AVR INSTRUCTION SET

Contents

List of Figures

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

15–12	11–10	9–8	7–4	3–0
op1	op2	rd	dddd	rrrr

op1	op2	
0000	11	ADD
	00	NOP
0001	01	CP
	11	ADC
0010	00	AND
	01	EOR (CLR)
	10	OR
	11	MOV

Immediate Instructions

15–12	11–8	7–4	3–0
op1	KKKK	dddd	KKKK

op1	
0011	CPI
0110	ORI
0111	ANDI
1110	LDI

Unary Logical Instructions

15–12	11–9	8	7–4	3–0
1001	010	d	dddd	op4

op4	
0000	COM
0001	NEG
0101	ASR
0110	LSR

Load/Store Instructions

15–12	11–9	8	7–4	3–0
1001	op3	d	dddd	0000

op3	
000	LDS
001	STS

Second word:

15–0
KKKK KKKK KKKK KKKK

Branch Instructions

15–12	11–10	9–8	7–4	3	2–0
1111	0x	kk	kkkk	k	sss

sss: cc

x	
0	BRBS
1	BRBC

Input/Output Instructions

15–12	11	10–9	8	7–4	3–0
1011	x	AA	r	rrrr	AAAA

x	
0	IN
1	OUT

Call/Jump Instructions

15–12	11–8	7–4	3–0
1001	0100	0000	op4

op4	
1110	CALL
1100	JMP

Second word:

15–0
KKKK KKKK KKKK KKKK

Return Instructions

15–12	11–8	7–5	4	3–0
1001	0101	000	x	1000

x	
0	RET
1	RETI

Stack Instructions

15–12	11–10	9–8	7–4	3–0
1001	00	xd	dddd	1111

x	
0	POP
1	PUSH

Relative Jump Instructions

15–12	11–8	7–4	3–0
110x	KKKK	KKKK	KKKK

x	
0	RJMP
1	RCALL

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

$\Leftarrow$: 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
$\texttt{Rd} \Leftarrow \texttt{Rd} + \texttt{Rr} + \texttt{C}$

Syntax	Operands	Program Counter
`ADC Rd,Rr`	$0 \le \mathrm{d} \le 31,\ 0 \le \mathrm{r} \le 31$	$\mathrm{PC} \Leftarrow \mathrm{PC} + 1$

Instruction Format
`0001 11rd dddd rrrr`

Status Register Usage

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

H $\Leftarrow Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + Rd3 \bullet \overline{R3}$
Set if there was a carry from bit 3.

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

V $\Leftarrow Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation.

N $\Leftarrow R7$
Set if MSB of the result is set.

Z $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result of the operation was 0.

C $\Leftarrow Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + Rd7 \bullet \overline{R7}$
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
`Rd` $\Leftarrow$ `Rd` + `Rr`

Syntax	**Operands**	**Program Counter**
`ADD Rd,Rr`	$0 \le \mathrm{d} \le 31$, $0 \le \mathrm{r} \le 31$	$\mathrm{PC} \Leftarrow \mathrm{PC} + 1$

Instruction Format
`0000 11rd dddd rrrr`

Status Register Usage

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

H $\Leftarrow Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + Rd3 \bullet \overline{R3}$
Set if there was a carry from bit 3.

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

V $\Leftarrow Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation.

N $\Leftarrow R7$
Set if MSB of the result is set.

Z $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result of the operation was 0.

C $\Leftarrow Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + Rd7 \bullet \overline{R7}$
Set if there was a carry from the MSB of the result.

Example

```
add r1,r2   ; add r2 to r1
add r28,r28 ; add r28 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` $\Leftarrow$ `Rd` $\bullet$ `Rr`

Syntax	Operands	Program Counter
`AND Rd,Rr`	$0 \leq d \leq 31$, $0 \leq r \leq 31$	$PC \Leftarrow PC + 1$

Instruction Format
`0010 00rd 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.

$N \Leftarrow R7$
Set if MSB of the result is set.

$Z \Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
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` $\bullet$ `K`

Syntax	**Operands**	**Program Counter**
`ANDI Rd,K`	$16 \leq \mathrm{d} \leq 31$, $0 \leq \mathrm{K} \leq 255$	PC $\Leftarrow$ PC + 1

Instruction Format
`0111 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 is always cleared by this instruction.

N $\Leftarrow R7$

Set if MSB of the result is set.

Z $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result of the operation was 0.

Example

```
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

`C` $\Leftarrow$ `Rd[0]` $\Leftarrow$ `Rd[1]` $\Leftarrow$ `Rd[2]` $\Leftarrow$ `Rd[3]` $\Leftarrow$ `Rd[4]` $\Leftarrow$ `Rd[5]` $\Leftarrow$ `Rd[6]` $\Leftarrow$ `Rd[7]`

Syntax	Operands	Program Counter
`ASR Rd`	$0 \le d \le 31$	PC $\Leftarrow$ PC + 1

Instruction Format

`1001 010d dddd 0101`

Status Register Usage

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

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 $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result of the operation was 0.

C $\Leftarrow Rd0$
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 ($\texttt{PC} - 63 \leq \text{destination} \leq \texttt{PC} + 64$). The parameter `k` is the offset from the `PC` and is represented in two's complement form.

Operation

if $\texttt{SREG}[s] = 0$ then $\texttt{PC} \Leftarrow \texttt{PC} + \texttt{k}$

Syntax	Operands	Program Counter
`BRBC s,k`	$0 \leq s \leq 7$, $-64 \leq k \leq 63$	$PC \Leftarrow PC + 1$

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 ($\texttt{PC} - 63 \leq \text{destination} \leq \texttt{PC} + 64$). The parameter `k` is the offset from the `PC` and is represented in two's complement form.

Operation
if $\texttt{SREG}[s] = 1$ then $\texttt{PC} \Leftarrow \texttt{PC} + \texttt{k}$

Syntax	Operands	Program Counter
`BRBS s,k`	$0 \leq s \leq 7$, $-64 \leq k \leq 63$	$PC \Leftarrow PC + 1$

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`	$0 \le \mathrm{k} \le 32\mathrm{K}$	PC $\Leftarrow$ k	STACK $\Leftarrow$ PC SP $\Leftarrow$ SP $-$ 2

Instruction Format

```
1001 010k kkkk 111k
kkkk kkkk kkkk kkkk
```

Status Register Usage

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

This instruction does not modify any `SREG` bits.

Example

```
    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	Operands	Program Counter
`COM Rd`	$0 \leq d \leq 31$	$PC \Leftarrow PC + 1$

Instruction Format
`1001 010d dddd 0000`

Status Register Usage

```
I T H S V N Z C
- - - ? 0 ? ? 1
```

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 $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

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 `Rd` and `Rr`. The values in `Rd` and `Rr` are not modified by this instruction.

Operation

`ALU_Result` $\Leftarrow$ `Rd` $-$ `Rr`

Syntax	Operands	Program Counter
`CP Rd,Rr`	$0 \leq d \leq 31$, $0 \leq r \leq 31$	PC $\Leftarrow$ PC + 1

Instruction Format

```
0001 01rd dddd rrrr
```

Status Register Usage

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

H $\Leftarrow \overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$

Set if there was a borrow from bit 3.

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

V $\Leftarrow Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$

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

N $\Leftarrow R7$

Set if MSB of the result is set.

Z $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result of the operation was 0.

C $\Leftarrow \overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$

Set if the absolute value of the contents of `Rr` is greater than the absolute value of the contents of `Rd`.

Example

```
    cp r4,r19        ; compare r4 with r19
    brbc 1,NotEqual  ; branch if r4 != r19
    ...
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 ⇐ Rd − K

Syntax	Operands	Program Counter
CPI Rd,K	$16 \leq d \leq 31$, $0 \leq K \leq 255$	PC ⇐ PC + 1

Instruction Format
`0011 KKKK dddd KKKK`

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

H ⇐ $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3.

S ⇐ $N \oplus V$, for signed tests

V ⇐ $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$
Set if two's complement overflow resulted from the operation.

N ⇐ $R7$
Set if MSB of the result is set.

Z ⇐ $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result of the operation was 0.

C ⇐ $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of K is greater than the absolute value of the contents of Rd.

Example
```
    cpi r19,$CC   ; compare r19 with 0xCC
    brbs 1,Equal ; branch if r19 = 0xCC
    ...
Equal: nop        ; branch destination
```

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

$\texttt{Rd} \Leftarrow \texttt{Rd} \oplus \texttt{Rr}$

Syntax	Operands	Program Counter
`EOR Rd,Rr`	$0 \le d \le 31$, $0 \le r \le 31$	$PC \Leftarrow PC + 1$

Instruction Format

`0010 01rd 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.

$N \Leftarrow R7$

Set if MSB of the result is set.

$Z \Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result of the operation was 0.

Example

```
eor r4,r4    ; clear all bits in r4
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 $\Leftarrow$ k

Syntax	Operands	Program Counter
JMP k	$0 \leq k \leq 32K$	PC $\Leftarrow$ 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	Operands	Program Counter
`LDI Rd,K`	$16 \leq d \leq 31$, $0 \leq K \leq 255$	$PC \Leftarrow PC + 1$

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 `Rd`. The data space consists of the register file, I/O memory, and internal SRAM.

Operation

`Rd` $\Leftarrow$ (`k`)

Syntax	**Operands**	**Program Counter**
`LDS Rd,k`	$0 \leq d \leq 31$, $0 \leq k \leq 65535$	$PC \Leftarrow PC + 2$

Instruction Format

```
1001 000d dddd 0000
kkkk kkkk kkkk kkkk
```

Status Register Usage

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

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.

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

`C` $\Leftarrow$ `Rd[0]` $\Leftarrow$ `Rd[1]` $\Leftarrow$ `Rd[2]` $\Leftarrow$ `Rd[3]` $\Leftarrow$ `Rd[4]` $\Leftarrow$ `Rd[5]` $\Leftarrow$ `Rd[6]` $\Leftarrow$ `Rd[7]` $\Leftarrow 0$

Syntax	Operands	Program Counter
`LSR Rd`	$0 \leq d \leq 31$	$PC \Leftarrow PC + 1$

Instruction Format

`1001 010d dddd 0110`

Status Register Usage

```
I T H S V N Z C
- - - ? ? 0 ? ?
```

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

V $\Leftarrow N \oplus C$ (for N and C *after* the shift)

N $\Leftarrow 0$

Set if MSB of the result is set.

Z $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result of the operation was 0.

C $\Leftarrow Rd0$

Set if the LSB of `Rd` was set before the shift.

Example

```
    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	Operands	Program Counter
`MOV Rd,Rr`	$0 \le \mathrm{d} \le 31$, $0 \le \mathrm{r} \le 31$	$\mathrm{PC} \Leftarrow \mathrm{PC} + 1$

Instruction Format
`0010 11rd dddd rrrr`

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

This instruction does not modify any `SREG` bits.

Example
```
    mov r16,r0 ; copy r0 into r16
    call check ; call subroutine
    ...
check:
    cpi r16,$11 ; compare r16 to 17
    ...
    ret         ; return from subroutine
```

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
`Rd` $\Leftarrow$ \$00 $-$ `Rd`

Syntax	Operands	Program Counter
`NEG Rd`	$0 \leq \mathrm{d} \leq 31$	$\mathrm{PC} \Leftarrow \mathrm{PC} + 1$

Instruction Format
`1001 010d dddd 0001`

Status Register Usage

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

H $\Leftarrow R3 + Rd3$
Set if there was a borrow from bit 3.

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

V $\Leftarrow R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
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 $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
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

Syntax	Operands	Program Counter
`NOP`	none	$PC \Leftarrow PC + 1$

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

```
    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 `Rd` and `Rr` and places the result in `Rd`.

Operation
`Rd` $\Leftarrow$ `Rd` v `Rr`

Syntax	**Operands**	**Program Counter**
`OR Rd,Rr`	$0 \leq d \leq 31,\ 0 \leq r \leq 31$	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 $\Leftarrow R7$
Set if MSB of the result is set.

Z $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
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

`Rd` $\Leftarrow$ `Rd` v `K`

Syntax	Operands	Program Counter
`ORI Rd,K`	$16 \leq d \leq 31$, $0 \leq K \leq 255$	$PC \Leftarrow PC + 1$

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 is always cleared by this instruction.

N $\Leftarrow R7$
Set if MSB of the result is set.

Z $\Leftarrow \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result of the operation was 0.

Example

```
ori  r16,$F0 ; set high nibble of r16
ori  r17,1   ; 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	**Operands**	**Program Counter**
`OUT A,Rr`	$0 \leq r \leq 31$, $0 \leq A \leq 63$	$PC \Leftarrow PC + 1$

Instruction Format
`1011 1AAr rrrr AAAA`

Status Register Usage

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

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 `Rd` 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
`Rd` $\Leftarrow$ STACK(SP)

Syntax	Operands	Program Counter	Stack
`POP Rd`	$0 \leq d \leq 31$	PC $\Leftarrow$ PC + 1	SP $\Leftarrow$ 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 `Rr` on the stack. The stack pointer (`SP`) is decremented by 1 *after* the push. The value in register `Rr` is not affected by this instruction.

Operation
STACK(SP) $\Leftarrow$ `Rr`

Syntax	**Operands**	**Program Counter**	**Stack**
`PUSH Rr`	$0 \leq r \leq 31$	PC $\Leftarrow$ PC + 1	SP $\Leftarrow$ SP − 1

Instruction Format
`1001 001r rrrr 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.

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

$\texttt{PC} \Leftarrow \texttt{PC} + \texttt{k}$

Syntax	**Operands**	**Program Counter**	**Stack**
`RCALL k`	-2K $\leq$ k $\leq$ K	PC $\Leftarrow$ PC + 1	STACK $\Leftarrow$ PC
.			SP $\Leftarrow$ SP $-$ 2

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` $\Leftarrow$ STACK(SP)

Syntax	**Operands**	**Stack**
`RET`	This instruction has no operands	SP $\Leftarrow$ 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` $\Leftarrow$ STACK(SP)

Syntax	**Operands**	**Stack**
`RETI`	This instruction has no operands	SP $\Leftarrow$ SP + 2

Instruction Format

`1001 0101 0001 1000`

Status Register Usage

```
I T H S V N Z C
1 - - - - - - -
```

I $\Leftarrow$ 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 $\Leftarrow$ `PC + k`

Syntax	Operands	Program Counter
`RJMP k`	$-2K \leq k \leq 2K$	$PC \Leftarrow PC + k$

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	Operands	Program Counter
`STS k,Rr`	$0 \leq r \leq 31$, $0 \leq k \leq 65535$	PC $\Leftarrow$ PC + 2

Instruction Format

```
1001 001r rrrr 0000
kkkk kkkk kkkk kkkk
```

Status Register Usage

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

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.

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

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.

```
.equ _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