc2Pressed() ) cnt++; }
    return 0;
}
```