Computational hardware

- Digital logic (CSE370)
  - Gates and flip-flops: glue logic, simple FSMs, registers
  - Two-level PLDs: FSMs, muxes, decoders
- Programmable logic devices (CSE370, CSE467)
  - Field-programmable gate arrays: FSMs, basic data-paths
  - Mapping algorithms to hardware
- Microprocessors (CSE378)
  - General-purpose computer
  - Instructions can implement complex control structures
  - Supports computations/manipulations of data in memory

Microprocessors

- Arbitrary computations
  - Arbitrary control structures
  - Arbitrary data structures
  - Specify function at high-level and use compilers and debuggers
- Microprocessors can lower hardware costs
  - If function requires too much logic when implemented with gates/FFs
    - Operations are too complex, better broken down as instructions
    - Lots of data manipulation (memory)
  - If function does not require higher performance of customized logic
    - Ever-increasing performance of processors puts more and more applications in this category
    - Minimize the amount of external logic
Microprocessor basics

- Composed of three parts
  - Data-path: data manipulation and storage
  - Control: determines sequence of actions executed in data-path and interactions to be had with environment
  - Interface: signals seen by the environment of the processor

- Instruction execution engine: fetch/execute cycle
  - Flow of control determined by modifications to program counter
  - Instruction classes:
    - Data: move, arithmetic and logical operations
    - Control: branch, loop, subroutine call
    - Interface: load, store from external memory

Microprocessor basics (cont’d)

- Can implement arbitrary state machine with auxiliary data-path
  - Control instructions implement state diagram
  - Registers and ALUs act as data storage and manipulation
  - Interaction with the environment through memory interface
  - How are individual signal wires sensed and controlled?
Microprocessor organization

- Controller
  - Inputs: from ALU (conditions), instruction read from memory
  - Outputs: select inputs for registers, ALU operations, read/write to memory
- Data-path
  - Register file to hold data
  - Arithmetic logic unit to manipulate data
  - Program counter (to implement relative jumps and increments)
- Interface
  - Data to/from memory (address and data registers in data path)
  - Read/write signals to memory (from control)

General-purpose processor

- Programmed by user
- New applications are developed routinely
- General-purpose
  - Must handle a wide ranging variety of applications
- Interacts with environment through memory
  - All devices communicate through memory data
  - Direct-memory access (DMA) operations between disk and I/O devices
  - Dual-ported memory (e.g., display screen)
  - Generally, oblivious to passage of time
Embedded processor

- Typically programmed once by manufacturer of system
  - Rarely does the user load new software
- Executes a single program (or a limited suite) with few parameters
- Task-specific
  - Can be optimized for a specific application
- Interacts with environment in many ways
  - Direct sensing and control of signal wires
  - Communication protocols to environment and other devices
  - Real-time interactions and constraints
  - Power-saving modes of operation to conserve battery power

Why embedded processors?

- High overhead in building a general-purpose system
  - Storing/loading programs
  - Operating system manages running of programs and access to data
  - Shared system resources (e.g., system bus, large memory)
  - Many parts
    - Communication through shared memory/bus
    - Each I/O device often requires its own separate hardware unit
- Optimization opportunities
  - As much hardware as necessary for application
    - Cheaper, portable, lower-power systems
  - As much software as necessary for application
    - Doesn’t require a complete OS, get a lot done with a smaller processor
  - Can integrate processor, memory, and I/O devices on to a single chip
Typical general-purpose architecture

CPU + Cache

Memory

Display (with dual-port video RAM)

Disk

I/O (serial line, keyboard, mouse)

Network Interface

Sound Interface

all the parts around the processor are usually required

standard interfaces

Typical task-specific architecture

Diagnostic tools

Microcontroller (CPU+mem+...)

Special I/O Device Driver

General Purpose I/O

A/D-D/A Conversion

Timers

ROM

RAM

Custom Logic

Feedback/ control through physical environment

low-speed interactions

medium-speed interactions

high-speed interactions

standard interface

any of the parts around the microcontroller are optional

Compilers

Operating systems

Narrow performance-oriented view

Compiler Interfaces

Operating System Interfaces
How does this change things?

- Sense and control of environment
  - Processor must be able to “read” and “write” individual wires
  - Controls I/O interfaces directly
- Measurement of time
  - Many applications require precise spacing of events in time
  - Real-time is not the same thing as fast as possible
  - Reaction times to external stimuli may be constrained
- Communication
  - Protocols must be implemented by processor
  - Integrate I/O device or emulate in software
  - Capability of using external device when necessary

Interactions with the environment

- Basic processor only has address and data busses to memory
- Inputs are read from memory
- Outputs are written to memory
- Thus, for a processor to sense/control signal wires in the environment they must be made to appear as memory bits
  - How do we make wires look like memory?
  - How long does it take to do these things?
### Sensing external signals

- Map external wire to a bit in the address space of the processor
- External register or latch buffers values coming from environment
  - Map register into address space
    - Decoder selects register for reading
  - Output enable (OE) to get value on to data bus
    - Lets many registers use the same data bus

### Controlling external signals

- Map external wire to a bit in the address space of the processor
- Connect output of memory-mapped register to environment
  - Map register into address space
    - Decoder selects register for writing (holds value indefinitely)
  - Input enable (EN) to take value from data bus
    - Lets many registers use the same data bus
Time and instruction execution

- Keep track of detailed timing of each instruction's execution
  - Highly dependent on code
  - Hard to use compilers
  - Not enough control over code generation
  - Interactions with caches/instruction-buffers
- Loops to implement delays
  - Keep track of time in counters
  - Keeps processor busy counting and not doing other useful things
- Timer
  - Take differences between measurements at different points in code
  - Keeps running even if processor is idle to save power
  - An independent "co-processor" to main processor

Time measurement via parallel timers

- Separate and parallel counting unit(s)
  - Co-processor to microprocessor
  - Does not require microprocessor intervention
  - May be a simple counter or a more featured real-time clock
  - Alarms can be set to generate interrupts
- More interesting timer units
  - Self reloading timers for regular interrupts
  - Pre-scaling for measuring larger times
  - Started by external events
Input/output events

- Input capture
  - Record time when input event occurred
  - Can be used in later handling of event

- Output compare
  - Set output event to happen at a point in the future
  - Reactive outputs
    - e.g., set output to happen a pre-defined time after some input
  - Processor can go on to do other things in the meantime

System bus based communication

- Extend address/data bus outside of chip
- Use specialized devices to implement communication protocol
- Map devices and their registers to memory locations
- Read/write data to receive/send buffers in shared memory or device
- Poll registers for status of communication
- Wait for interrupt from device on interesting events
  - Send completed
  - Receive occurred
Support for communication protocols

- **Built-in device drivers**
  - For common communication protocols
    - e.g., RS232, IrDA, USB, Bluetooth, etc.
  - Serial-line protocols most common as they require fewer pins

- **Serial-line controller**
  - Special registers in memory space for interaction
  - May use timer unit(s) to generate timing events
    - For spacing of bits on signal wire
    - For sampling rate

- **Increase level of integration**
  - No external devices
  - May further eliminate need for shared memory or system bus

Microcontrollers

- **Embedded processor with much more integrated on same chip**
  - Processor core + co-processors + memory
  - ROM for program memory, RAM for data memory, special registers to interface to outside world
  - Parallel I/O ports to sense and control wires
  - Timer units to measure time in various ways
  - Communication subsystems to permit direct links to other devices
Microcontrollers (cont’d)

- Other features not usually found in general-purpose CPUs
  - Expanded interrupt handling capabilities
    - Multiple interrupts with priority and selective enable/disable
    - Automatic saving of context before handling interrupt
    - Interrupt vectoring to quickly jump to handlers
  - More instructions for bit manipulations
    - Support operations on bits (signal wires) rather than just words
- Integrated memory and support functions for cheaper system cost
  - Built-in EEPROM, Flash, and/or RAM
  - DRAM controller to handle refresh
  - Page-mode support for faster block transfers
### The AVR Microcontroller Family

#### tiny11, tiny12, tiny15, tiny28

<table>
<thead>
<tr>
<th></th>
<th>tiny11</th>
<th>tiny12</th>
<th>tiny15</th>
<th>tiny28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>28/32</td>
</tr>
<tr>
<td>Flash</td>
<td>1 KB</td>
<td>1 KB</td>
<td>1 KB</td>
<td>2 KB</td>
</tr>
<tr>
<td>EEPROM</td>
<td>-</td>
<td>64 B</td>
<td>64 B</td>
<td>-</td>
</tr>
<tr>
<td>PWMs</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADC</td>
<td>-</td>
<td>-</td>
<td>4@10-bit</td>
<td>-</td>
</tr>
</tbody>
</table>

#### S1200, S2323, S2343, S2313

<table>
<thead>
<tr>
<th></th>
<th>S1200</th>
<th>S2323</th>
<th>S2343</th>
<th>S2313</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>20</td>
<td>8</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>Flash</td>
<td>1 KB</td>
<td>2 KB</td>
<td>2 KB</td>
<td>2 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>-</td>
<td>128 B</td>
<td>128 B</td>
<td>128 B</td>
</tr>
<tr>
<td>EEPROM</td>
<td>64 B</td>
<td>128 B</td>
<td>128 B</td>
<td>128 B</td>
</tr>
<tr>
<td>UART</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>PWM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

#### S4433, S515, VC8534, S5556

<table>
<thead>
<tr>
<th></th>
<th>S4433</th>
<th>S515</th>
<th>VC8534</th>
<th>S5556</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>28/32</td>
<td>40/44</td>
<td>48</td>
<td>40/44</td>
</tr>
<tr>
<td>Flash</td>
<td>4 KB</td>
<td>8 KB</td>
<td>8 KB</td>
<td>8 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>128 B</td>
<td>512 B</td>
<td>256 B</td>
<td>512 B</td>
</tr>
<tr>
<td>EEPROM</td>
<td>256 B</td>
<td>512 B</td>
<td>512 B</td>
<td>512 B</td>
</tr>
<tr>
<td>UART</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>PWM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>ADC</td>
<td>6@10-bit</td>
<td>6@10-bit</td>
<td>6@10-bit</td>
<td>-</td>
</tr>
<tr>
<td>RTC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### mega161, mega163, mega32, mega103

<table>
<thead>
<tr>
<th></th>
<th>mega161</th>
<th>mega163</th>
<th>mega32</th>
<th>mega103</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>40/44</td>
<td>40/44</td>
<td>40/44</td>
<td>64</td>
</tr>
<tr>
<td>Flash</td>
<td>16 KB</td>
<td>16 KB</td>
<td>32 KB</td>
<td>128 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>1 KB</td>
<td>1 KB</td>
<td>2 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>EEPROM</td>
<td>512 B</td>
<td>512 B</td>
<td>1 KB</td>
<td>2 KB</td>
</tr>
<tr>
<td>UART</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PWM</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>ADC</td>
<td>-</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
</tr>
<tr>
<td>RTC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JTAG/OCD</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Self Program</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>HW MULT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Brown Out</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>

#### mega32, mega16, mega32, mega64, mega128

<table>
<thead>
<tr>
<th></th>
<th>mega32</th>
<th>mega16</th>
<th>mega32</th>
<th>mega64</th>
<th>mega128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins</td>
<td>28/32</td>
<td>40/44</td>
<td>40/44</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Flash</td>
<td>8 KB</td>
<td>16 KB</td>
<td>32 KB</td>
<td>64 KB</td>
<td>128 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>1 KB</td>
<td>1 KB</td>
<td>2 KB</td>
<td>4 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>EEPROM</td>
<td>512 B</td>
<td>512 B</td>
<td>1 KB</td>
<td>2 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>UART</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>WI</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PWM</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>ADC</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
<td>8@10-bit</td>
</tr>
<tr>
<td>RTC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JTAG/OCD</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Self Program</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HW MULT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Brown Out</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Microcontroller we will be using

- Atmel AVR Microcontroller (ATmega16) – 16 MIPS at 16 MHz
  - 8-bit microcontroller (8-bit data, 16-bit instructions) – RISC Architecture
    - 131 instructions (mostly single-cycle – on-chip 2-cycle multiplier)
    - 32 general-purpose registers
    - Internal and external interrupts
  - Memory
    - instruction (16KB Flash memory – read-while-write)
    - boot ROM (512 Byte EEPROM)
    - data (1K static RAM)
  - Timers/counters
    - 2 8-bit and 1 16-bit timer/counters with compare modes and prescalers
    - Real-time clock (32.768 kHz) with separate oscillator
    - Programmable watchdog timer
  - Serial communication interfaces
    - JTAG boundary-scan interface for programming/debugging
    - Programmable USART (universal synchronous/asynchronous receiver transmitter)
    - Two-wire serial interface (can emulate different communication protocols)
    - SPI serial port (serial peripheral interface)
  - Peripheral features
    - Four channels with support for pulse-width modulation
    - Analog-digital converter (8-channel, 10-bit)
    - Up to 32 general-purpose I/O pins (with interrupt support)
  - Six power saving modes

Why did we pick the ATmega16

- Modern microcontroller
- Easy to use C compiler
- Better performance/power than competitors
  - Microchip PIC
  - Motorola 68HC11
  - Intel 80C51
- Excellent support for 16-bit arithmetic operations
- A lot of registers that eliminate moves to and from SRAM
- Single cycle execution of most instructions
- Very popular family in a variety of applications including many sensor network platforms
ATmega16 Internals

General-Purpose Working Registers

- 32 registers
- 6 are special
  - Used for addressing modes
- Addressed in regular memory space
  - Easier to use instructions
Operations on Register/Immediate Values

- One cycle register/immediate instructions

Instruction Timing
The Five Memory Areas

- General Purpose Register File = 32 B
- Flash Program Memory = 8 KB (≤ 8 MB)
- SRAM Data Memory = 1 KB (≤ 16 MB)
- I/O Memory = 64 B (≤ 64 B)
- EEPROM Data Memory = 512 B (≤ 16 MB)
Data SRAM → Register File (RF)

- “LD Rd,<PTR>” - Load indirect
- “LD Rd,<PTR>+” - Load indirect with post-increment
- “LD Rd,-<PTR>” - Load indirect with pre-decrement
- “LDD Rd,<PTR>+q” - Load indirect with displacement (0-63)*

* PTR = X, Y or Z

Data SRAM ← Register File (RF)

- “ST <PTR>,Rd” - Store indirect
- “ST <PTR>+,Rd” - Store indirect with post-increment
- “ST -<PTR>,Rd” - Store indirect with pre-decrement
- “STD <PTR>+q,Rd” - Store indirect with displacement (0-63) *

* PTR = X, Y or Z
Data Transfer Program Memory −→ RF

- **"LDI"**  Load a register with an immediate value (1 Cycle) *
- **"LPM"**  Transfer a byte from program Memory@Z to R0 (3 Cycles)
- **"LPM Rd,Z"**  Transfer a byte from program Memory@Z to Rd (3 Cycles)
- **"LPM Rd,Z+"**  As above but with post-increment of the Z pointer

* Works on R16 - R31

Register File −→ Register File

- **"OUT"**  Transfer a byte from RF to I/O
- **"IN"**  Transfer a byte from I/O to RF
- **"MOV"**  Copy a register to another register
- **"MOVW"**  Copy a register pair to another register pair. Aligned.
C-like Addressing Modes (1)

- Auto Increment/Decrement:
  - C Source:
    ```c
    unsigned char *var1, *var2;
    *var1++ = *--var2;
    ```
  - Generated code:
    ```
    LD R16,-X
    ST Z+,R16
    ```

C-Like Addressing Modes (2)

- Indirect with displacement
- Efficient for accessing arrays and structs

```c
Struct square {
    int x_min;
    int x_max;
    int y_min;
    int y_max;
} my_square;
```

SRAM:

```
Z(my_square) ++

<table>
<thead>
<tr>
<th>x_min</th>
<th>x_max</th>
<th>y_min</th>
<th>y_max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z+2</td>
<td>Z+4</td>
<td>Z+6</td>
<td></td>
</tr>
</tbody>
</table>
```
### The Status Register - SREG

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Interrupt Enable</td>
</tr>
<tr>
<td>6</td>
<td>T Flag Source and Destination for BLD and BST</td>
</tr>
<tr>
<td>5</td>
<td>Half Carry Set if an operation has half carry</td>
</tr>
<tr>
<td>4</td>
<td>Signed Flag Set if Signed Overflow</td>
</tr>
<tr>
<td>3</td>
<td>Overflow Flag Set if Signed Overflow</td>
</tr>
<tr>
<td>2</td>
<td>Negative Flag Set if a Result is Negative</td>
</tr>
<tr>
<td>1</td>
<td>Zero Flag Set if a Result is Zero</td>
</tr>
<tr>
<td>0</td>
<td>Carry Flag Set if an operation has Carry</td>
</tr>
</tbody>
</table>

### Branch on SREG Settings

**Branches if Bit Clear**

- BRID
- BRTC
- BRHC
- BRGE
- BRVC
- BRPL
- BRNE
- BRSH, BRCC

**Branches if Bit Set**

- BRIE
- BRTS
- BRHS
- BRLT
- BRVS
- BRMI
- BREQ
- BRCS, BRLO
A Small C Function

/* Return the maximum value of a table of 16 integers */

int max(int *array)
{
    char a;
    int maximum = SMALLEST_NUMBER; /* this is -32768 */

    for (a=0; a<16; a++)
        if (array[a] > maximum)
            maximum = array[a];
    return (maximum);
}

AVR Assembly output

; 7.    for (a=0; a<16; a++)
LDI    R18, LOW(0)
LDI    R19, 128
CLR    R22
?0001:
    CPI    R22, LOW(16)
    BRCC   ?0000
; 8.    {
; 9.        if (array[a] > maximum)
MOV     R30, R22
CLR     R31
LSL     R30
ADD     R30, R16
ADC     R31, R17
LDD     R20, Z+0
LDD     R21, Z+1
CP     R18, R20
CPC    R19, R21
BRGE   ?0005
    ; 10.            maximum = array[a];
MOV     R18, R20
MOV     R19, R21
?0005:
    ; 11.        }
INC     R22
RJMP   ?0001
?0000:
; 12.    } return (maximum);
MOV     R16, R18
MOV     R17, R19
; 13.    } RET

Code Size: 46 Bytes, Execution time: 335 cycles
I/O Ports General Features

- Push-pull drivers
- High current drive (sinks up to 40 mA)
- Pin-wise controlled pull-up resistors
- Pin-wise controlled data direction
- Fully synchronized inputs
- Three control/status bits per bit/pin
I/O Port Configurations

- 3 Control/Status Bits per Pin
  - DDx Data Direction Control Bit
  - PORTx Output Data or Pull-Up Control Bit
  - PINx Pin Level Bit

<table>
<thead>
<tr>
<th>DDxn</th>
<th>PORTxn</th>
<th>PUD (In SFIOR)</th>
<th>I/O</th>
<th>Pull-up</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Input</td>
<td>Yes</td>
<td>PINx will source current if ext. pulled low.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output Low (Sink)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output High (Source)</td>
</tr>
</tbody>
</table>

Port is Input (Default Configuration)
Port is Open-Collector Input (Switch On Pull-Up)

Port is Output
Sample Code from Lab 1

.include "C:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes\m16def.inc"

.cseg
ldi r16, 0xff
out DDRB, r16
ldi r16, 0x00
out PORTB, r16
loop:
jmp loop

Alternate Port Functions

- Generalizing I/O ports so that they can be used by other I/O devices
### I/O Port Registers

#### Port A Data Register – PORTA

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Port A Data Direction Register – DDRA

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Port A Input Pins Address – PINA

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

---

### Full I/O Register Map (pg. 331)

![Full I/O Register Map](image-url)
Instruction Classes (pg. 333)

- Arithmetic/Logic Instructions
- Data Transfer Instructions
- Program Control Instructions
- Bit Set/Test Instructions

Arithmetic/Logical Instructions

- Add Instructions
  - "ADD" Add Two Registers
  - "ADC" Add Two Registers and Carry
  - "INC" Increment a Register
  - "ADIW" Add Immediate to Word *
- Subtract Instructions
  - "SUB" Subtract Two Registers
  - "SBC" Subtract with Carry Two Registers
  - "SUI" Subtract Immediate from Register*
  - "SBCI" Subtract with Carry Immediate from Register*
  - "DEC" Decrement Register
  - "SBIW " Subtract Immediate From Word**
- Compare Instructions
  - "CP" Compare Two Registers
  - "CPC" Compare with Carry Two Registers
  - "CPI" Compare Register and Immediate*
  - "CPSE" Compare Two Registers and Skip Next Instruction if Equal
16-bit and 32-bit support

- **Carry instructions**
  - Addition, subtraction and comparison
  - Register with register or immediate
  - Zero flag propagation

```
SUB   R16,R24    SUBI   R16,1
SBC   R17,R25    SBCI   R17,0
```

- All branches can be made based on last result
- **Direct 16 bit instructions**
  - Addition and subtraction of small immediates
  - Pointer arithmetics

Subtracting Two 16-Bit Values

- **R1:R0 – R3:R2**  (e.g., $E104 – $E101)
  - Without zero-flag propagation

```
sub   r0,r2
sbc   r1,r3
```

- With zero-flag propagation

```
sub   r0,r2
sbc   r1,r3
```

Wrong!

Correct!
Comparing Two 32-Bit Values

- Example: Compare R3:R2:R1:R0 and R7:R6:R5:R4

    cp   r0,r4
    cpc  r1,r5
    cpc  r2,r6
    cpc  r3,r7

- After last instruction, status register indicates equal, higher, lower, greater (signed), or less than (signed)

Arithmetic/Logical Instructions (cont’d)

- Multiply instructions
  - “MUL”  8x8 → 16  (UxU)
  - “MULS” 8x8 → 16  (SxS)*
  - “MULSU” 8x8 → 16  (SxU)**

    * Works on Registers R16 - R31  
    ** Works on Registers R16-R23

- Logical Instructions
  - “AND” Logical AND Two Registers
  - “ANDI” Logical AND Immediate and Register *
  - “OR” Logical OR Two Registers
  - “ORI” Logical OR Immediate and Register *
  - “EOR” Logical XOR Two Registers

The result is present in R1:R0. All multiplication instructions are 2 cycles.
Arithmetic/Logical Instructions (cont’d)

- Shift / Rotate Instructions
  - “LSL” Logical Shift Left
  - “LSR” Logical Shift Right
  - “ROL” Rotate Left Through Carry
  - “ROR” Rotate Right Through Carry
  - “ASR” Arithmetic Shift Right

![Shift/rotate instructions diagram]

Data Transfer Instruction Types

<table>
<thead>
<tr>
<th>Data Transfer Instruction</th>
<th># of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data SRAM &lt;-&gt; Register File</td>
<td>2</td>
</tr>
<tr>
<td>Program Memory -&gt; Register File</td>
<td>1/3</td>
</tr>
<tr>
<td>I/O Memory &lt;-&gt; Register File</td>
<td>1</td>
</tr>
<tr>
<td>Register File &lt;-&gt; Register File</td>
<td>1</td>
</tr>
</tbody>
</table>
Data Transfer RF <-> SRAM Stack

- "PUSH"  PUSH a register on the stack
  - Decrements stack pointer by 1
  - Decremented by 2 when a return address is pushed on the stack
- "POP"  POP a register from the stack
  - Increments stack pointer by 1
  - Incremented by 2 when a return address is popped off on return

- Stack grows from higher to lower memory locations
  - Must start after $0060 (past I/O registers)
  - Size of pointer varies depending on memory available

Flow Control

- Unconditional Jumps
- Conditional Branches
- Subroutine Call and Returns
Unconditional Jump Instructions

- “RJMP” Relative Jump *
- “JMP” Absolute Jump **

* Reaches ±2K instructions from current program location. Reaches all locations for devices up to 8KBytes (wrapping)
** 4-Byte Instruction

Conditional Branches (Flag Set)

- “BREQ” Branch if Equal
- “BRSH” Branch if Same or Higher
- “BRGE” Branch if Greater or Equal (Signed)
- “BRHS” Branch if Half Carry Set
- “BRCS” Branch if Carry Set
- “BRMI” Branch if Minus
- “BRVS” Branch if Overflow Flag Set
- “BRTS” Branch if T Flag Set
- “BRIE” Branch if Interrupt Enabled
Subroutine Call and Return

- "RCALL" Relative Subroutine Call *
- "CALL" Absolute Subroutine Call **

- "RET" Return from Subroutine
- "RETI" Return from Interrupt Routine

* Reaches ±2K instructions from current program location.
  Reaches all locations for devices up to 8KBytes (wrapping)

** 4-Byte Instruction

Bit Set/Clear and Bit Test Instructions

- "SBR" Set Bit(s) in Register *
- "SBI" Set Bit in I/O Register **
- "SBRS" Skip if Bit in Register Set
- "SBIS" Skip if Bit in I/O Register Set **
- "CBR" Clear Bit(s) in Register *
- "CBI" Clear Bit in I/O Register **
- "SBRC" Skip if Bit in Register Clear
- "SBIC" Skip if Bit in I/O Register Clear **

* Works on Registers R16 - R31
** Works on I/O Addresses $00 - $1F
Programming the ATmega16

- Traditional in-system programming
  - In-system programmable FLASH, EEPROM, and lock bits
  - Programmable at all frequencies
  - Programmable at any value of Vcc above 2.7V
  - Only four pins + ground required
  - Requires adapter device to control programming pins

- Self programming
  - The AVR reprograms itself without any external components
  - Re-programmable through any communication interface
    - Does not have to be removed from board
    - Uses existing communication ports
  - Critical functions still operating
    - device is running during programming

AVR JTAG Interface

- Complies to IEEE std 1149.1 (JTAG)
- Boundary-scan for efficient PCB test
  - Standard for interconnection test
  - All I/O pins controllable and observable from tester
- On-chip debugging in production
JTAG In System Programming

- The JTAG interface can be used to program the Flash and EEPROM
- Save time and production cost
  - No additional programming stage
  - Programming time independent of system clock

JTAG In-Circuit Emulator

- Controlled by AVR Studio
- Real-Time emulation in actual silicon
  - Debug the real device at the target board
  - Talks directly to the device through the 4-pin JTAG interface
- Supports
  - Program and data breakpoints
  - Full execution control
  - Full I/O-view and watches
AVR Studio

- Integrated development environment for AVR
- Front end for the AVR simulator and emulators
- C and assembly source level debugging
- Supports third party compilers
- Maintains project information
- Freely available from www.atmel.com
- Third-party compilers