# Contents

Title Page i  
Contents ii  
List of Figures iv  

1 Introduction  
1.1 Welcome! 1  

2 CS150 AVR Instruction Subset 2  
2.1 Preliminaries 2  
2.2 Instruction Encoding/Decoding Amulet 3  
2.3 Instruction Legend 4  
2.3.1 SREG: The ATmega328 status register 4  
2.3.2 Registers and Operands 4  
2.3.3 Stack 5  
2.3.4 Flags 5  
2.3.5 Boolean Equations 5  
2.4 The Instructions 6  
ADC 7  
ADD 8  
AND 9  
ANDI 10  
ASR 11  
BRBC 12  
BRBS 13  
CALL 14  
COM 15  
CP 16  
CPI 17  
EOR 18  
IN 19  
JMP 20  
LDI 21  
LDS 22
3 ATmega328 Ports
List of Figures

3.1 Memory-Mapped ATmega328 Ports .................................... 37
3.2 Configuring PORTD Pin 2 as Input ................................. 38
Chapter 1

Introduction

1.1 Welcome!

Welcome to Computer Organization and Architecture (CS150) at the University of Idaho. We are certainly happy to have you in this course. CS150 is a multidisciplinary course, so you will rub shoulders with aspiring mages from fields such as Computer Science, Computer Engineering, Electrical Engineering, and usually a few others. Regardless of your background, we are glad that you’ve chosen to enroll in the course and we hope that you will discover interesting ingredients and incantations to enhance your repertoire. The passages ahead are twisty and difficult, but there is nothing in the course that you can’t accomplish if you “put your mind to it” as the saying goes. So remember to keep your toad spittle warm and your mandrake root dry and enjoy the semester!
Chapter 2

CS150 AVR Instruction Subset

2.1 Preliminaries

Although we study precepts that span all processors in this course, we apply these precepts to one processor in particular. That processor is the ATMEL ATmega328. The ATmega328 is a member of the AVR family of processors, and implements the AVR instruction set. The AVR instruction set is a set of instructions that are typically found on all processors in the AVR family.

The ATmega328 implements hundreds of instructions. In CS150 we only study (and use) a select subset of all instructions available on the ATmega328. This subset is referred to as the “CS150 AVR instruction subset” and is detailed in the remainder of this chapter. You are not required to understand or use any of the AVR instructions implemented by the ATmega328 that do not appear in the CS150 AVR instruction subset. Moreover, you are not permitted to use any of the AVR instructions available on the ATmega328 that do not appear in the CS150 AVR instruction subset. All work on assignments in this course must refer only to instructions contained within this subset.
### 2.2 Instruction Encoding/Decoding Amulet

#### AVR Instruction Subset

**ALU Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>ADD</td>
</tr>
<tr>
<td>0001</td>
<td>ADDC</td>
</tr>
<tr>
<td>0001</td>
<td>AND</td>
</tr>
<tr>
<td>0010</td>
<td>OR</td>
</tr>
<tr>
<td>0100</td>
<td>XOR</td>
</tr>
</tbody>
</table>

**Immediate Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>CPI</td>
</tr>
<tr>
<td>0110</td>
<td>ORI</td>
</tr>
<tr>
<td>0111</td>
<td>ANDI</td>
</tr>
<tr>
<td>1110</td>
<td>LDI</td>
</tr>
</tbody>
</table>

**Unary Logical Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>COM</td>
</tr>
<tr>
<td>0001</td>
<td>NEG</td>
</tr>
<tr>
<td>0101</td>
<td>ASR</td>
</tr>
<tr>
<td>0110</td>
<td>LSR</td>
</tr>
</tbody>
</table>

**Branch Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>BRB</td>
</tr>
<tr>
<td>1001</td>
<td>BRBC</td>
</tr>
<tr>
<td>1100</td>
<td>CALL</td>
</tr>
<tr>
<td>1110</td>
<td>JMP</td>
</tr>
</tbody>
</table>

**Load/Store Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>LDS</td>
</tr>
<tr>
<td>0110</td>
<td>STB</td>
</tr>
</tbody>
</table>

**Input/Output Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>IN</td>
</tr>
<tr>
<td>1010</td>
<td>OUT</td>
</tr>
</tbody>
</table>

**Call/Jump Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>POP</td>
</tr>
<tr>
<td>1001</td>
<td>PUSH</td>
</tr>
</tbody>
</table>

**Return Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>RET</td>
</tr>
<tr>
<td>1100</td>
<td>RETI</td>
</tr>
</tbody>
</table>

**Relative Jump Instructions**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110</td>
<td>RJMP</td>
</tr>
<tr>
<td>1000</td>
<td>RCALL</td>
</tr>
</tbody>
</table>
2.3 Instruction Legend

This section is a legend for understanding the instructions that appear in the remainder of this chapter. Each instruction is copied from the “ATMEL AVR Instruction Set Manual” reference which is on the course website, although there are minor modifications. The following terms and symbols are used in the narrative of these instructions:

2.3.1 SREG: The ATmega328 status register

C: The carry flag
Z: The zero flag
N: The negative flag
V: The overflow flag
S: The sign flag
H: The half-carry flag
T: The transfer flag
I: The interrupt flag

2.3.2 Registers and Operands

Rd: Destination register in the Register File
Rr: Source register in the Register File
R: Result after instruction has been executed
ALU_RESULT: Register inside the ALU where a result is temporarily stored
K: Constant data
k: Constant address
A: I/O space address
s: Part of a 3-bit field indexing a bit in the status register
2.3.3 Stack

STACK: Location used for storing return address and pushed registers

SP: Stack Pointer (address of top of stack)

2.3.4 Flags

? : Flag is affected by a given instruction
- : Flag is unaffected by a given instruction
1 : Flag is always set by a given instruction
0 : Flag is always cleared by a given instruction

2.3.5 Boolean Equations

\( \equiv \) : Concurrent assignment

\( \bullet \) : Logical AND

\( + \) : Logical OR

\( \oplus \) : Logical XOR

\( \overline{X} \) : Logical NOT (complement) of X
2.4 The Instructions

The rest of this chapter covers all the ATmega328 instructions that are members of the “CS150 AVR Instruction Subset.” Each of these instructions is also contained in the “ATMEL AVR Instruction Set Manual” reference which is on the course website. It is advisable to avail yourself of both these resources when studying these instructions.
**ADC: Add with carry**

**Description**
Adds two registers and the contents of the C bit in the **SREG** and places the result in the destination register **Rd**.

**Operation**
\[ Rd \leftarrow Rd + Rr + C \]

**Syntax**
ADC Rd,Rr

**Operands**
0 ≤ d ≤ 31, 0 ≤ r ≤ 31

**Program Counter**
PC ← PC + 1

**Instruction Format**
0001 11rd dddd rrrr

**Status Register Usage**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
</table>
| - | - | - | - | ? | ? | ? | ?

**H** ← \( Rd3 \cdot Rr3 + Rr3 \cdot \overline{R3} + Rd3 \cdot \overline{R3} \)
Set if there was a carry from bit 3.

\[ S \leftarrow N \oplus V, \text{ for signed tests} \]

\[ V \leftarrow Rd7 \cdot Rr7 \cdot \overline{R7} + \overline{Rd7} \cdot Rr7 \cdot R7 \]
Set if two’s complement overflow resulted from the operation.

\[ N \leftarrow R7 \]
Set if MSB of the result is set.

\[ Z \leftarrow \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot R3 \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0} \]
Set if the result of the operation was 0.

\[ C \leftarrow Rd7 \cdot Rr7 + Rr7 \cdot \overline{R7} + Rd7 \cdot \overline{R7} \]
Set if there was a carry from the MSB of the result.

**Example**

; add r1:r0 to r3:r2
add r2,r0 ; add the low byte
adc r3,r1 ; add with carry the high byte

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**ADD:** Add without carry

**Description**
Adds two registers and places the result in the destination register Rd.

**Operation**

\[ \text{Rd} \leftarrow \text{Rd} + \text{Rr} \]

**Syntax**

\[ \text{ADD Rd}, \text{Rr} \]

**Operands**

\[ 0 \leq d \leq 31, \ 0 \leq r \leq 31 \]

**Program Counter**

\[ \text{PC} \leftarrow \text{PC} + 1 \]

**Instruction Format**

\[ 0000 \ 11\text{rd} \ \text{dddd} \ \text{rrrr} \]

**Status Register Usage**

\[ \begin{array}{cccccccc}
I & T & H & S & V & N & Z & C \\
\hline
\end{array} \]

\[ H \leftarrow R_{d3} \bullet R_{r3} + R_{r3} \bullet \overline{R_{d3}} + R_{d3} \bullet \overline{R_{r3}} \]

Set if there was a carry from bit 3.

\[ S \leftarrow N \oplus V, \text{ for signed tests} \]

\[ V \leftarrow R_{d7} \bullet R_{r7} \bullet \overline{R_{r7}} + R_{d7} \bullet \overline{R_{r7}} \bullet R_{r7} \]

Set if two’s complement overflow resulted from the operation.

\[ N \leftarrow R_{7} \]

Set if MSB of the result is set.

\[ Z \leftarrow R_{7} \bullet R_{6} \bullet R_{5} \bullet R_{4} \bullet R_{3} \bullet R_{2} \bullet R_{1} \bullet R_{0} \]

Set if the result of the operation was 0.

\[ C \leftarrow R_{d7} \bullet R_{r7} + R_{r7} \bullet \overline{R_{r7}} + R_{d7} \bullet \overline{R_{r7}} \]

Set if there was a carry from the MSB of the result.

**Example**

\[ \text{add} \ r1, r2 \quad ; \text{add} \ r2 \text{ to } r1 \]
\[ \text{add} \ r28, r28 \ ; \text{add} \ r28 \text{ to itself (shift left r28 by one)} \]

**Space/Time**

This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
AND: Logical AND

Description
Computes the logical and of the contents of register Rd and register Rr and stores the result in register Rd.

Operation
Rd ← Rd • Rr

Syntax
AND Rd,Rr

Operands
0 ≤ d ≤ 31, 0 ≤ r ≤ 31

Program Counter
PC ← PC + 1

Instruction Format
0010 00rd dddd rrrr

Status Register Usage

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>-</td>
</tr>
</tbody>
</table>

S ← N ⊕ V, for signed tests

V ← 0
V is always cleared by this instruction.

N ← R7
Set if MSB of the result is set.

\[ Z ← R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0 \]
Set if the result of the operation was 0.

Example

and r2,r3 ; bitwise and r2 and r3, result in r2
ldi r16,1 ; load bitmask 0000 0001 into r16
and r2,r16 ; isolate bit 0 in r2

Space/Time
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**ANDI: Logical AND with Immediate**

**Description**
Computes the logical and of the contents of register $Rd$ and constant $K$ and stores the result in register $Rd$.

**Operation**
$$Rd \leftarrow Rd \cdot K$$

**Syntax**

<table>
<thead>
<tr>
<th>ANDI Rd,K</th>
<th>Operands</th>
<th>Program Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$16 \leq d \leq 31, 0 \leq K \leq 255$</td>
<td>$PC \leftarrow PC + 1$</td>
<td></td>
</tr>
</tbody>
</table>

**Instruction Format**

0111 KKKK dddd KKKK

**Status Register Usage**

- $I \leftarrow - - -$  
- $T \leftarrow - - -$  
- $H \leftarrow - - -$  
- $V \leftarrow 0$  
- $N \leftarrow R7 \oplus V$, for signed tests  
- $Z \leftarrow R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0$  
- $S \leftarrow N \oplus V$, for signed tests

$V$ is always cleared by this instruction.

$N \leftarrow R7$

Set if MSB of the result is set.

$Z \leftarrow R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0$

Set if the result of the operation was 0.

**Example**

```c
andi r17,$0f ; clear upper nibble of r17
andi r18,$10 ; isolate bit 4 in r18
andi r19,$aa ; clear bits 0, 2, 4, and 6 in r19
```

**Space/Time**

This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**ASR**: Arithmetic Shift Right

**Description**
Shifts all bits in Rd one place to the right. Bit 7 of Rd is held constant. Bit 0 of Rd is loaded into the C flag of the SREG. This operation effectively divides a signed value by 2 without changing its sign. The C flag can be used to round the result.

**Operation**

**Syntax**

<table>
<thead>
<tr>
<th>ASR</th>
<th>Rd</th>
</tr>
</thead>
</table>

**Operands**

<table>
<thead>
<tr>
<th>Syntax Operands</th>
<th>Program Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ≤ d ≤ 31</td>
<td>PC ← PC + 1</td>
</tr>
</tbody>
</table>

**Instruction Format**

1001 010d dddd 0101

**Status Register Usage**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td>?</td>
</tr>
</tbody>
</table>

- \( S \leftarrow N \oplus V \), for signed tests
- \( V \leftarrow N \oplus C \) (for N and C after the shift)
- \( N \leftarrow R7 \)
  - Set if MSB of the result is set.
- \( Z \leftarrow \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot R1 \cdot R0 \)
  - Set if the result of the operation was 0.
- \( C \leftarrow R0 \)
  - Set if the LSB of Rd was set before the shift.

**Example**

- ldi r16, $10 ; load 16 into r16
- asr r16 ; r16 = r16 / 2

**Space/Time**

This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**BRBC**: Branch if Bit is Clear

**Description**
Conditional relative branch predicated by a single bit in the status register (SREG). This instruction tests a single bit in SREG specified by the programmer. If the specified bit in the SREG is clear, this instruction branches to an instruction relative to the PC. If the specified bit in the SREG is set, no branch is taken. This instruction branches relative to the PC in either direction (PC − 63 ≤ destination ≤ PC + 64). The parameter k is the offset from the PC and is represented in two’s complement form.

**Operation**
if SREG[s] = 0 then PC ← PC + k

**Syntax**
BRBC s,k

**Operands**
0 ≤ s ≤ 7, -64 ≤ k ≤ 63

**Program Counter**
PC ← PC + 1

**Instruction Format**
1111 01kk kkkk ksss

**Status Register Usage**
I T H S V N Z C
- - - - - - - -

This instruction does not modify any SREG bits.

**Example**
cpi r20,0 ; does r20 contain the value 0?
brbc 1,IsFalse ; branch to IsFalse if the Z flag is clear

... IsFalse: nop ; branch destination

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 2 machine cycles to complete if the predicate is true, or 1 if the predicate is false.
**BRBS**: Branch if Bit is Set

**Description**
Conditional relative branch predicated by a single bit in the status register (**SREG**). This instruction tests a single bit in **SREG** specified by the programmer. If the specified bit in the **SREG** is set, this instruction branches to an instruction relative to the **PC**. If the specified bit in the **SREG** is clear, no branch is taken. This instruction branches relative to the **PC** in either direction ($PC - 63 \leq \text{destination} \leq PC + 64$). The parameter $k$ is the offset from the **PC** and is represented in two’s complement form.

**Operation**
if $SREG[s] = 1$ then $PC \leftarrow PC + k$

**Syntax**
<table>
<thead>
<tr>
<th>BRBS $s,k$</th>
</tr>
</thead>
</table>

**Operands**
$0 \leq s \leq 7, -64 \leq k \leq 63$

**Program Counter**
$PC \leftarrow PC + 1$

**Instruction Format**

```
1111 00kk kkkk ksss
```

**Status Register Usage**

```
I T H S V N Z C
- - - - - - - -
```

This instruction does not modify any **SREG** bits.

**Example**
```
cpi r20,0 ; does r20 contain the value 0?
brbs 1,IsTrue ; branch to IsTrue if the Z flag is set
... IsTrue: nop ; branch destination
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 2 machine cycles to complete if the predicate is true, or 1 if the predicate is false.
CALL: Call to a Subroutine

Description
This instruction makes an unconditional absolute branch to a subroutine located anywhere within Program Memory. This instruction stores the return address (the address of the instruction immediately after the CALL) on the stack. After the return address is stored on the stack, this instruction decrements the stack pointer by 2 (uses a post-decrement scheme).

Operation
PC $\leftarrow k$

Syntax | Operands | Program Counter | Stack
---|---|---|---
CALL $k$ | $0 \leq k \leq 32K$ | PC $\leftarrow k$ | STACK $\leftarrow$ PC

SP $\leftarrow$ SP $-$ 2

Instruction Format
1001 010k kkkk 111k
kkkk kkkk kkkk

Status Register Usage
I T H S V N Z C
- - - - - - - -

This instruction does not modify any SREG bits.

Example
```assembly
ldi r16, $a5 ; load r16 with sanity value
call CheckSanity ; check for sanity
nop
... CheckSanity:
cpi r16, $a5 ; does r16 contain the value 0xa5?
brbc 1, Insane ; if it doesn’t something is very wrong
ret
... Insane:
rjmp Insane ; stay right here cause we’re loopy
```

Space/Time
This instruction is 2 instruction words (4 bytes) wide and takes 4 machine cycles to complete.
**COM: One’s Complement**

**Description**
This instruction computes the one’s complement of the value in Rd and stores the result in Rd.

**Operation**
\[ Rd \leftarrow \$FF - Rd \]

**Syntax**

<table>
<thead>
<tr>
<th>COM</th>
<th>Rd</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Operands</th>
<th>Program Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ≤ d ≤ 31</td>
<td>PC \leftarrow PC + 1</td>
</tr>
</tbody>
</table>

**Instruction Format**

1001 010d dddd 0000

**Status Register Usage**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>?</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>1</td>
</tr>
</tbody>
</table>

\( S \leftarrow N \oplus V \), for signed tests

\( V \leftarrow 0 \)

\( V \) is always cleared by this instruction.

\( N \leftarrow R7 \)

Set if MSB of the result is set.

\( Z \leftarrow R7 \cdot R6 \cdot R5 \cdot R4 \cdot \overline{R3} \cdot R2 \cdot R1 \cdot \overline{R0} \)

Set if the result of the operation was 0.

\( C \leftarrow 1 \)

\( C \) is always set by this instruction.

**Example**

```
com r4 ; take one’s complement of r4
brbs 1,IsZero ; branch if result is zero
...
IsZero: nop ; branch destination
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**CP: Compare Registers**

**Description**
This instruction compares the values of registers \( R_d \) and \( R_r \). The values in \( R_d \) and \( R_r \) are not modified by this instruction.

**Operation**

\[
ALU_{\text{Result}} \leftarrow R_d - R_r
\]

**Syntax**

\[ CP \ R_d,R_r \]

**Operands**

\[ 0 \leq d \leq 31, 0 \leq r \leq 31 \]

**Program Counter**

\[ PC \leftarrow PC + 1 \]

**Instruction Format**

\[ 0001 01rd \ \text{dddd} \ \text{rrrr} \]

**Status Register Usage**

\[ \text{H} \leftarrow R_d 3 \cdot R_r 3 + R_r 3 \cdot R_3 + R_3 \cdot \overline{R_d 3} \]

Set if there was a borrow from bit 3.

\[ S \leftarrow N \oplus V, \text{ for signed tests} \]

\[ V \leftarrow R_d 7 \cdot R_r 7 \cdot \overline{R_r 7} + \overline{R_d 7} \cdot \overline{R_r 7} \cdot R_7 \]

Set if two’s complement overflow resulted from the operation.

\[ N \leftarrow R_7 \]

Set if MSB of the result is set.

\[ Z \leftarrow \overline{R_d 7} \cdot \overline{R_r 7} \cdot \overline{R_r 7} \cdot R_7 + R_7 \cdot \overline{R_d 7} \]

Set if the result of the operation was 0.

\[ C \leftarrow \overline{R_d 7} \cdot R_r 7 \cdot \overline{R_r 7} \cdot R_7 + R_7 \cdot \overline{R_d 7} \]

Set if the absolute value of the contents of \( R_r \) is greater than the absolute value of the contents of \( R_d \).

**Example**

\[ \text{cp } r_4, r_19 \]

\[ \text{brbc } 1, \text{NotEqual} ; \text{branch if } r_4 \neq r_19 \]

... NotEqual: nop ; branch destination

**Space/Time**

This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
CPI: Compare Register with Immediate

Description
This instruction compares the value of register Rd and a constant value. The value in Rd is not modified by this instruction.

Operation
$ALU_{Result} \leftarrow Rd - K$

Syntax
$CPI \text{ } Rd,K$

Operands
$16 \leq d \leq 31, \ 0 \leq K \leq 255$

Program Counter
$PC \leftarrow PC + 1$

Instruction Format
0011 KKKK dddd KKKK

Status Register Usage

\begin{align*}
I & \leftarrow \text{ ? ? ? ? ? } \\
T & \leftarrow \text{ ? ? ? ? ? } \\
H & \leftarrow \overline{Rd3} \cdot K3 + K3 \cdot R3 + R3 \cdot \overline{Rd3}
\end{align*}

Set if there was a borrow from bit 3.

\begin{align*}
S & \leftarrow N \oplus V, \text{ for signed tests} \\
V & \leftarrow Rd7 \cdot \overline{R7} \cdot \overline{R5} + \overline{Rd7} \cdot K7 \cdot R7
\end{align*}

Set if two’s complement overflow resulted from the operation.

\begin{align*}
N & \leftarrow R7 \\
& \text{Set if MSB of the result is set.}
\end{align*}

\begin{align*}
Z & \leftarrow \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot R4 \cdot \overline{R3} \cdot \overline{R2} \cdot R1 \cdot \overline{R0} \\
& \text{Set if the result of the operation was 0.}
\end{align*}

\begin{align*}
C & \leftarrow \overline{Rd7} \cdot K7 + K7 \cdot R7 + R7 \cdot \overline{Rd7} \\
& \text{Set if the absolute value of the contents of K is greater than the} \\
& \text{absolute value of the contents of Rd.}
\end{align*}

Example
\begin{verbatim}
cpi r19,$CC ; compare r19 with 0xCC
brbs 1,Equal ; branch if r19 = 0xCC
...
Equal: nop ; branch destination
\end{verbatim}

Space/Time
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**EOR**: Exclusive OR

**Description**
This instruction computes the logical exclusive-or of register Rd and register Rr and places the result in the destination register Rd.

**Operation**
\[ \text{Rd} \leftarrow \text{Rd} \oplus \text{Rr} \]

**Syntax**
EOR Rd,Rr

**Operands**
- \( 0 \leq d \leq 31 \)
- \( 0 \leq r \leq 31 \)

**Program Counter**
\( \text{PC} \leftarrow \text{PC} + 1 \)

**Instruction Format**
0010 01rd dddd rrrr

**Status Register Usage**
- \( S \leftarrow N \oplus V \), for signed tests
  - \( V \leftarrow 0 \)
  - V is always cleared by this instruction.
  - \( N \leftarrow R7 \)
    - Set if MSB of the result is set.
  - \( Z \leftarrow R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
    - Set if the result of the operation was 0.

**Example**
- \( \text{eor r4,r4} \); clear all bits in r4
- \( \text{eor r0,r22} \); bitwise xor of r0 and r22

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
IN: Load an I/O Location into a Register

Description
This instruction loads the data at an address in I/O space into register Rd.

Operation
$Rd \leftarrow I/O(A)$

Syntax Operands Program Counter
IN Rd,A  $0 \leq d \leq 31$, $0 \leq A \leq 63$  PC $\leftarrow$ PC + 1

Instruction Format
1011 0AAd dddd AAAA

Status Register Usage
I T H S V N Z C
- - - - - - - -

This instruction does not modify any SREG bits.

Example
in r16,5 ; load value of PORTB into r16
cpi r16,$ff ; check if all bits in r16 are set
brbs 1,AllSet ; branch if all bits are set in r16
...
AllSet: nop ; branch destination

Space/Time
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
JMP: Jump

Description
This instruction makes an unconditional absolute branch to a location anywhere within Program Memory.

Operation
PC ⇐ k

Syntax Operands Program Counter
JMP k 0 ≤ k ≤ 32K PC ⇐ k

Instruction Format
1001 0100 0000 1100
0kkk kkkk kkkk kkkk

Status Register Usage
I T H S V N Z C
- - - - - - - -

This instruction does not modify any SREG bits.

Example
mov r1,r0 ; copy r0 into r1
jmp farplc ; unconditional branch
...
farplc: nop ; branch destination

Space/Time
This instruction is 2 instruction words (4 bytes) wide and takes 3 machine cycles to complete.
**LDI: Load Immediate**

**Description**
This instruction loads an 8-bit constant into register $Rd$.

**Operation**
$Rd \leftarrow K$

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Operands</th>
<th>Program Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDI Rd,K</td>
<td>$16 \leq d \leq 31$, $0 \leq K \leq 255$</td>
<td>$PC \leftarrow PC + 1$</td>
</tr>
</tbody>
</table>

**Instruction Format**

```
1110 KKKK dddd KKKK
```

**Status Register Usage**

```
I T H S V N Z C
- - - - - - - -
```

This instruction does not modify any SREG bits.

**Example**

```
ldi r31,$1f ; load 31 into r31
ldi r30,30 ; load 30 into r30
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
LDS: Load Direct from Data Space

Description
This instruction loads one byte from data space into register \( \text{Rd} \). The data space consists of the register file, I/O memory, and internal SRAM.

Operation
\( \text{Rd} \leftarrow (k) \)

Syntax
\[
\text{LDS \ Rd, k}
\]
Operands
\( 0 \leq d \leq 31, \ 0 \leq k \leq 65535 \)
Program Counter
\( \text{PC} \leftarrow \text{PC} + 2 \)

Instruction Format
1001 000d dddd 0000
\( kkkk \ kkkk \ kkkk \ kkkk \)

Status Register Usage
\[
\begin{array}{cccccccc}
I & T & H & S & V & N & Z & C \\
\hline
\end{array}
\]
This instruction does not modify any SREG bits.

Example
\[
\begin{align*}
\text{lds r22,$ff00} &; \text{load r22 with contents of data space location $ff00} \\
\text{andi r22,$fe} &; \text{clear bit 0 in r22} \\
\text{sts $ff00,r22} &; \text{write modified data back to where it came from}
\end{align*}
\]

Space/Time
This instruction is 2 instruction words (4 bytes) wide and takes 2 machine cycles to complete.
**LSR:** Logical Shift Right

**Description**
Shifts all bits in Rd one place to the right. Bit 7 of Rd is cleared. Bit 0 of Rd is loaded into the C flag of the SREG. This operation effectively divides an unsigned value by two. The C flag can be used to round the result.

**Operation**

**Syntax**
LSR Rd

**Operands**
0 ≤ d ≤ 31

**Program Counter**
PC ← PC + 1

**Instruction Format**
1001 010d dddd 0110

**Status Register Usage**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>V</td>
<td>N</td>
<td>Z</td>
<td>C</td>
</tr>
</tbody>
</table>

S ← N ⊕ V, for signed tests

V ← N ⊕ C (for N and C after the shift)

N ← 0
Set if MSB of the result is set.

Z ← R7 • R6 • R5 • R4 • R3 • R2 • R1 • R0
Set if the result of the operation was 0.

C ← Rd0
Set if the LSB of Rd was set before the shift.

**Example**
```assembly
    lsr r8 ; shift r8 right, putting bit 0 into the C flag
    brbs 0,BitWasOne ; if bit 0 was a 1, branch to BitWasOne
    ...;
    BitWasOne: nop ; branch destination
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**MOV: Move Value From Register**

**Description**
This instruction moves the value in \( Rr \) into \( Rd \). The value in \( Rr \) remains unchanged, while the destination register \( Rd \) is loaded with a copy of \( Rr \).

**Operation**
\( Rd \leftarrow Rr \)

**Syntax**
\[ \text{MOV } Rd, Rr \]

**Operands**
\( 0 \leq d \leq 31, \ 0 \leq r \leq 31 \)

**Program Counter**
\( PC \leftarrow PC + 1 \)

**Instruction Format**

\[ \begin{array}{lll}
\text{0010 11rd dddd rrrr} \\
\end{array} \]

**Status Register Usage**

\[ \begin{array}{cccccccccc}
I & T & H & S & V & N & Z & C \\
\end{array} \]

This instruction does not modify any SREG bits.

**Example**
\begin{align*}
\text{mov r16,r0 ; copy r0 into r16} \\
\text{call check ; call subroutine} \\
\text{...} \\
\text{check:} \\
\text{cpi r16,$11 ; compare r16 to 17} \\
\text{...} \\
\text{ret ; return from subroutine}
\end{align*}

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
NEG: Two’s Complement

Description
This instruction replaces the contents of register Rd with its two’s complement; the value $80$ is left unchanged.

Operation
\( \text{Rd} \leftarrow \$00 - \text{Rd} \)

Syntax | Operands | Program Counter
---|---|---
NEG Rd | 0 ≤ d ≤ 31 | PC ← PC + 1

Instruction Format
1001 010d dddd 0001

Status Register Usage

\( H \leftarrow R3 + Rd3 \)

Set if there was a borrow from bit 3.

\( S \leftarrow N \oplus V \), for signed tests

\( V \leftarrow R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)

Set if there is a two’s complement overflow from the implied subtraction from $00$. A two’s complement overflow will occur if and only if the result is $80$.

\( N \leftarrow R7 \)

Set if MSB of the result is set.

\( Z \leftarrow R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)

Set if the result of the operation was 0.

\( C \leftarrow R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0 \)

Set if there is a borrow in the implied subtraction from $00$. The C flag will always be set unless the result is $00$.

Example
neg r19

Space/Time
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**NOP**: No Operation

**Description**
This instruction performs a single-cycle No Operation.

**Operation**
none

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Operands</th>
<th>Program Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>none</td>
<td>PC ← PC + 1</td>
</tr>
</tbody>
</table>

**Instruction Format**
0000 0000 0000 0000

**Status Register Usage**
I T H S V N Z C
- - - - - - - -

This instruction does not modify any SREG bits.

**Example**
```plaintext```
call DelaySevenCycles

```
DelaySevenCycles:
    nop
    nop
    nop
    ret ; return takes 4 cycles
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**OR**: Logical OR

**Description**
Computes the bitwise logical OR of the contents of registers $R_d$ and $R_r$ and places the result in $R_d$.

**Operation**

$$R_d \leftarrow R_d \lor R_r$$

**Syntax**

```
OR Rd, Rr
```

**Operands**

| $0 \leq d \leq 31$, $0 \leq r \leq 31$ |

**Program Counter**

$$PC \leftarrow PC + 1$$

**Instruction Format**

```
0010 10rd dddd rrrr
```

**Status Register Usage**

$I$ $T$ $H$ $S$ $V$ $N$ $Z$ $C$

- - - ? 0 ? ? -

$S \leftarrow N \oplus V$, for signed tests

$V \leftarrow 0$

$V$ is always cleared by this instruction.

$N \leftarrow R_7$

Set if MSB of the result is set.

$Z \leftarrow R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0$

Set if the result of the operation was 0.

**Example**

```
or r16, r1 ; perform OR of r16 with r1
```

**Space/Time**

This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**ORI**: Logical OR with Immediate

**Description**
Computes the bitwise logical OR of the contents of register Rd and a constant and places the result in Rd.

**Operation**
\[ \text{Rd} \leftarrow \text{Rd} \lor K \]

**Syntax**  
ORI Rd, K

**Operands**
\[ 16 \leq d \leq 31, \ 0 \leq K \leq 255 \]

**Program Counter**
\[ \text{PC} \leftarrow \text{PC} + 1 \]

**Instruction Format**
\[ 0110 \ KKKK \ dddd \ KKKK \]

**Status Register Usage**
\[ I \ T \ H \ S \ V \ N \ Z \ C \]
\[- \ - \ - \ ? \ 0 \ ? \ ? \ - \]

- \[ S \leftarrow N \oplus V \], for signed tests
- \[ V \leftarrow 0 \]
  \[ V \text{ is always cleared by this instruction.} \]
- \[ N \leftarrow R7 \]
  \[ \text{Set if MSB of the result is set.} \]
- \[ Z \leftarrow R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \]
  \[ \text{Set if the result of the operation was 0.} \]

**Example**
- \[ \text{ori} \ r16, \$F0 \ ; \text{set high nibble of r16} \]
- \[ \text{ori} \ r17, 1 \ ; \text{set bit 0 of r17} \]

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**OUT**: Store Register Value to I/O Location

**Description**
This instruction stores the value of register $Rr$ into an address in I/O space.

**Operation**
$I/O(A) \leftarrow Rr$

**Syntax**  
`OUT A,Rr`

**Operands**  
$0 \leq r \leq 31$, $0 \leq A \leq 63$

**Program Counter**  
$PC \leftarrow PC + 1$

**Instruction Format**

```
1011 1AAr rrrr AAAA
```

**Status Register Usage**

\[
\begin{array}{cccccccc}
I & T & H & S & V & N & Z & C \\
- & - & - & - & - & - & - & -
\end{array}
\]

This instruction does not modify any SREG bits.

**Example**

```
ldi r16,$08 ; load 0000 1000 into r16
out $05,r16 ; set PORTB pin 3 (PORTB[3])
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**POP**: Pop Value from Stack into Register

**Description**
This instruction loads register \( R_d \) with a value from the stack. The stack pointer (SP) is incremented by 1 *before* the pop. The stack is not scrubbed as a result of this operation.

**Operation**
\[ R_d \leftarrow \text{STACK}(SP) \]

**Syntax**
\[
\text{POP} \ R_d \quad 0 \leq d \leq 31
\]

**Operands**
0 \leq d \leq 31

**Program Counter**
\[ \text{PC} \leftarrow \text{PC} + 1 \]

**Stack**
\[ \text{SP} \leftarrow \text{SP} + 1 \]

**Instruction Format**
1001 000d dddd 1111

**Status Register Usage**
I T H S V N Z C
--- --- --- --- --- ---

This instruction does not modify any SREG bits.

**Example**
```
call MyFunction
...
MyFunction:
push r31 ; save r31
push r30 ; save r30

... ; more work here...

pop r30 ; restore r30
pop r31 ; restore r31
ret
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**PUSH**: Push Register Value onto Stack

**Description**
This instruction stores the value of register \( R_r \) on the stack. The stack pointer (SP) is decremented by 1 after the push. The value in register \( R_r \) is not affected by this instruction.

**Operation**
\[ \text{STACK}(\text{SP}) \leftarrow R_r \]

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Operands</th>
<th>Program Counter</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH ( R_r )</td>
<td>0 \leq r \leq 31</td>
<td>PC \leftarrow PC + 1</td>
<td>SP \leftarrow SP - 1</td>
</tr>
</tbody>
</table>

**Instruction Format**
1001 001r rrrr 1111

**Status Register Usage**
- - - - - - - -

This instruction does not modify any SREG bits.

**Example**
```assembly
call MyFunction
...
MyFunction:
push r31 ; save r31
push r30 ; save r30

... ; more work here...

pop r30 ; restore r30
pop r31 ; restore r31
ret
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 1 machine cycle to complete.
**RCALL**: Relative Call to Subroutine

**Description**
This instruction makes a relative call to an address within PC - 2K + 1 and PC + 2K instruction words. The address of the instruction after the RCALL is stored onto the stack as the return address. The stack pointer (SP) is decremented by two bytes (one instruction word) after the return address is stored.

**Operation**
\[ \text{PC} \leftarrow \text{PC} + k \]

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Operands</th>
<th>Program Counter</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCALL k</td>
<td>-2K \leq k \leq K</td>
<td>PC \leftarrow \text{PC} + 1</td>
<td>STACK \leftarrow \text{PC}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SP \leftarrow SP - 2</td>
<td></td>
</tr>
</tbody>
</table>

**Instruction Format**
1101 kkkk kkkk kkkk

**Status Register Usage**
I T H S V N Z C
- - - - - - - -

This instruction does not modify any SREG bits.

**Example**
```
rcall MyFunction
...
MyFunction:
push r31 ; save r31
push r30 ; save r30

... ; more work here...

pop r30 ; restore r30
pop r31 ; restore r31
ret
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 3 machine cycles to complete.
RET: Return from Subroutine

Description
This instruction actualizes a return from a subroutine. The return address is loaded from the stack. The stack pointer (SP) is incremented by two bytes (one instruction word) before the return address is retrieved from the stack.

Operation
PC ← STACK(SP)

Syntax  Operands  Stack
RET  This instruction has no operands  SP ← SP + 2

Instruction Format
1001 0101 0000 1000

Status Register Usage
I  T  H  S  V  N  Z  C
- - - - - - - -

This instruction does not modify any SREG bits.

Example
rcall MyFunction
...
MyFunction:
push r31 ; save r31
push r30 ; save r30
...
; more work here...
pop r30 ; restore r30
pop r31 ; restore r31
ret

Space/Time
This instruction is 1 instruction word (2 bytes) wide and takes 4 machine cycles to complete.
RETI: Return from Interrupt

Description
This instruction actualizes a return from an interrupt. The return address is loaded
from the stack. The stack pointer (SP) is incremented by two bytes (one instruction
word) before the return address is retrieved from the stack. This instruction sets the
Global Interrupt flag in SREG to permit further interrupts.

Operation
PC ← STACK(SP)

Syntax Operands Stack
RETI This instruction has no operands SP ← SP + 2

Instruction Format
1001 0101 0001 1000

Status Register Usage
I T H S V N Z C
1 - - - - - - -

I ← 1
I is always set by this instruction.

Example
ISR_0:
    push r0 ; save r0
    ... ; more work here...
    pop r0 ; restore r0
    reti

Space/Time
This instruction is 1 instruction word (2 bytes) wide and takes 4 machine cycles to
complete.
**RJMP:** Relative Jump

**Description**
This instruction makes an unconditional absolute branch to a location within PC - 2K + 1 and PC + 2K instruction words in Program Memory.

**Operation**
PC ← PC + k

**Syntax**
RJMP k

**Operands**
-2K ≤ k ≤ 2K

**Program Counter**
PC ← PC + k

**Instruction Format**
1100 kkkk kkkk kkkk

**Status Register Usage**
I T H S V N Z C
- - - - - - - -

This instruction does not modify any SREG bits.

**Example**
```
cpi r16,$42  ; compare r16 to 66
brbs 1,error ; if not equal, error
rjmp ok
```
```
error:
   add r16,r17
   inc r16
ok:  nop
```

**Space/Time**
This instruction is 1 instruction word (2 bytes) wide and takes 2 machine cycles to complete.
**STS: Store Direct to Data Space**

**Description**
This instruction stores one byte from register \( Rr \) into data space. The data space consists of the register file, I/O memory, and internal SRAM.

**Operation**
\[(k) \leftarrow Rr\]

**Syntax**
STS \( k,Rr \)

**Operands**
\(0 \leq r \leq 31, \ 0 \leq k \leq 65535\)

**Program Counter**
\(PC \leftarrow PC + 2\)

**Instruction Format**
1001 001r rrrr 0000

**Status Register Usage**

\[\begin{array}{cccccc}
I & T & H & S & V & N & Z & C \\
- & - & - & - & - & - & - & -
\end{array}\]

This instruction does not modify any SREG bits.

**Example**
- `lds r22,$ff00 ; load r22 with contents of data space location $ff00`
- `andi r22,$fe ; clear bit 0 in r22`
- `sts $ff00,r22 ; write modified data back to where it came from`

**Space/Time**
This instruction is 2 instruction words (4 bytes) wide and takes 2 machine cycles to complete.
Chapter 3

ATmega328 Ports

The AVR family of processors contains a set of 8-bit ports. These ports are essentially pins that are tied to 8-bit registers, and are used to communicate with the world outside the processor. Ports on AVR processors are reconfigurable and are mapped into data memory. Ports must be configured before they are used. Figure 3.1 shows the ports on the ATmega328 and the addresses to which they are mapped.

<table>
<thead>
<tr>
<th>I/O Space Address</th>
<th>Data Space Address</th>
<th>Register Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0B</td>
<td>0x2B</td>
<td>PORTD</td>
</tr>
<tr>
<td>0x0A</td>
<td>0x2A</td>
<td>DDRD</td>
</tr>
<tr>
<td>0x09</td>
<td>0x29</td>
<td>PIND</td>
</tr>
<tr>
<td>0x08</td>
<td>0x28</td>
<td>PORTC</td>
</tr>
<tr>
<td>0x07</td>
<td>0x27</td>
<td>DDRC</td>
</tr>
<tr>
<td>0x06</td>
<td>0x26</td>
<td>PINC</td>
</tr>
<tr>
<td>0x05</td>
<td>0x25</td>
<td>PORTB</td>
</tr>
<tr>
<td>0x04</td>
<td>0x24</td>
<td>DDRB</td>
</tr>
<tr>
<td>0x03</td>
<td>0x23</td>
<td>PINB</td>
</tr>
</tbody>
</table>

Figure 3.1: Memory-Mapped ATmega328 Ports

Individual pins associated with a given port can be configured to be either an input or an output at any given time. The role of an individual pin (or all of the pins) associated with a given port can be changed at runtime. Figure 3.2 shows assembly language code for configuring pin 2 of PORTD (or PORTD[2]) as an input. This sample code configures the data direction of PORTD[2] without changing the data direction associated with any of the other pins. This sample uses data space
instructions (lds and sts), and therefore must use the “Data Space Address” of DDRD as depicted in Figure 3.1 in order to configure the data direction of PORTD pin 2.

.lkü _ddrd = $2a

lds r24,_ddrd ; load r24 with DDRD
andi r24,$fb ; clear Pin 2 (to make it an input)
sts _ddrd,r24 ; write back to DDRD

Figure 3.2: Configuring PORTD Pin 2 as Input