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

AVR Series
- AT tiny
- AT mega
  - ATmega328P
  - ATmega2560
  - Other chips
- Others

Different "Instruction Set Architectures" (ISAs)

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

They say "uploading..."

Runs on your MAC/Linux/PC
"UNO" BOARD

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

**Input/Output Pins:** 14
- Digital I/O (0 = LOW, 1 = HIGH)
- Also: 6 pins can be used for PWM

**Analog Input Pins:** 6

**Power:**
- 7-12 Volts
- 9V Battery
- USB Cable

**Pulse-Width Modulation** for controlling servo motors

**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.

Registers are mapped into low memory addresses.

Mem. Addresses

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00_x - 1F_x</td>
<td>Registers (32)</td>
</tr>
<tr>
<td>20_x - 5F_x</td>
<td>I/O Registers (64)</td>
</tr>
<tr>
<td>60_x - FF_x</td>
<td>Memory Mapped Regs (160)</td>
</tr>
<tr>
<td>100_x - and up</td>
<td>Extended I/O Regs</td>
</tr>
<tr>
<td></td>
<td>SRAM (1.75K) = 1792</td>
</tr>
</tbody>
</table>

REGISTERS $r_0$ - $r_{15}$, $r_{16}$ - $r_{31}$

These use a different address mode.

I/O Port "Regs" $0$ - $31$, $32$ - $63$
INSTRUCTIONS

Two stage pipeline
- FETCH next instruction
- EXECUTE current instruction

Simultaneous

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.

\[
\begin{array}{c|c|c}
    & \text{MSB} & \text{LSB} \\
    X: & r27 & r26 \\
    Y: & r29 & r28 \\
    Z: & r31 & r30 \\
\end{array}
\]
SREG: Status Register

Eight Bits

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

BCD: Binary Coded Decimal

01101001

69 in decimal!
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 x 2 bytes

INSTRUCTION LENGTH: 2 bytes
(SOME TAKE 4 bytes)

PROGRAM COUNTER: 14 bits

$2^{14} = 16K$

FLASH MEMORY

3FFF:

APPLICATION PROGRAM

BOOT LOADER

0002: 0001: 0000:
**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

ADDW

Register Pair r8: r7

r7, 123

6-bits of immediate data

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

16-Bit Addition

Arithmetic and Logic Instructions

ADD
SUBTRACT
INCREMENT
DECREMENT
AND
OR
EXCLUSIVE-OR
COMPLEMENT
NEGATE
COMPARE
SWAP
SHIFT

ADD
ADC
ADIW

SUB
SUBI
SBC
SBCI
SBIW

INC
DEC

AND
ANDI

OR
ORI

EOR

COM
NEG

CP
CPL
CPI

SWAP (nibbles)

LSR
ROR
ASR
ROL
**INSTRUCTION ENCODING**

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
0 0 0 1 1             Rd
                       Rr
0 = NO CARRY
1 = WITH CARRY
```

- **add** \( R_d, R_r \) (No carry)
- **adc** \( R_d, R_r \) (With carry)

**ROL** Rotate Left Through Carry (1 bit)

**LSL** Left Shift Logical (1 bit)

Same encoding, except \( R_d = R_r \)

- **rol** \( R_d \) ← like "adc" w/ \( R_d = R_r \)
- **lsl** \( R_d \) ← like "add" w/ \( R_d = R_r \)
MULTIPLY

8-bit x 8-bit → 16-bit

2 cycles

MUL
MULS
MULSU

Unsigned
Signed
Signed x unsigned

The result always goes into R1:R0.

FRACTIONAL MULTIPLY

FMUL, FMULS, FMULSU

Also performs shifting

16-bit x 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]
Must be "X", "Y", or "Z"

Any reg

ld5 r3, address
16-bit address; instr is 32-bits

"Data Space" = SRAM

lm r3, Z
lm r3, Z+

"Program Memory" = FLASH

Register is always Z;
Optional Post-Increment

also for "store"
BRANCHING

Conditional Branches

\( BR_{xx} \) Test Status Reg and Branch.

Jump Unconditionally

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

\( IJMP \) INDIRECT THROUGH THE "Z" REGISTER

Skip Instruction

\( SB_{xx} \) Test Status Reg flags and skip the next instruction if true.

Push/Pop

Push
Pop

Call/Return

Call
Ret

Push/Pop 8-bit reg.

→/from the stack

Absolute Addr or PC-Relative with 12-bit offset.
NOP

BREAK

SLEEP

WATCHDOG RESET

STORE INTO PROGRAM MEMORY

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

Put the chip in "SLEEP" mode.

Reset the Watchdog timer.

Can erase Pages in FLASH memory.
Can write bits to pages in FLASH memory.
Can set bootloader LOCK bits.
"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." (e.g. 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 threshold.
interrupt vector

located in program (flash) memory.
starts at address 0000x
26 entries.
2 words (= 16 * 2 = 32 bits) per entry.

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>reset external interrupt (pin 16)</td>
</tr>
<tr>
<td>0002</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>00c</td>
<td>watchdog time-out</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>001c</td>
<td>timer</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>uart/serial recv. complete</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

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")
• **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 receives 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
- Small Addresses
  → Never Much Memory
- Small Registers
- No Cache
- Minimal Pipelining
- Slow Clock
- No Floating Point
  → No "Number Crunching"
  → No Graphics

No
UNIX

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 Microcontroller
Good for Embedded Applications
- Robots
- Appliances
DEMO

BUTTON
- DIGITAL INPUT PIN

+5V

LED
- DIGITAL OUTPUT PIN

POTENTIOMETER
- ANALOG INPUT PIN

+5V

SERVO MOTOR
- Pulse Width Modulated OUTPUT
**Pulse Width Modulation (PWM)**

**Goal:** Send a value over a wire.

(e.g., ranging from 0% to 100%)

**Option 1:** Analog; vary the voltage.

**Option 2:** PWM!

---

**Value = 0%**

---

**Value = 50%**

---

**Value = 100%**

---

**Details**

- **Cycle Time = 20 ms** (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 $\rightarrow$ LED ON

---

**CODE**

```c
void setup () {
  pinMode (3, INPUT);
  pinMode (5, OUTPUT);
}

void loop () {
  int i;
  i = digitalRead (3);
  digitalWrite (5, i);
}
```
**Goal**

**READ POTENTIOMETER**

\((0 - 1023) \leftrightarrow \text{PIN A0}\)

**UPDATE SERVO POSITION**

\((0° - 180°) \leftrightarrow \text{PIN 9}\)

**REPEAT FOREVER**

**Code**

```c
#include <Servo.h>
Servo myServo;
int i;

void setup () {
  myServo. attach (9);
}

void loop () {
  i = analogRead (A0);
  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