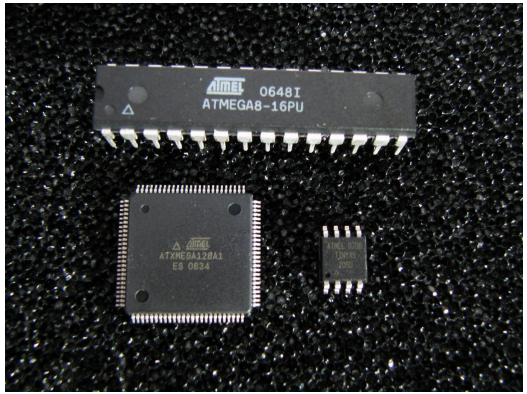
AVR

- **A**lf and **V**egard's **R**isc processor
- Popular 8-bit Microcontroller Architecture
- Easy to learn (e.g. Arduino)



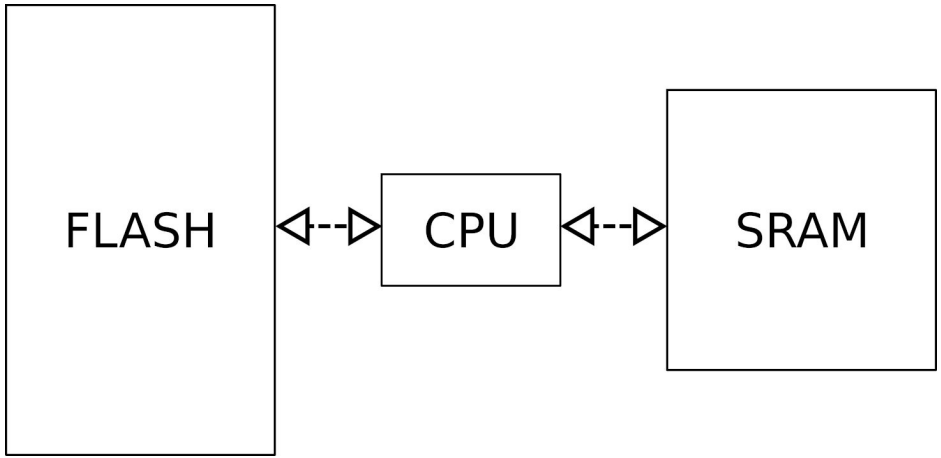
By Springob [CC BY 3.0], from Wikimedia Commons

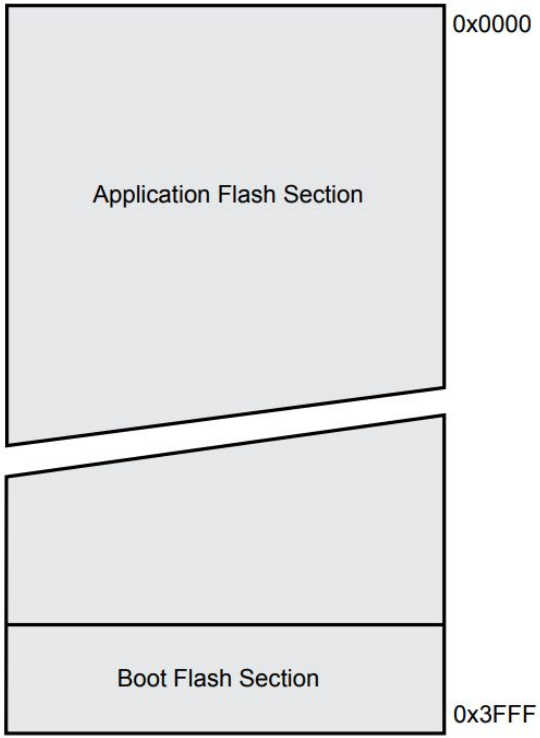
slaps roof of AVR architecture
this bad boy can fit so many
device specific differences in it

Architecture

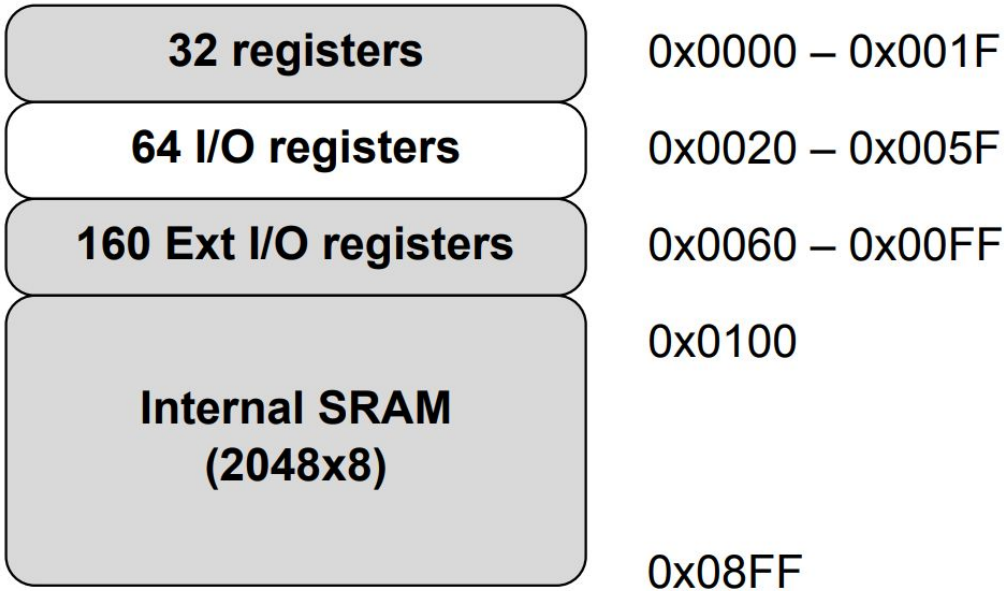
Architecture

- Harvard Architecture
- Usually flash memory, SRAM and EEPROM
- Different memory layouts per device





Program Memory


32 registers
64 I/O registers
160 Ext I/O registers
Internal SRAM
(2048x8)
0x0000 – 0x001F
0x0020 – 0x005F
0x0060 – 0x00FF
0x0100
0x08FF

Data Memory

Vector No	Program Address(2)	Source	Interrupts definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B

Interrupts

Instruction Set

- Word size: 2 byte
- Registers: r0 - r31
- Pointer Registers:
  - **X** - r27:r26
  - **Y** - r29:r28
  - **Z** - r31:r30

Instruction Set

```
10e0        ldi r17, 0x00
caed        ldi r28, 0xda
d0e0        ldi r29, 0x00
04c0        rjmp 0x1f8
2197        sbiw r28, 0x01
fe01        movw r30, r28
0e94f816    call 0x2df0
```

Instruction Set

```
10e0          ldi r17, 0x00

caed          ldi r28, 0xda

d0e0          ldi r29, 0x00

04c0          rjmp 0x1f8

2197          sbiw r28, 0x01

fe01          movw r30, r28

0e94f816      call 0x2df0     ; Wide, two-word call
```

Instruction Set

```
10e0          ldi r17, 0x00   ; Load immediate into register
caed          ldi r28, 0xda
d0e0          ldi r29, 0x00
04c0          rjmp 0x1f8
2197          sbiw r28, 0x01
fe01          movw r30, r28
0e94f816      call 0x2df0     ; Wide, two-word call
```

Instruction Set

```
10e0          ldi r17, 0x00   ; Load immediate into register
caed          ldi r28, 0xda
d0e0          ldi r29, 0x00
04c0          rjmp 0x1f8
2197          sbiw r28, 0x01  ; Subtract immediate from word
fe01          movw r30, r28   ; Move word
0e94f816      call 0x2df0     ; Wide, two-word call
```

Instruction Set

```
10e0          ldi r17, 0x00   ; Load immediate into register
caed          ldi r28, 0xda
d0e0          ldi r29, 0x00
04c0          rjmp 0x1f8      ; Relative jump to 0x1f8
2197          sbiw r28, 0x01  ; Subtract immediate from word
fe01          movw r30, r28   ; Move word
0e94f816      call 0x2df0     ; Wide, two-word call
```

Peripherals

- Everything too complex for instruction set
- Communication with the outside world
- All different kinds of peripherals (EEPROM, UART, I2C, ...)
- Controlled via I/O registers

Example: UART

- Protocol used for serial communication
- In AVRS
  - Only care about UART on AVR side
  - Just dump received/sent values
- For the proof of concept: just ATmega328

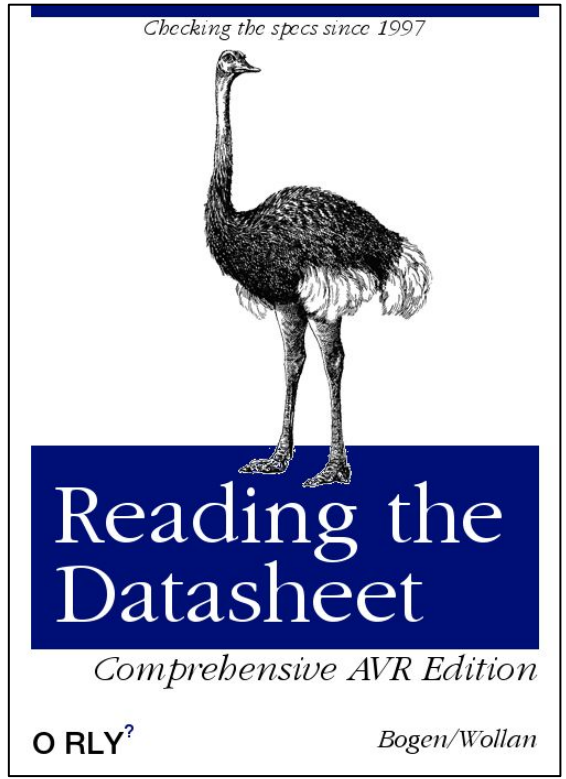
UART

UART-specific I/O registers on ATmega328:

Register Address	Register Name
0xc0	**UCSR0A**
0xc1	UCSR0B
0xc2	UCSR0C
0xc3	-
0xc4	UBRR0L
0xc5	UBRR0H
0xc6	**UDR0**

Understanding UART

```
read  18 from c1
write 98   to c1
read  98 from c1
write 98   to c1
read  60 from c0
write 45   to c6
read  60 from c0
```



UART Control Register A

RXC0	**TXC0**	**UDRE0**	FE0	DOR0	UPE0	U2X0	MPCM0

- **RXC0**: Receive Complete
- **TXC0**: Transmit Complete
- **UDRE0**: USART Data Register Empty

UART Control Register A

RXC0	TXC0	**UDRE0**	FE0	DOR0	UPE0	U2X0	MPCM0

1. Wait till UDRE0 is set (data register is empty)

UART Control Register A

RXC0	TXC0	**UDRE0**	FE0	DOR0	UPE0	U2X0	MPCM0

1. Wait till UDRE0 is set (data register is empty)
2. Take data from UDR (and handle it however you like)

UART Control Register A

RXC0	**TXC0**	**UDRE0**	FE0	DOR0	UPE0	U2X0	MPCM0

1. Wait till UDRE0 is set (data register is empty)
2. Take data from UDR (and handle it however you like)
3. Data was “sent”, set UDRE and TXC

UART Control Register A

RXC0	**TXC0**	**UDRE0**	FE0	DOR0	UPE0	U2X0	MPCM0

1. Wait till UDRE0 is set (data register is empty)
2. Take data from UDR (and handle it however you like)
3. Data was “sent”, set UDRE and TXC

UART Control Register A

RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0

- Same as TX, other way around...
- ... but not necessarily smart (interrupts!)
- Reading from UDR takes element from RX queue

Peripherals in SIMAVR

- Modular system for adding new components
- Register handler functions/structs for...
  - Interrupts
  - I/O read/write
  - Internal IRQs
- Internal IRQs for additional logic

Peripherals in SIMAVR

- Interrupts

```
// Define RXC interrupt via macro
.rxc = { \
    .enable = AVR_IO_REGBIT(UCSR ## _name ## B, RXCIE ## _name), \
    .raised = AVR_IO_REGBIT(UCSR ## _name ## A, RXC ## _name), \
    .vector = USART ## _name ## _RX_vect, \
    .raise_sticky = 1, \
}

// Register IV
avr_register_vector(avr, &p->rxc);
```

Peripherals in SIMAVR

- I/O read/write

```
// Define handler function for UDR write (TX)
static void avr_uart_udr_write(
    struct avr_t * avr,
    avr_io_addr_t addr,
    uint8_t v,
    void * param)
{
...
}

// Register handler
avr_register_io_write(avr, p->r_udr, avr_uart_udr_write, p);
```

Peripherals in SIMAVR

- Internal IRQs

```
// Connect two IRQs
void avr_connect_irq(avr_irq_t *src, avr_irq_t *dst);

// Raise (signal) the IRQ
void avr_raise_irq(avr_irq_t *irq, uint32_t value);

// Register a callback to be executed on signal raise
void avr_irq_register_notify(
    avr_irq_t *irq,
    avr_irq_notify_t notify,
    void *param);
```

SIMAVR Resources

- Overview of simavr inner workings: http://polprog.net/papiery/avr/simavr.pdf
- Simavr repository: https://github.com/buserror/simavr