ARDUINO AND THE AVR PROCESSOR

PROF. HARRY PORTER
PORTLAND STATE UNIV.
cs.pdx.edu/~harry

The AVR Microprocessor

Architecture & ISA

Registers

Instructions

The Arduino

The single-board computer

Integrated Development Environment (IDE)

How to program it.

Demo

Setup

A small program

Compiling + Downloading

Controlling Devices

Sensors + Actuators

The "AVR" Architecture

Company: ATMEL

Products:

- AVR Processors
  - "Micro" Controller
- ARM® Processors
  - For Linux/Phones/etc.
- Other Electronic Products

Different "Instruction Set Architectures" (ISAs)

AVR series

- AT tiny
- AT mega
  - ATmega328P — The chip on the Arduino "UNO"
  - ATmega2560 — The chip on the Arduino "Mega2560"
  - ⋮ other chips
- Others

®And other companies make ARM chips.

ARDUINO

ARDUINO BOARDS

"UNO" ← $24.00

"Mega 2560" ← $40.00

⋮

"Due" ← Uses an ARM Processor. Bigger + Faster

ARDUINO SOFTWARE

INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

PROGRAMMING IN "C"

- EDITOR
- COMPILER
- DOWNLOADING FRAMEWORK

All combined

They say "UPLOADING"...

RUNS ON YOUR MAC / LINUX / PC

"UNO" BOARD

ATmega328P Chip DIP-28

PROCESSOR:

AVR instruction Set.

SoC: All on one chip "System on a Chip"

- SRAM: 2 Kbytes
- FLASH: 32 Kbytes
- SPEED: 16 MHz (≈16 MIPS)
- UART: 1 (Serial Interface)
- EEPROM: 1024 bytes.

POWER:
7-12 Volts
- 9V Battery
- USB cable.

INPUT/OUTPUT PINS: 14

DIGITAL I/O (0 = LOW, 1 = HIGH)

ALSO: 6 PINS CAN BE USED FOR PWM

Pulse-Width Modulation: for controlling SERVO motors

ANALOG INPUT PINS: 6

USB: TO DOWNLOAD PROGRAM TO BOARD.

INSTRUCTIONS:

- All code must be in FLASH.
- 16-bit WORDS.
- INSTRUCTIONS ARE 1 OR 2 WORDS. (16 OR 32 bits)

REGISTERS

- 32 Regs
- 8-bits in size.

The first 16 have direct addressing: "r0, r1, ... r15"

REGISTERS ARE MAPPED INTO LOW MEMORY ADDRESSES.

MEM. ADDRESSES	
$00_x - 1F_x$	REGISTERS (32)
$20_x - 5F_x$	I/O REGISTERS (64)
$60_x - FF_x$	MEMORY MAPPED REGS (160) ↑ EXTENDED I/O REGS
100_x – and up	SRAM (1.75K = 1,792)

REGISTERS r0 - r15, r16 - r31

These use a different address mode.

I/O PORT "REGS" 0 - 31, 32 - 63

INSTRUCTIONS

TWO STAGE PIPELINE (Simultaneous)

- FETCH next instruction
- EXECUTE current instruction

CYCLES PER INSTRUCTION:

MOST INSTRUCTIONS TAKE 1 CYCLE.

1 MHz → 1 MIPS (Million Instructions per second)

16 MHz → 16 MIPS

OTHERS TAKE 2, 3, 4, OR 5 CYCLES.

EXAMPLES: 16-BIT ADD; MULTIPLY.

LOAD, STORE, BRANCHES: 2 CYCLES.

ISA: INSTRUCTION SET ARCHITECTURE

AVR (Alf and Vegard's RISC Processor)

NORWEGIAN INSTITUTE OF TECHNOLOGY (NTH)

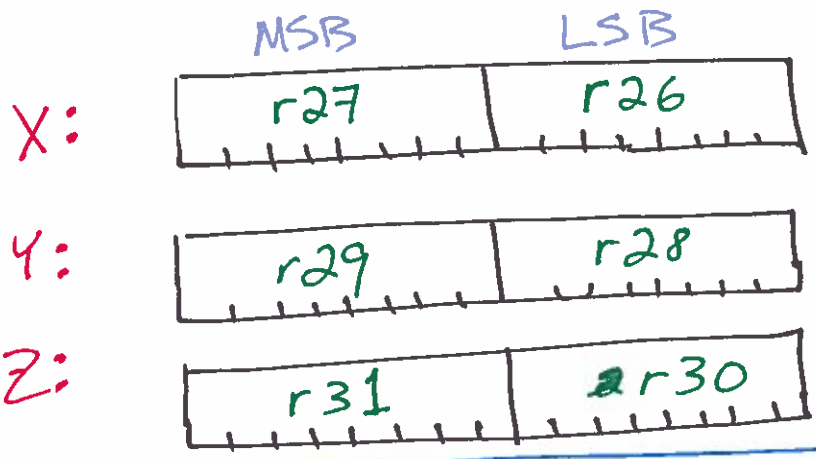
NORDIC SEMICONDUCTOR

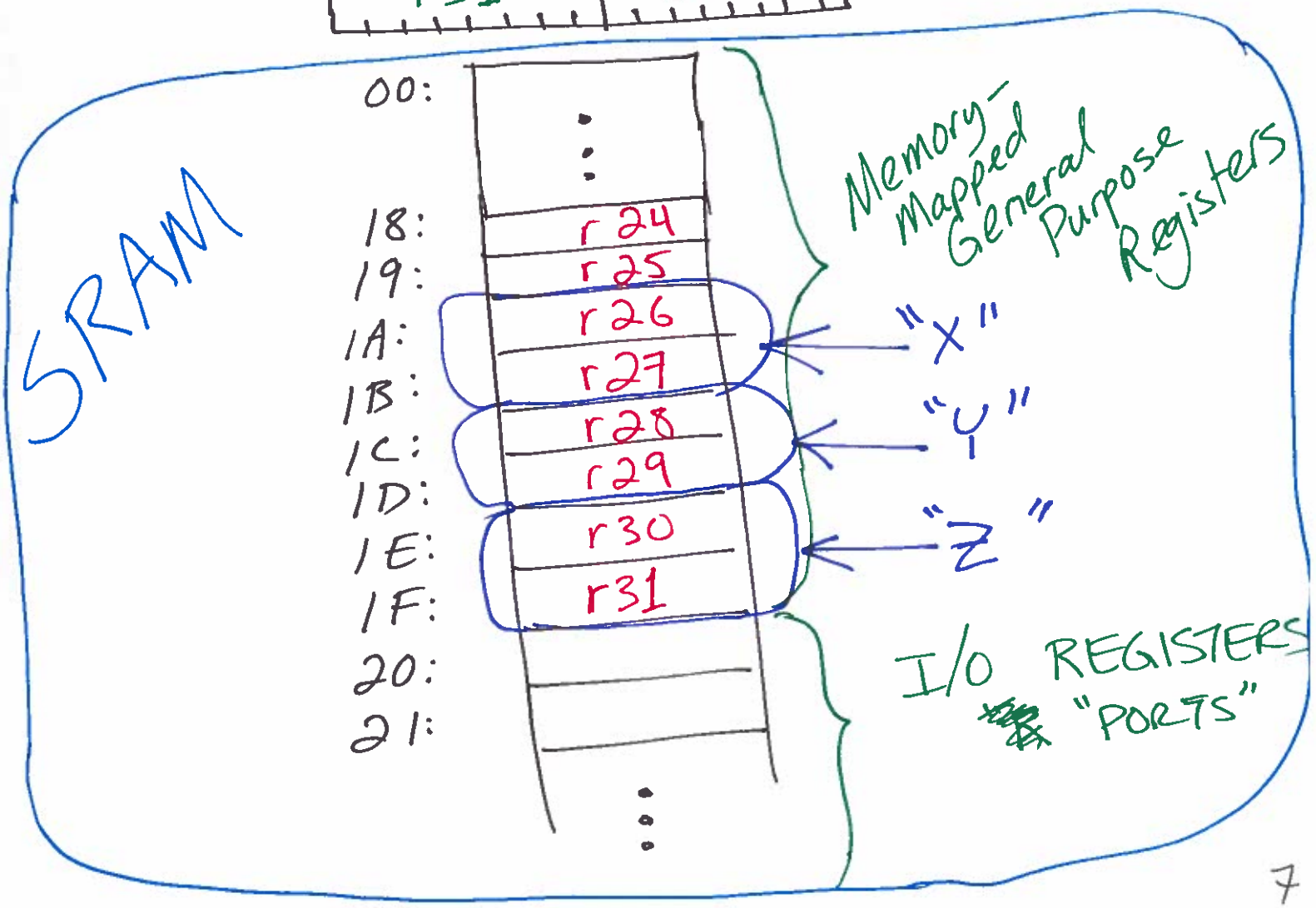
DESIGNED TO EXECUTE "C" CODE.

16-bit Registers

Names: "X", "Y", "Z"

REGISTERS r0 – r31 ARE 8-bits.

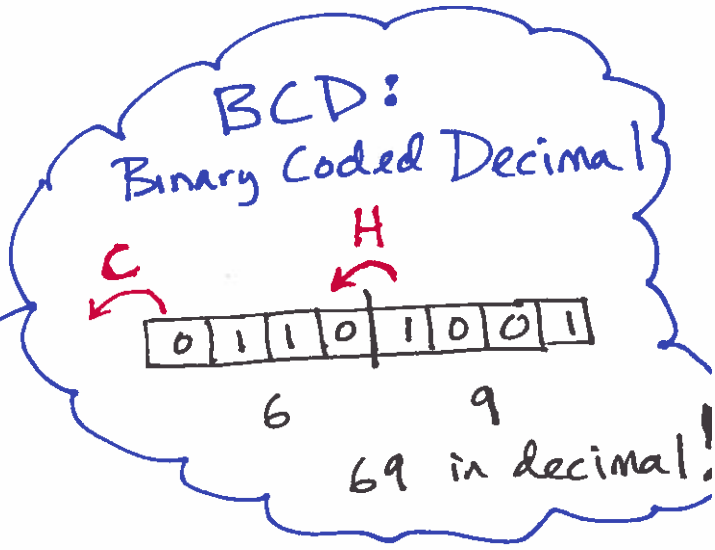




SREG: Status Register

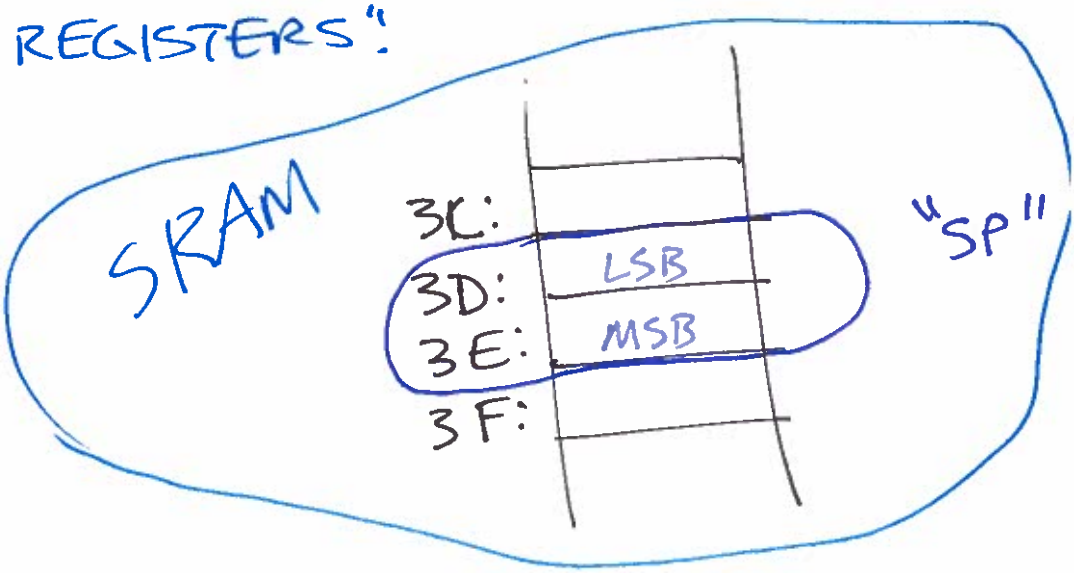
Eight Bits

C = Carry
Z = Zero
N = Negative/Sign
V = Overflow
S = N ⊕ V
H = Half Carry
T = For BIT Load/Store
I = Interrupts Enabled



THE STACK

- REGISTER: "SP" (16-bits)
- THE STACK IS IN SRAM.
- THE STACK GROWS DOWNWARD.
- "SP" IS DEFINED AS ONE OF THE "I/O REGISTERS".



- INITIALIZED ON POWER-UP TO END OF SRAM.
- USED IN THESE INSTRUCTIONS:

  PUSH
  POP
  CALL
  RET

FLASH MEMORY

CONTAINS INSTRUCTIONS.

ORGANIZATION:

16 K × 2 bytes

INSTRUCTION LENGTH: 2 bytes
(SOME TAKE 4 bytes)

PROGRAM COUNTER: 14 bits

$2^{14} = 16K$

FLASH MEMORY

APPLICATION PROGRAM

BOOT LOADER

ADD INSTRUCTIONS

ADD

add r7, r4

r7 ← r7 + r4

sets Carry bit

8 bit addition

1 cycle.

ADC

adc r8, r5

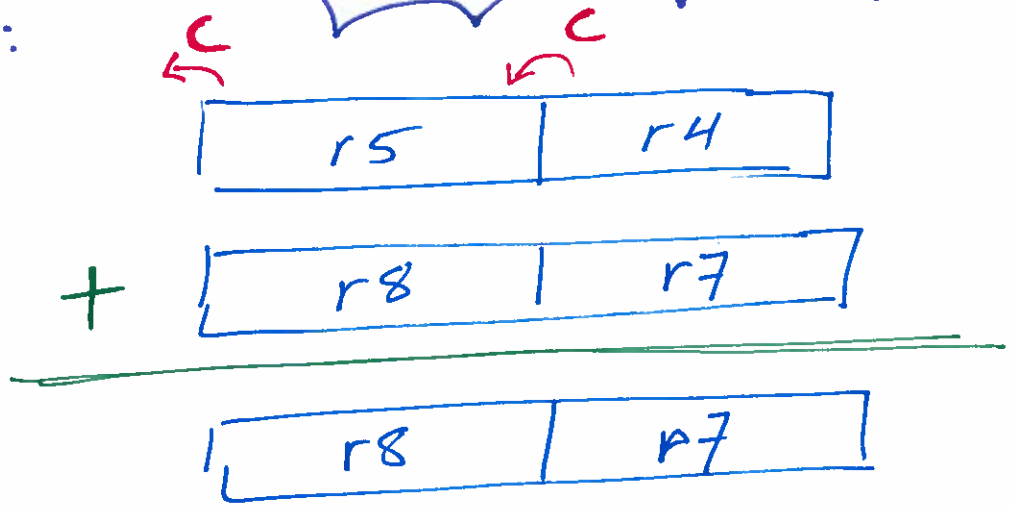
r8 ← r8 + r5 + Carry

Uses Carry bit

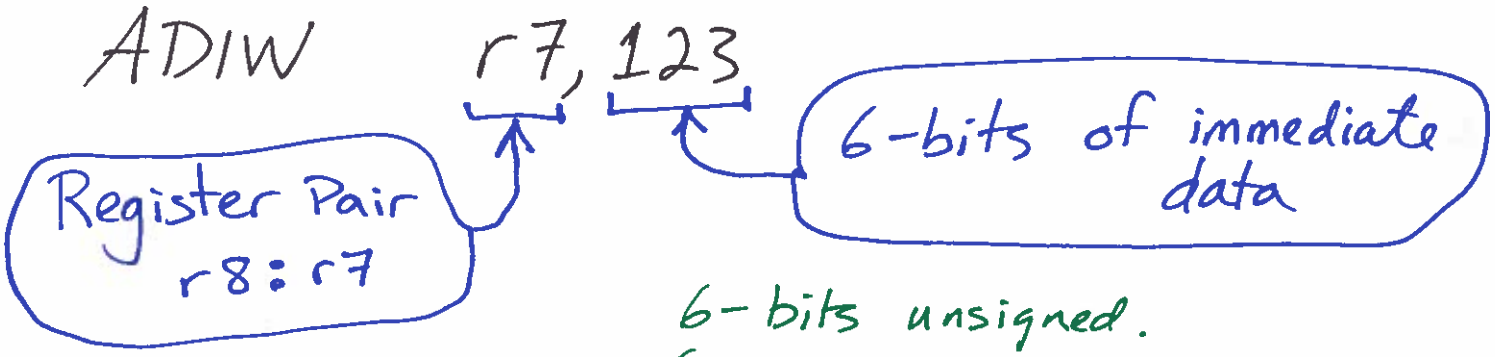
Also sets carry out.

1 cycle.

16-Bit Addition: 2 cycles



Immediate Data



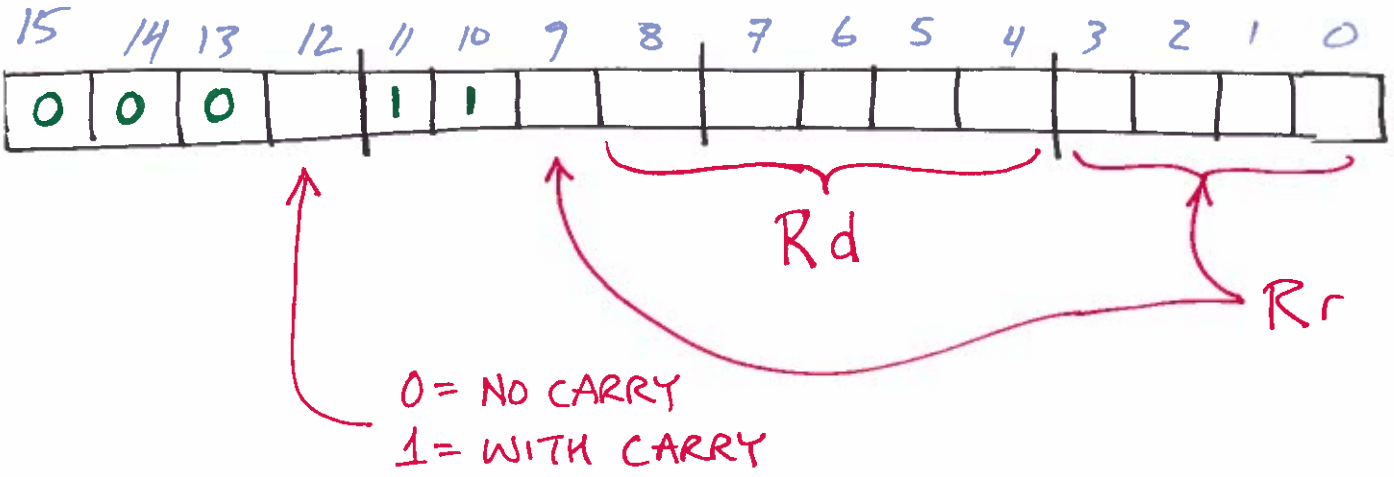
6-bits unsigned.
(8-bit values in some cases)

16-Bit Addition

Arithmetic and Logic Instructions

ADD	ADD ADC ADIW
SUBTRACT	SUB SUBI SBC SBCI SBIW
INCREMENT	INC
DECREMENT	DEC
AND	AND ANDI
OR	OR ORI
EXCLUSIVE-OR	EOR
COMPLEMENT	COM
NEGATE	NEG
COMPARE	CP CPC CPI
SWAP	SWAP (nibbles)
SHIFT	LSR ROR ASR ROL

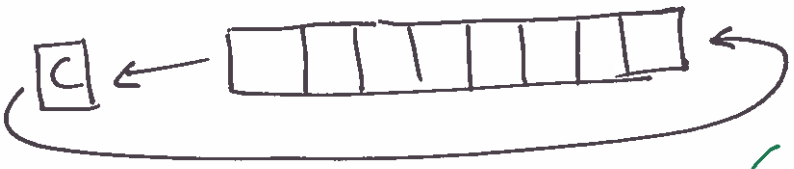
Instruction Encoding



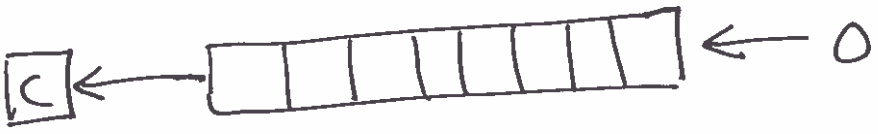
add Rd, Rr (No carry)

adc Rd, Rr (with carry)

ROL Rotate Left Through Carry (1 bit)



LSL Left Shift logical (1 bit)



Same encoding, except Rd = Rr

rol Rd ← like "adc" w/ Rd = Rr

lsl Rd ← like "add" w/ Rd = Rr

MULTIPLY

8-bit × 8-bit → 16-bit

2 cycles

MUL	unsigned
MULS	signed
MULSU	signed × unsigned.

The result always goes into R1:R0.

FRACTIONAL MULTIPLY

FMUL, FMULS, FMULSU

Also performs shifting

16-bit × 16-bit → 32-bit

Requires 16 instructions

DIVIDE

Sorry ☹

Write a subroutine!

MOV

```
mov   r8, r3
movw  r8, r3
```

r8 ← r3

r9:r8 ← r4:r3

LOAD FROM MEMORY

```
ld    r3, X
ld    r3, X+
ld    r3, -X
```

r3 ← M[X]

r3 ← M[X++]

r3 ← M[--X]

Must be "X", "Y", or "Z"

Any reg

```
lds   r3, address
```

16-bit address; instr is 32-bits

"Data Space" = SRAM

```
lpm   r3, Z
lpm   r3, Z+
```

Register is always "Z"; Optional Post-Increment

"Program Memory" = FLASH

ALSO FOR "STORE"

BRANCHING

CONDITIONAL BRANCHES

BRxx — Test StatusReg and Branch.

JUMP UNCONDITIONALLY

JMP
- Absolute Address
- PC-Relative (with 12-bit offset)

IJMP — INDIRECT THROUGH THE "Z" REGISTER

SKIP INSTRUCTION

SBxx — Test Status Reg flags and skip the next instruction if true.

PUSH/POP

PUSH
POP

Push/Pop 8-bit reg. to/from the stack

CALL/RETURN

CALL ← Absolute Addr or PC-Relative with 12 bit offset.

RET

NOP

NOP

BREAK

BREAK Uses ON-CHIP debug system.
Not used in application s/w.
Set CPU to "STOPPED" Mode

SLEEP

SLEEP Put the chip in "SLEEP" Mode.

WATCHDOG RESET

WDR Reset the watchdog timer.

STORE INTO PROGRAM MEMORY

Write all ones. → Can erase Pages in FLASH memory.
Can write bits to pages in FLASH memory.
Set some bits to zero. → Can set bootloader LOCK bits.

INTERRUPT MODEL

"GLOBAL INTERRUPT ENABLED BIT"

1 BIT IN THE STATUS REGISTER

"I" = ENABLED/DISABLED.

MANY SOURCES OF INTERRUPT

I/O DEVICES: *

- SERIAL COM (UART): I/O COMPLETED.
- EEPROM READY
- STORE FLASH MEMORY READY.
- TIMER
- ANALOG COMPARATOR.
- RESET PIN (Pin 9 on the chip)
- EXTERNAL INTERRUPTS (Pins 16, 17, 3)

EACH INTERRUPT SOURCE HAS ITS OWN "INTERRUPT ENABLED/DISABLED" BIT.

* More complex: Many devices have several types of interrupts.

EXAMPLE: "timer" (9 interrupts)
"serial" (3 interrupts)

RESET

POWER-ON RESET: SUPPLY VOLTAGE

EXTERNAL RESET: EXTERNAL PIN (BUTTON)

WATCH DOG TIMER:

IF THE "Watchdog" system is enabled...
Software must periodically "feed the dog". (eg every 1 second)
(Issue appropriate instruction)
Failure (e.g. Software bugs/looping) will cause a system reset.

BROWN-OUT DETECTOR:

If enabled...
A reset will occur if voltage falls below a threshhold.

Interrupt Vector

Located in Program (FLASH) memory.

Starts at address 0000x

26 Entries.

2 Words (= 16 x 2 = 32 bits) per entry.

Address	
0000	RESET
0002	EXTERNAL INTERRUPT (PIN 16)
⋮	
000C	WATCHDOG TIME-OUT
⋮	
001C	TIMER
⋮	
0024	UART/SERIAL RECV. COMPLETE
⋮	

Each Entry contains a "JMP"

JMP addr

2 word instruction

Interrupt Handling

When the interrupt occurs...
(Is it enabled? – wait if necessary.)

- Global Interrupt Enabled Bit ("I") in status reg is CLEARED (=Disabled)
- Store PC on the STACK.
- Load PC from the INTERRUPT VECTOR.
- Begin executing the Interrupt Handler. ("Interrupt Service Routine" = ISR)
- The Interrupt handler ends by executing

  RETI Return from interrupt.
  - Pop PC from STACK
  - SET THE "I" BIT (back to "enabled")

ARDUINO IDE ← FREE!

- INCLUDES A "C" COMPILER (+ ASSEMBLER + LINKER)
- YOU INVOKE FUNCTIONS TO PERFORM I/O.
- THEY PROVIDE A LIBRARY OF FUNCTIONS.
- EASY TO USE (IMHO)
- ALSO INCLUDES SOME DEMO PROGRAMS.
- THE "EXECUTABLE" IS IN "INTEL HEX" FORMAT.
- COMMUNICATES w/ THE BOARD USING USB (OR SERIAL COM)
- BOOTLOADER ON BOARD RECIEVES THE EXECUTABLE AND WRITES IT TO FLASH.
- "RESET" BUTTON — Interrupt causes a Jump to the PROGRAM in FLASH.

What is Missing?

- No Page Tables
  → No Virtual Memory
- No "System"/"User" modes
  → No protected Kernel

} No UNIX

- Small Addresses
  → Never much memory
- Small Registers
- No cache
- Minimal Pipelining
- Slow clock
- No Floating Point
  → No "number crunching"
  → No graphics

} Poor Performance

Strengths

- Simple ISA
  - Not too complicated!
- Low Cost
- Low Power
- Small
  - Entire system is small
- Good Software Support
  - Easy-to-use IDE
- Focus on
  - Digital I/O
  - Analog I/O
  - Pulse Width Modulation

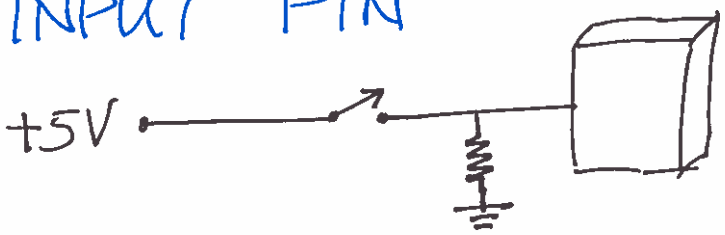
Modern Micro Controller
Good for Embedded Applications

- Robots
- Appliances

DEMO

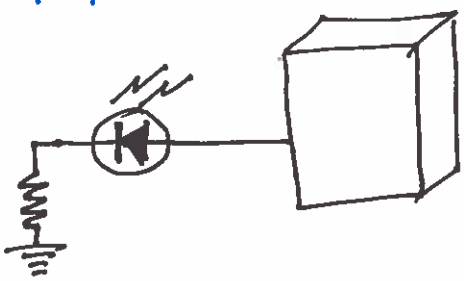
BUTTON

- DIGITAL INPUT PIN



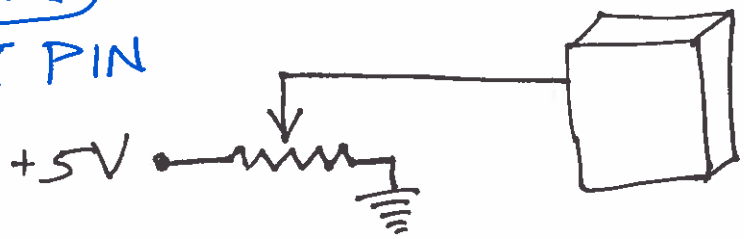
LED

- DIGITAL OUTPUT PIN



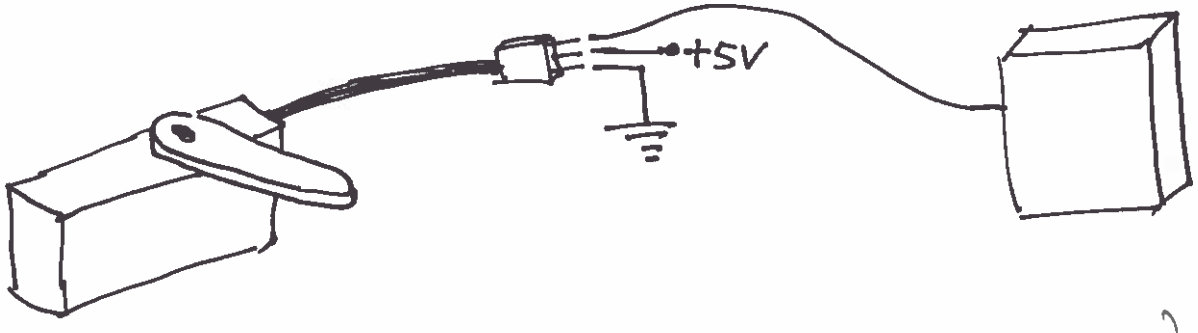
POTENTIOMETER

- ANALOG INPUT PIN



SERVO MOTOR

- PULSE WIDTH MODULATED OUTPUT



Pulse Width Modulation (PWM)

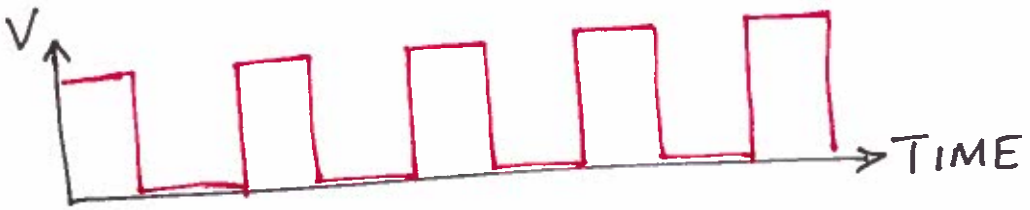
Goal: send a value over a wire.
(eg ranging from 0% to 100%)

Option 1: Analog; vary the voltage.

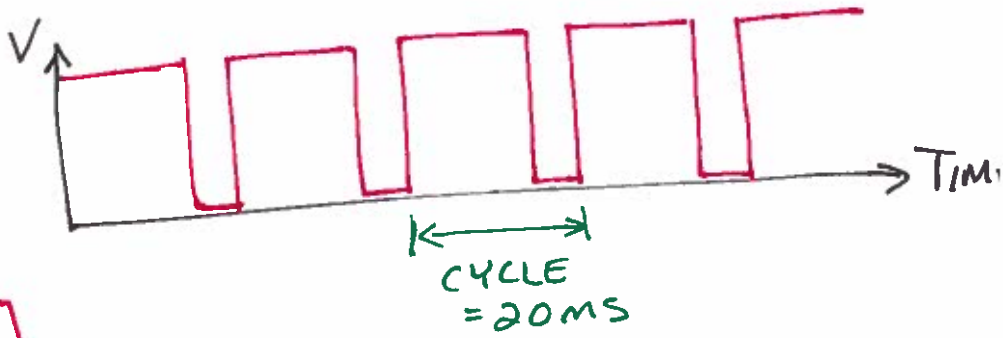
Option 2: PWM!

Value = 0%

Value = 50%



Value = 100%



Details

Cycle time = 20ms
(i.e., 50 Hz)

Pulse width:
1 ms = 0%
⋮
2 ms = 100%

PROGRAM IN "C"

GOAL:

Read from I/O pin ("BUTTON").

Write to I/O pin ("LED")

PRESSED ⇒ LED ON

CODE

```
void setup () {
    pinMode (3, INPUT);
    pinMode (5, OUTPUT);
}
void loop () {
    int i;
    i = digitalRead (3);
    digitalWrite (5, i);
}
```

GOAL

READ POTENTIOMETER (0 - 1023) ← PIN A0

UPDATE SERVO POSITION (0° - 180°) ← PIN 9

REPEAT FOREVER

CODE

```
#include <Servo.h>
Servo myServo;
int i;
void setup () {
    myServo.attach(9);
}
void loop() {
    i = analogRead (0);
    i = map(i, 0, 1023, 0, 180);
    myServo.write (i);
}
```

BOOK

"GETTING STARTED WITH ARDUINO"

MASSIMO BANZI

MAKE MAGAZINE / O'REILLY

STARTER KIT

"SPARKFUN INVENTOR'S KIT"

$90 - $100