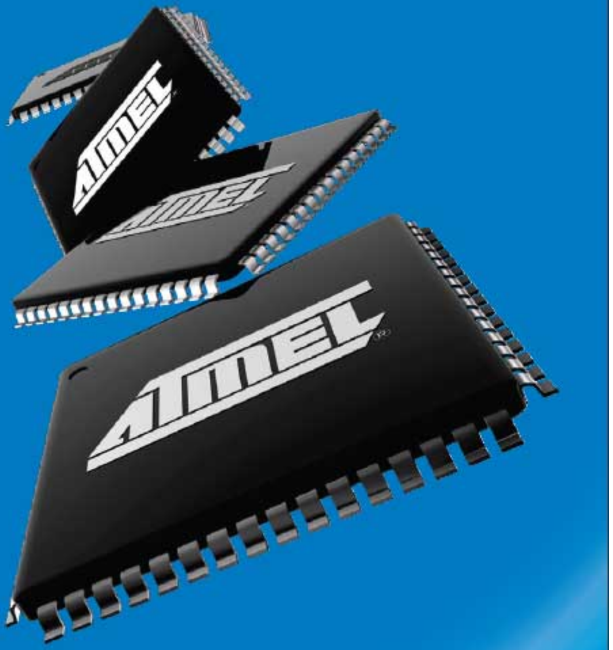
AVR®

8-bit Microcontrollers

AVR®32

32-bit Microcontrollers and Application Processors



Instruction Encoding

February 2009

ATMEL®

Everywhere You Are®

AVR Instruction Set Encoding

READING

Instruction Set Nomenclature

Status Register (SREG)

SREG:	Status Register
C:	Carry Flag
Z:	Zero Flag
N:	Negative Flag
V:	Two's complement overflow indicator
S:	N ⊕ V, For signed tests
H:	Half Carry Flag
T:	Transfer bit used by BLD and BST instructions
I:	Global Interrupt Enable/Disable Flag

Registers and Operands

Rd:	Destination (and source) register in the Register File
Rr:	Source register in the Register File
R:	Result after instruction is executed
K:	Constant data
k:	Constant address
b:	Bit in the Register File or I/O Register (3-bit)
s:	Bit in the Status Register (3-bit)
X,Y,Z:	Indirect Address Register (X=R27:R26, Y=R29:R28 and Z=R31:R30)
A:	I/O location address
q:	Displacement for direct addressing (6-bit)

8-bit AVR® Instruction Set

1. "AVR Instruction Set" document doc0856 "The Program and Data Addressing Modes."

2. In this lecture I will teach you how to translate your assembly code into machine code and vice-versa. You can learn more about each instruction in the AVR Studio 4 Help documentation (Help - Assembler help) or "AVR Instruction Set" document doc0856.

3. LSL - Logical Shift Left, and LSR - Logical Shift Right in the AVR Studio, AVR Assembler Help documentation or "AVR Instruction Set" document doc0856.

4. Section 6.4 "General Purpose Register File" and Section 7.3 "SRAM Data Memory" except section 7.3.1 in the ATmega328P datasheet

CONTENTS

Instruction Set Mapping

The ***Instruction Set*** of our AVR processor can be functionally divided (or classified) into: Data Transfer Instructions, Arithmetic and Logic Instructions, Bit and Bit-Test Instructions, Control Transfer (Branch) Instructions, and MCU Control Instructions.

While this functional division helps you quickly find the instruction you need when you are writing a program; it does not reflect how the designers of the AVR processor mapped an assembly instruction into a 16-bit machine instruction. For this task a better way to look at the instructions is from the perspective of their addressing mode. We will divide AVR instructions into the following addressing mode types.

Data Addressing Modes

- Direct Register Addressing, Single Register
- Direct Register Addressing, Two 32 General Purpose Registers Rd and Rr
- Direct Register Addressing, Two 16 and 8 General Purpose Registers Rd and Rr
- Direct I/O Addressing (including SREG)
- Direct I/O Addressing, First 32 I/O Registers
- Direct SRAM Data Addressing
- Immediate 8-bit Constant
- Immediate 6-bit and 4-bit Constant
- Indirect SRAM Data Addressing with Pre-decrement and Post-increment
- Indirect Program Memory Addressing (Atmel Program Memory Constant Addressing)

Control Transfer

- Direct
- Relative, Unconditional
- Relative, Conditional
- Indirect

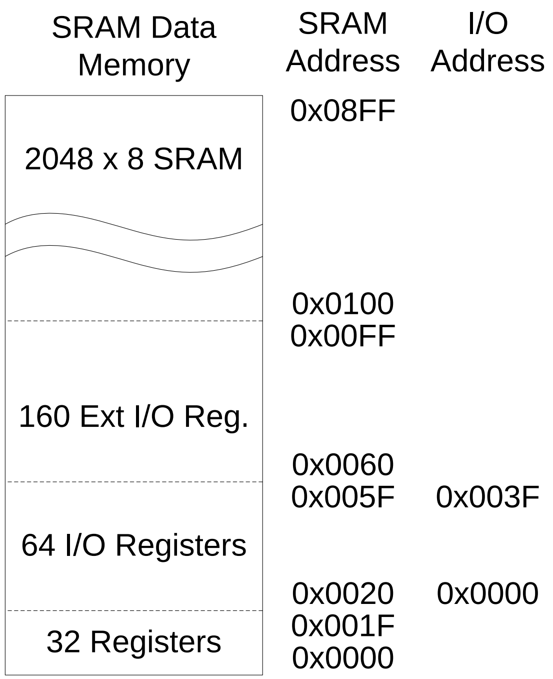
MCU Control Instructions

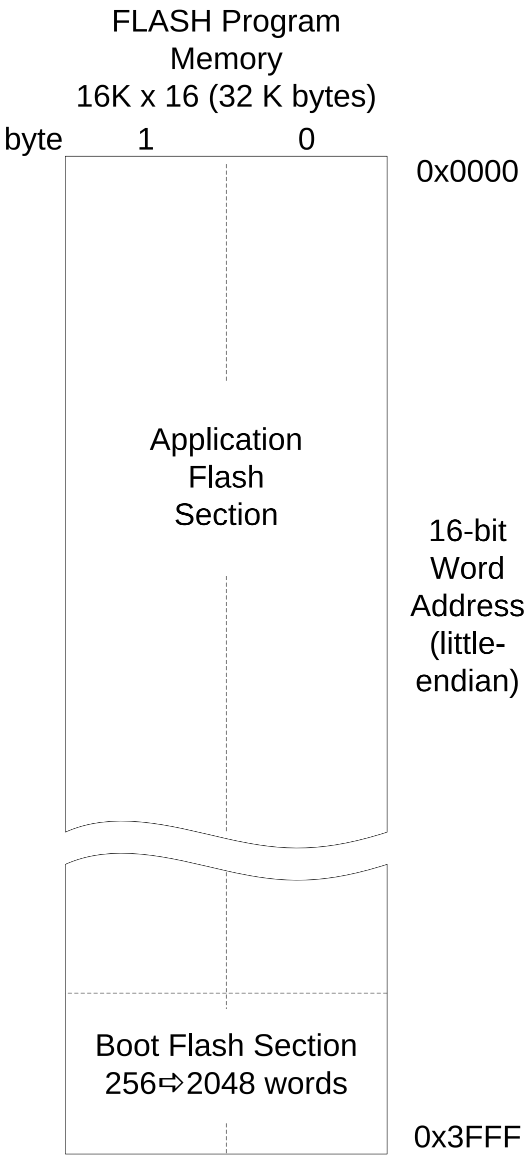
ATmega328P Operand Locations

- When selecting an addressing mode you should ask yourself where the operand is (data) located within the AVR processor.

General Purpose Registers

r1	r0	
	r2	
	...	
	r14	
	r15	
	r16	
	... r25	
r27	r26	X
r29	r28	Y
r31	r30	Z

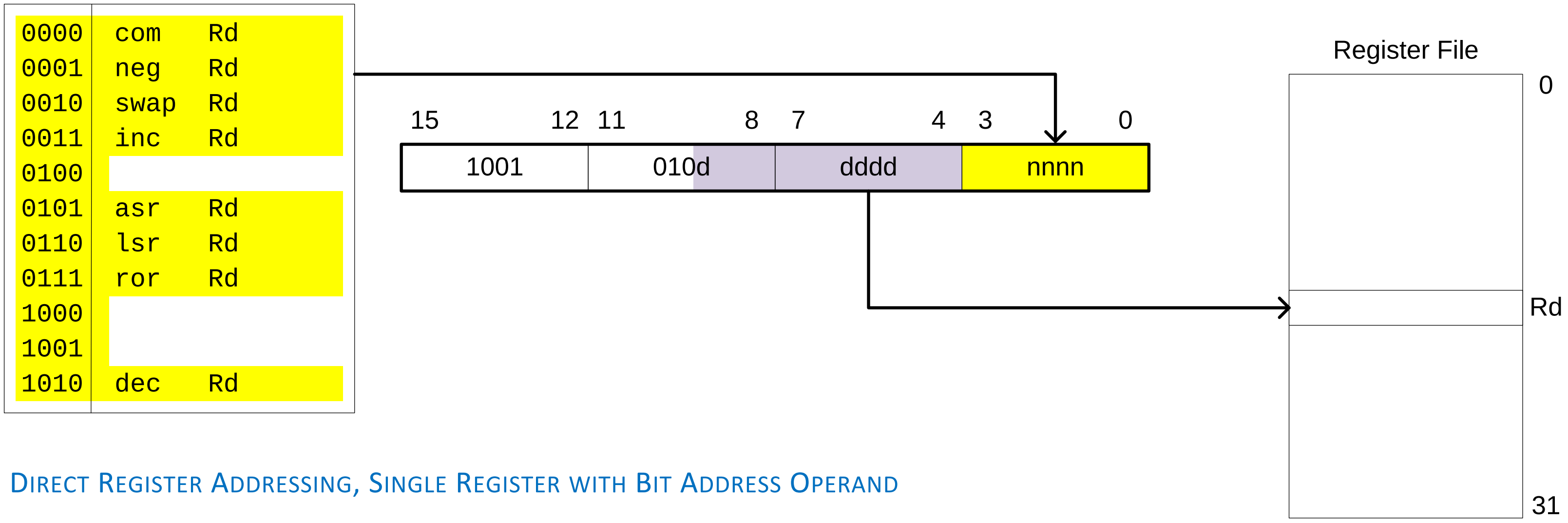




FLASH Program Memory
16K x 16 (32 K bytes)
byte
1
0
0x0000
Application Flash Section
16-bit Word Address (little-endian)
Boot Flash Section 256⇨2048 words
0x3FFF

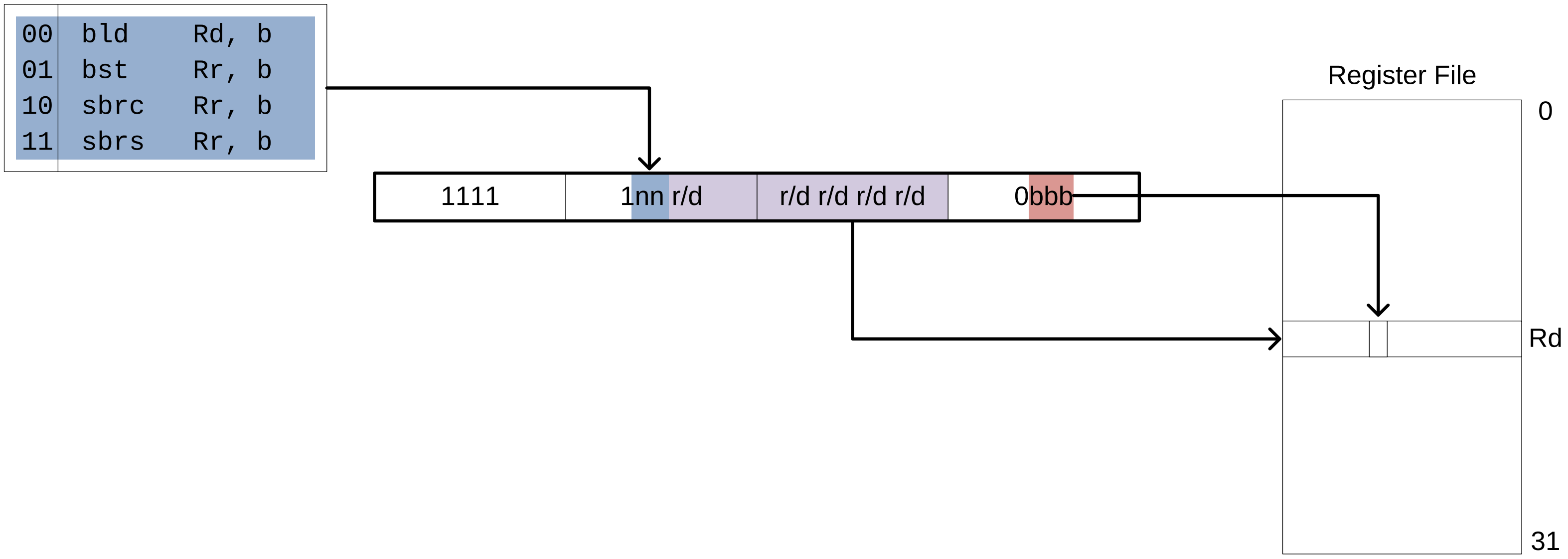
Data Addressing Modes

Direct Register Addressing, Single Register



0000	com	Rd
0001	neg	Rd
0010	swap	Rd
0011	inc	Rd
0100		
0101	asr	Rd
0110	lsr	Rd
0111	ror	Rd
1000		
1001		
1010	dec	Rd

Direct Register Addressing, Single Register with Bit Address Operand

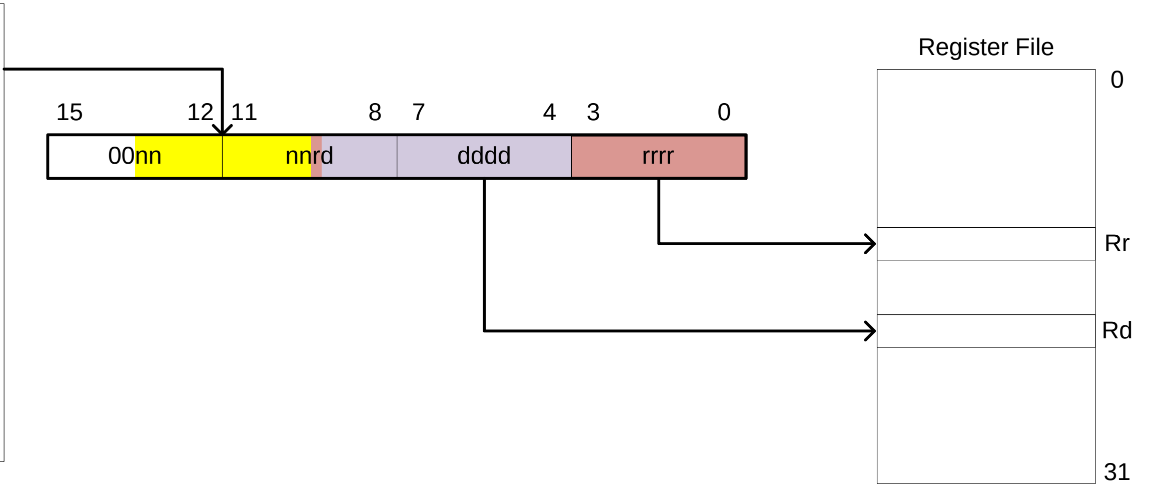


00	bld	Rd, b
01	bst	Rr, b
10	sbrc	Rr, b
11	sbrs	Rr, b

Data Addressing Modes

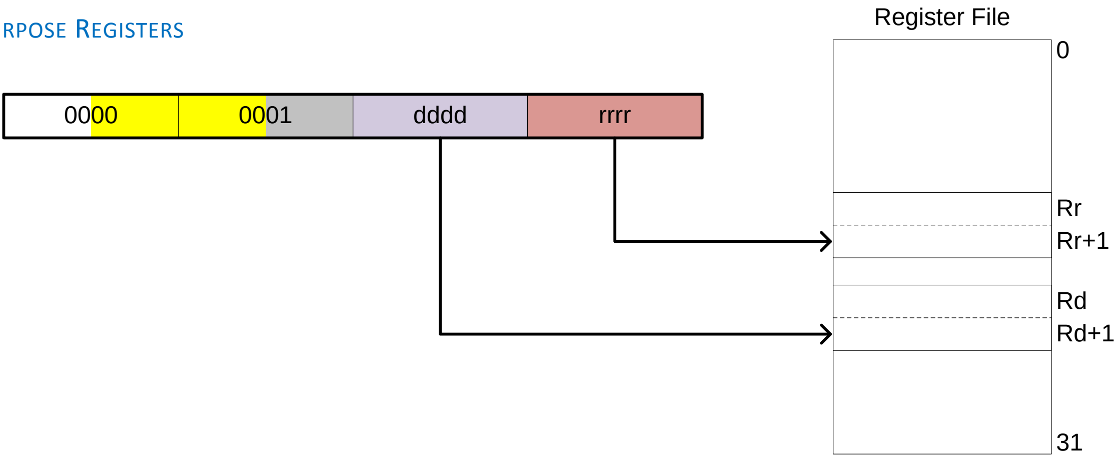
Direct Register Addressing, Two of 32 8-bit General Purpose Registers Rd and Rr

		alias
0000	*16-bit move and floating point*	
0001	cpc Rd, Rr	
0010	sbc Rd, Rr	
0011	add Rd, Rr	lsl Rd (let Rr = Rd)
0100	cpse Rd, Rr	
0101	cp Rd, Rr	
0110	sub Rd, Rr	
0111	adc Rd, Rr	rol Rd (let Rr = Rd)
1000	and Rd, Rr	tst Rd (let Rr = Rd)
1001	eor Rd, Rr	clr Rd (let Rr = Rd)
1010	or Rd, Rr	
1011	mov Rd, Rr	



Direct Register Addressing, Two of 16 16-bit General Purpose Registers

```
movw   Rd, Rr
```

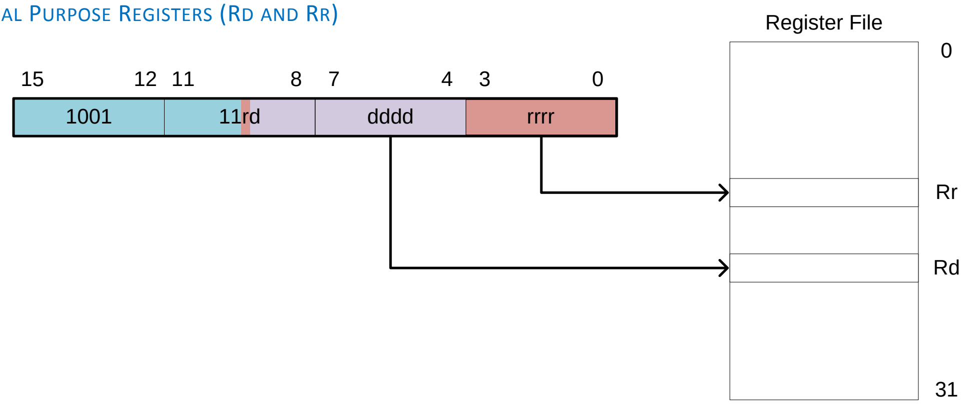


Data Addressing Modes

Multiply

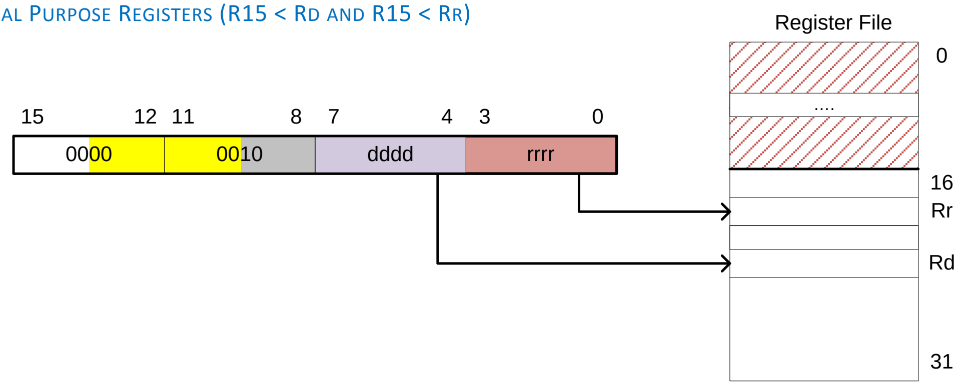
Direct Register Addressing, Two 32 General Purpose Registers (Rd and Rr)

```
mul    Rd, Rr
```



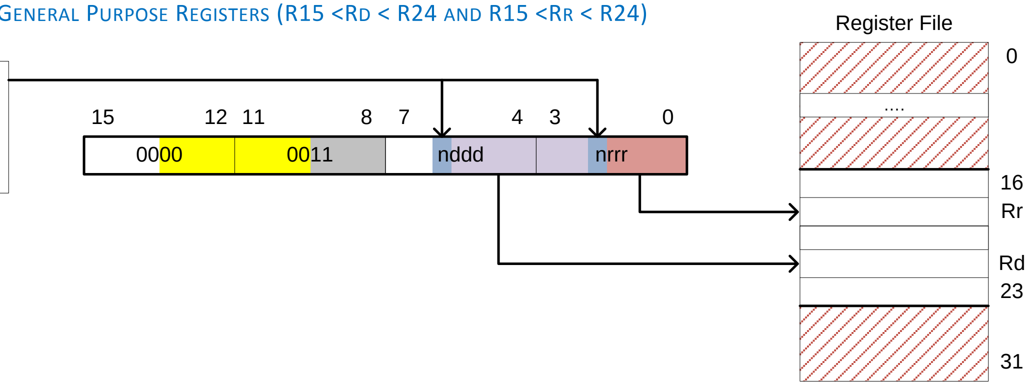
Direct Register Addressing, Two 16 General Purpose Registers (R15 < Rd and R15 < Rr)

```
muls   Rd, Rr
```



Direct Register Addressing, Two 8 General Purpose Registers (R15 <Rd < R24 and R15 <Rr < R24)

```
00   mulsu    Rd, Rr
01   fmul     Rd, Rr
10   fmuls    Rd, Rr
11   fmulsu   Rd, Rr
```

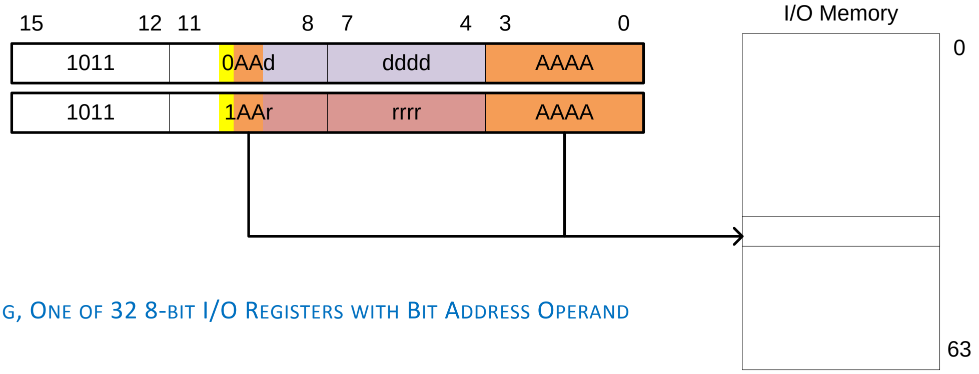


Data Addressing Modes

Direct I/O Addressing (including SREG)

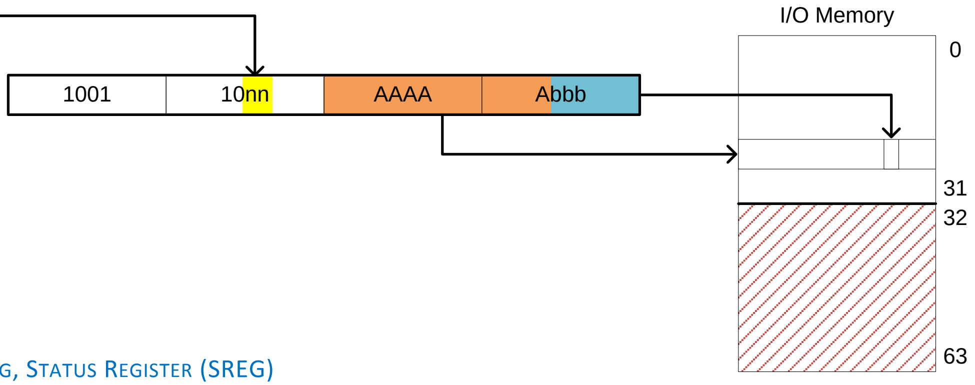
Direct I/O Addressing, One of 64 8-bit I/O Registers

```
in     Rd, A
out    A, Rr
```



Direct I/O Addressing, One of 32 8-bit I/O Registers with Bit Address Operand

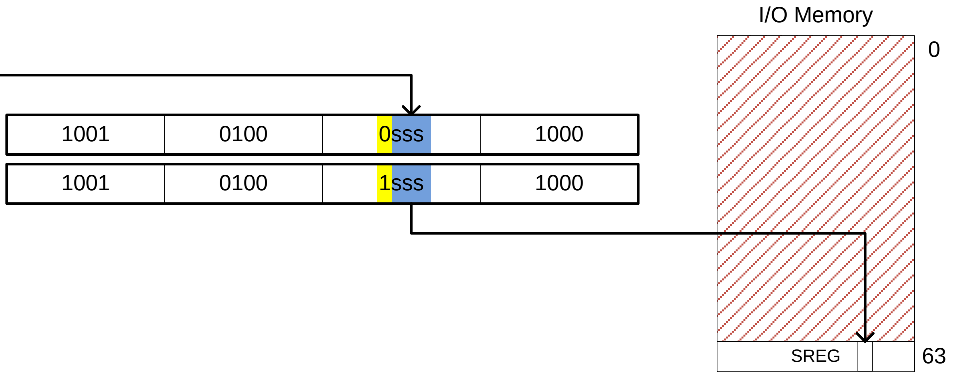
```
00 cbi     A, b
01 sbic    A, b
10 sbi     A, b
11 sbis    A, b
```



Direct I/O Addressing, Status Register (SREG)

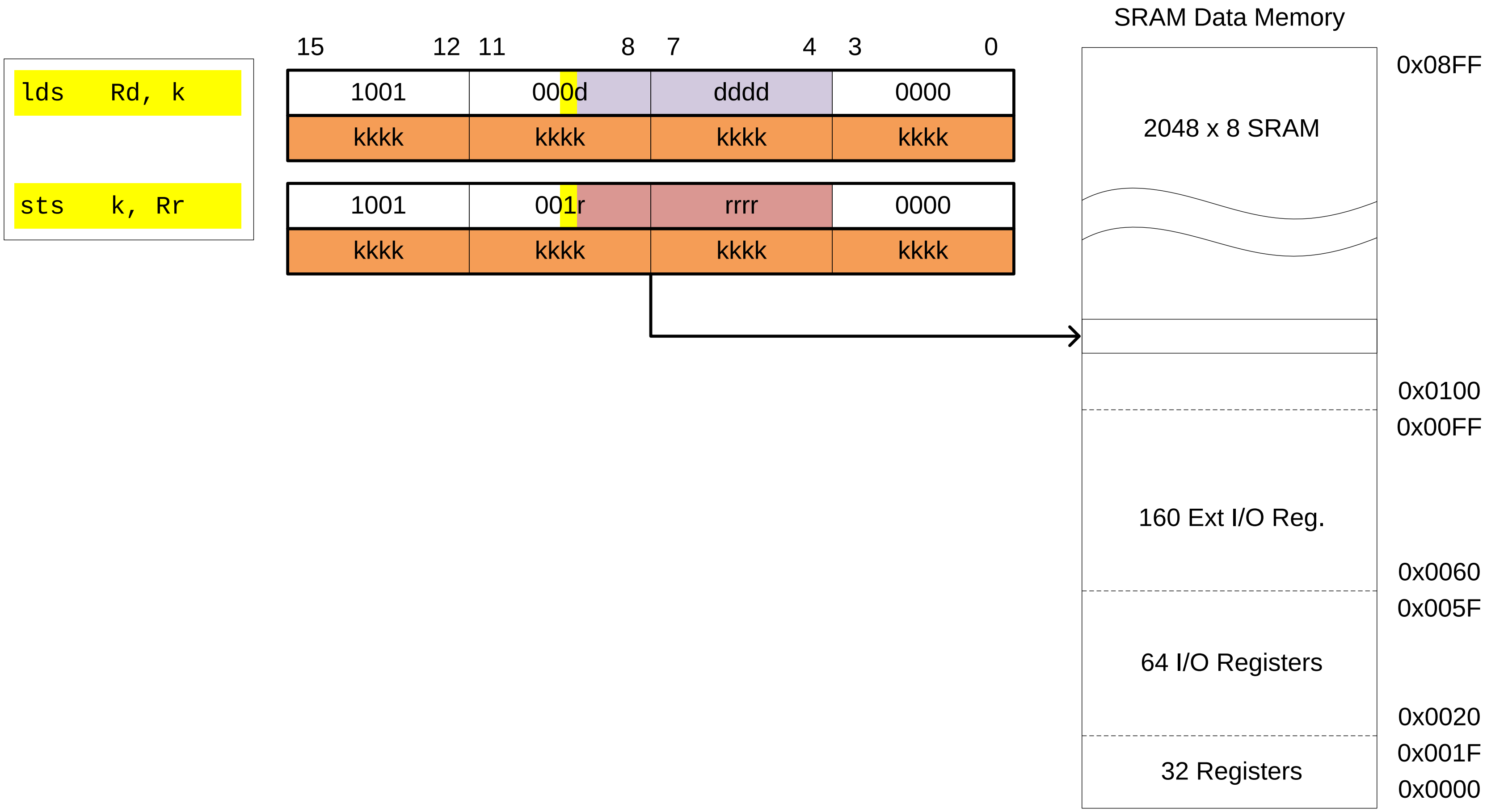
		alias	
SREG	sss	bset s	bclr s
I	111	sei	cli
T	110	set	clt
H	101	seh	clh
S	100	ses	cls
V	011	sev	clv
N	010	sen	cln
Z	001	sez	clz
C	000	sec	clc

```
bset    s
bclr    s
```



Data Addressing Modes

Direct SRAM Data Addressing



```
lds    Rd, k

sts    k, Rr
```

15 12	11 8	7 4	3 0
1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

15 12	11 8	7 4	3 0
1001	001r	rrrr	0000
kkkk	kkkk	kkkk	kkkk

Data Addressing Modes

Immediate

Immediate, 8-bit constant Source Operand with Register Destination Operand (R15 < Rd)

	alias
ldi Rd, K	ser Rd (let K = FF$_{16}$)
cpi Rd, k	

15 12	11 8	7 4	3 0
1110	KKKK	dddd	KKKK
0011	KKKK	dddd	KKKK

		alias
00	sbci Rd, K	
01	subi Rd, K	
10	ori Rd, K	sbr Rd, K
11	andi Rd, K	cbr Rd, K (K = FF – K)

01nn	KKKK	dddd	KKKK

Immediate, 6-bit Unsigned Constant (0 – 63) Source Operand with One of Four (4) 16-bit Registers (r25:r24, X, Y, Z)

adiw Rd, K
sbiw Rd, K

1001	0110	KKdd	KKKK
1001	0111	KKdd	KKKK

Immediate, 4-bit Constant Source Operand

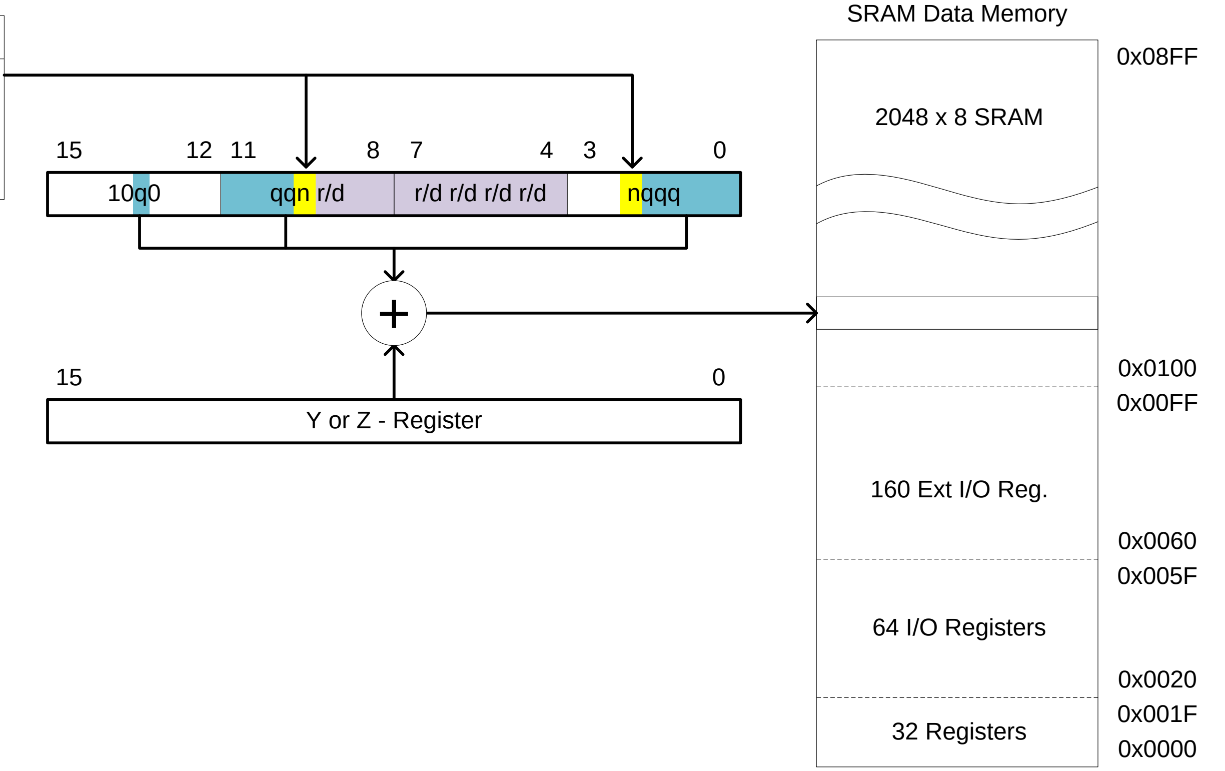
```
des  K
```

1001	0100	KKKK	1011

Data Addressing Modes

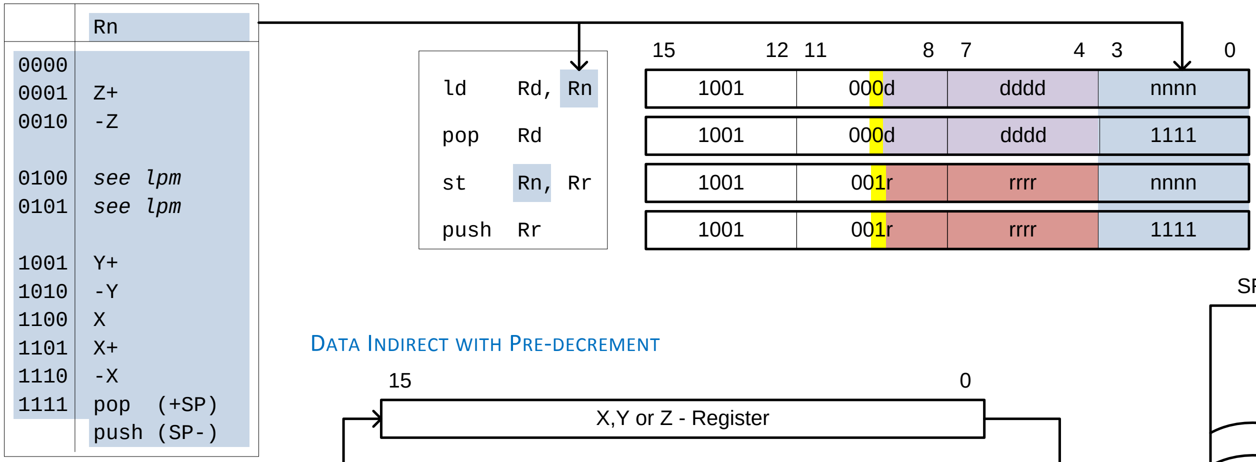
Indirect SRAM Data with Displacement

			alias (q = 0)	
00	ldd	Rd, Z+q	ld	Rd, Z
01	ldd	Rd, Y+q	ld	Rd, Y
10	std	Z+q, Rr	st	Z, Rr
11	std	Y+q, Rr	st	Y, Rr



Data Addressing Modes

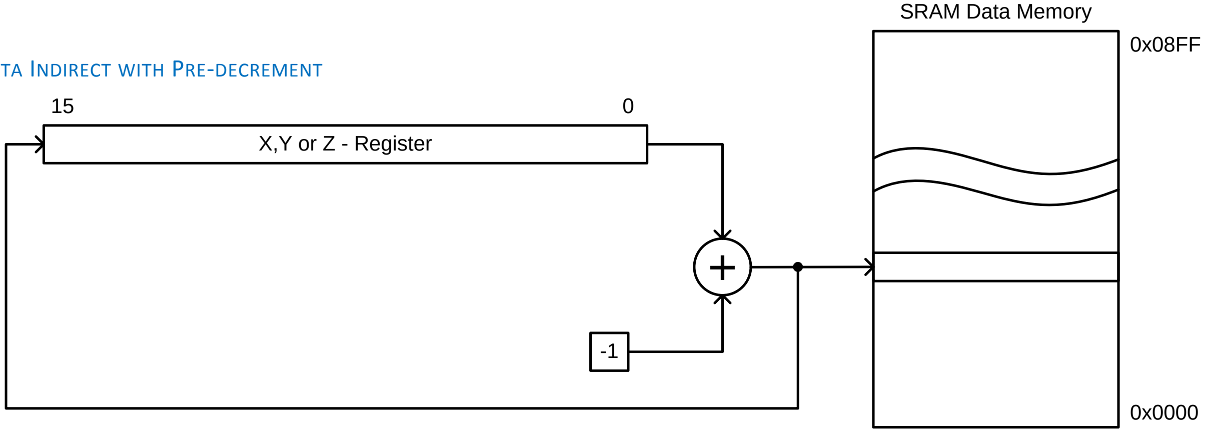
Indirect SRAM Data Addressing with Pre-decrement and Post-increment



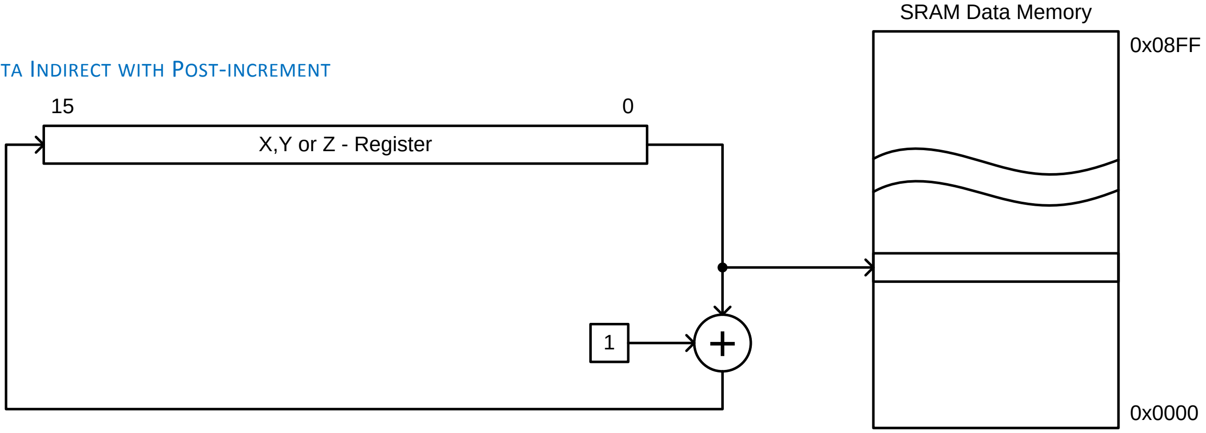
	Rn
0000	
0001	Z+
0010	-Z
0100	*see lpm*
0101	*see lpm*
1001	Y+
1010	-Y
1100	X
1101	X+
1110	-X
1111	pop (+SP) push (SP-)

	15 12	11 8	7 4	3 0
ld Rd, Rn	1001	000d	dddd	nnnn
pop Rd	1001	000d	dddd	1111
st Rn, Rr	1001	001r	rrrr	nnnn
push Rr	1001	001r	rrrr	1111

Data Indirect with Pre-decrement



Data Indirect with Post-increment



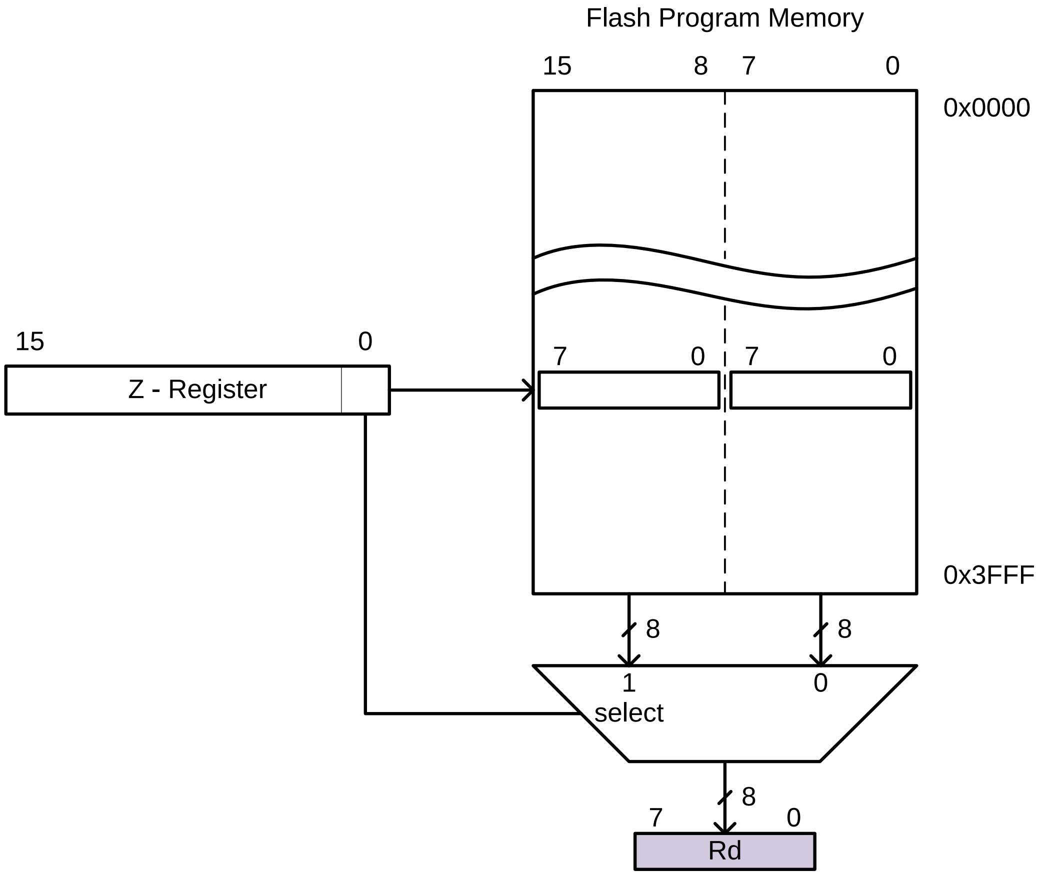
Data Addressing Modes

Indirect Program Memory Addressing (Atmel Program Memory Constant Addressing)

```
lpm    Rd, Z     see illustration

lpm    Rd, Z+

lpm
```

15 12	11 8	7 4	3 0
1001	000d	dddd	0100
1001	000d	dddd	0101
1000	0101	1100	1000

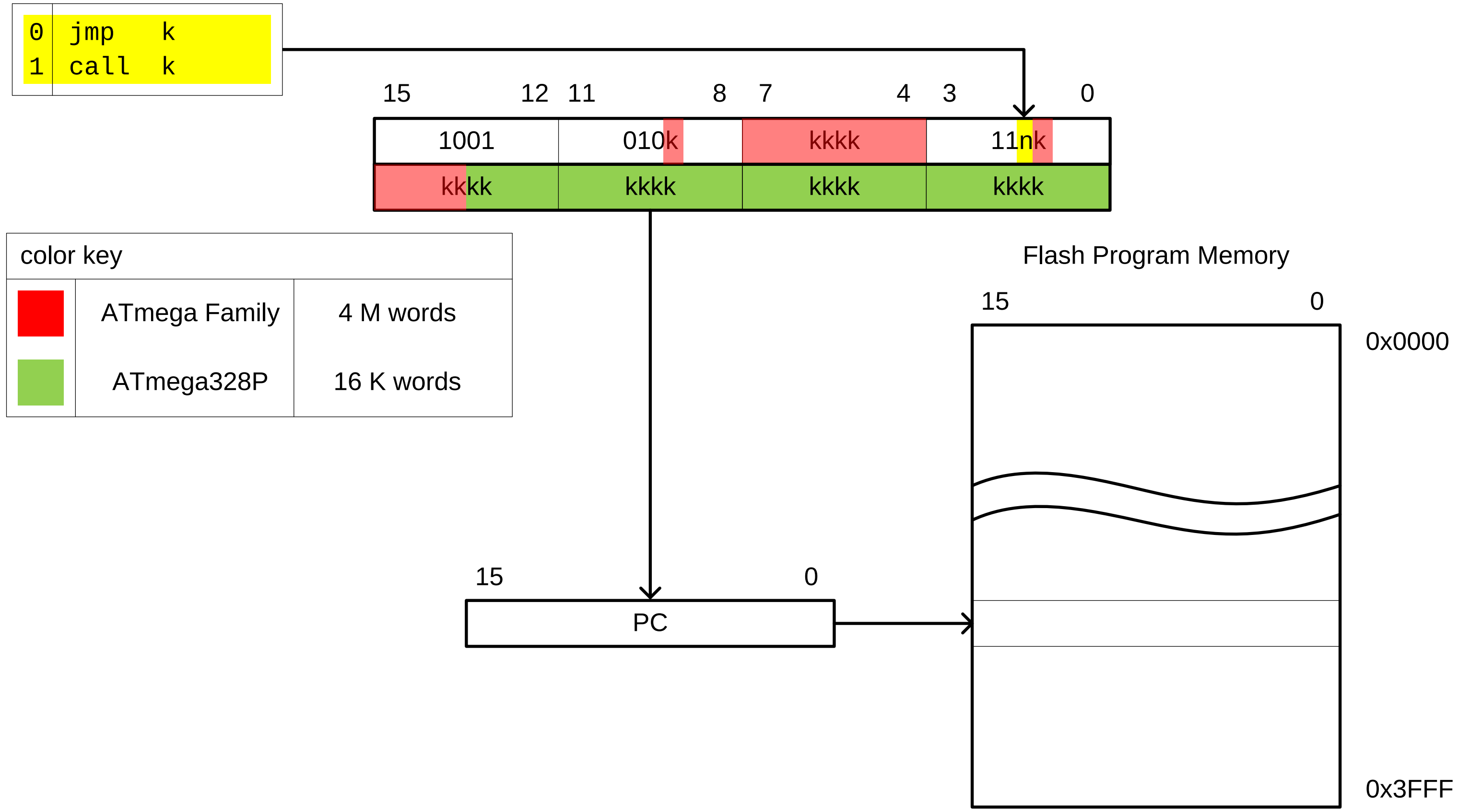
note: R0 implied



Control Transfer

Direct

- All control transfer addressing modes modify the program counter.

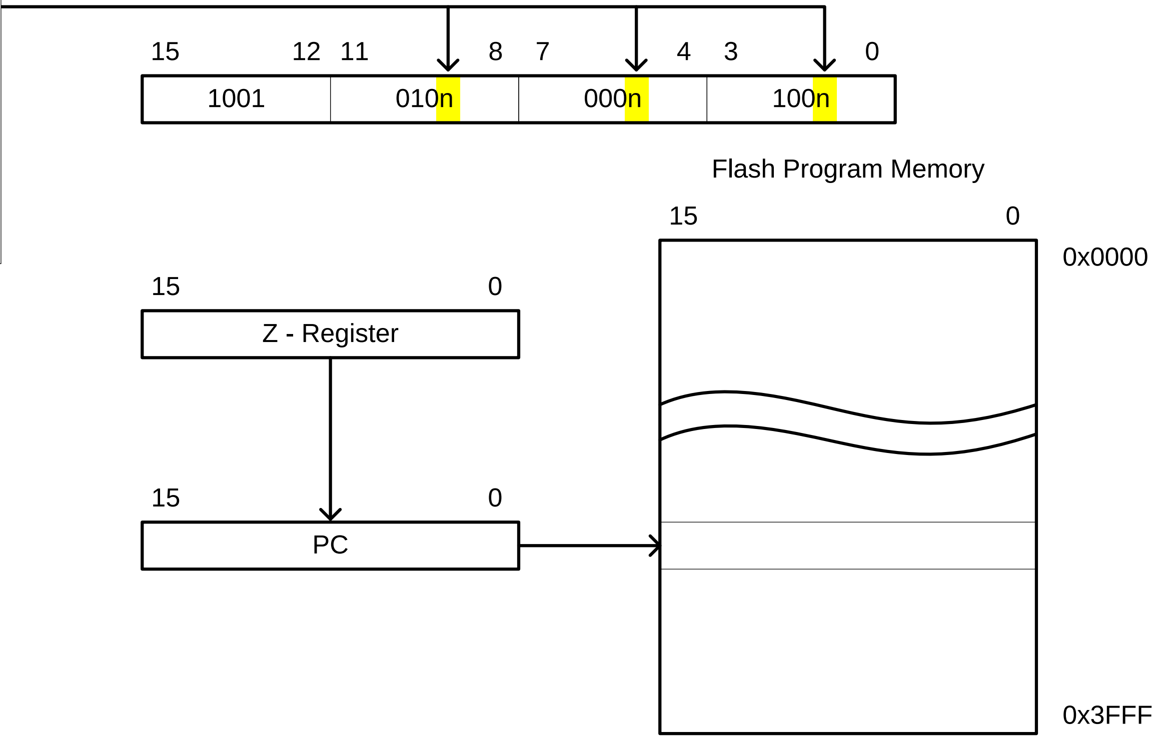


color key		
🟥	ATmega Family	4 M words
🟩	ATmega328P	16 K words

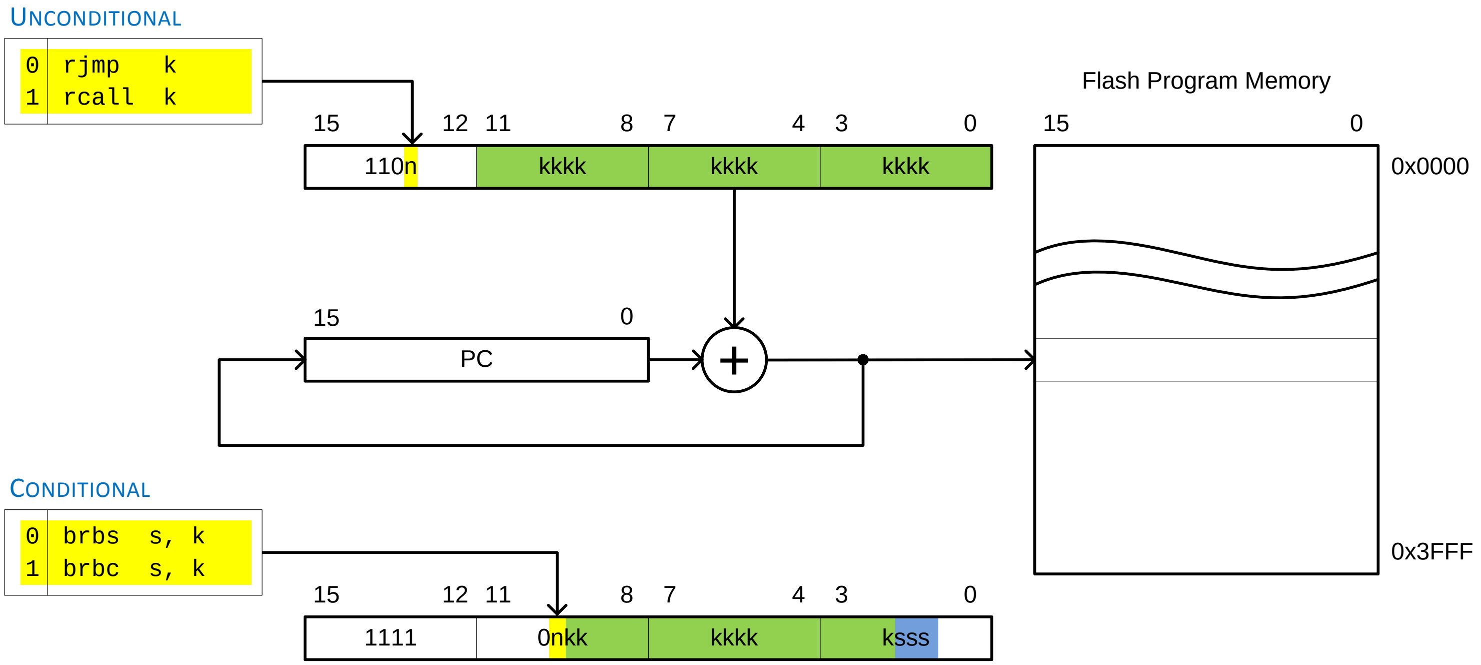
Control Transfer

Indirect

nnn	opcode	notes
000		
001	ijmp	see illustration
010		
011	eijmp	n/a to 328P
100	ret	PC ← M[SP + 1]
101	icall	see illustration
110	reti	PC ← M[SP + 1]
111	eicall	n/a to 328P



Control Transfer
Relative

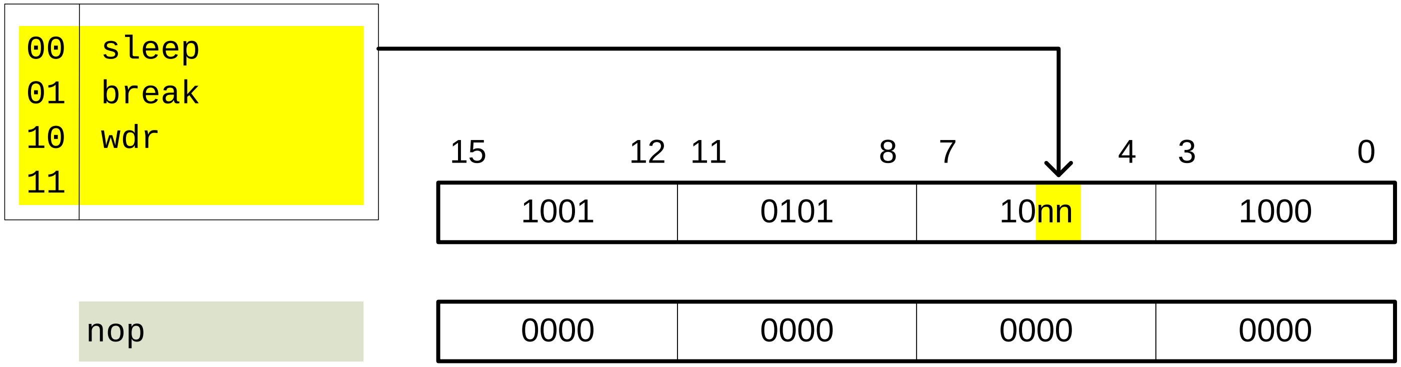


		alias			
SREG	sss	brbs s, k		brbc s, k	
I	111	brie k		brid k	
T	110	brts k		brtc k	
H	101	brhs k		brhc k	
S	100	brlt k		brge k	
V	011	brvs k		brvc k	
N	010	brmi k		brpl k	
Z	001	breq k		brne k	
C	000	brcs k	brlo k	brcc k	brsh k

Notes

1. See Register Direct Addressing for encoding of skip register bit set/clear instructions sbrc and sbrs.
2. See I/O Direct Addressing for encoding of skip I/O register bit set/clear instructions sbis and sbic.

MCU Control Instructions



00	sleep
01	break
10	wdr
11	

	15 12	11 8	7 4	3 0
	1001	0101	10nn	1000
nop	0000	0000	0000	0000

Program Decoding – Who Am I?

```
Addr  Machine
      Instruction

Who_Am_I #1:
0204  9a5d    ____  ____, ____     // I/O direct

0205  985d    ____  ____, ____     // I/O direct

0206  9508    ____


Who_Am_I #2:
01f8 934f      ____  ____          // Indirect SRAM Data Addressing

01f9 b74f      ____  ____, ____    // I/O Direct

01fa 930f      ____  ____          // Indirect SRAM Data Addressing

01fb 9180 0103 ____  ____, ____    // Direct SRAM Data Addressing

01fd 9100 0102 ____  ____, ____    // Direct SRAM Data Addressing

01ff 2380      ____  ____, ____    // Direct Register Addressing,

0200 910f      ____  ____          // Indirect SRAM Data Addressing

0201 bf4f      ____  ____, ____    // I/O Direct

0202 914f      ____  ____          // Indirect SRAM Data Addressing

0203 9508      ____
```

Program Encoding – Display

```
display:
   :
_____    lds   work0,imageR
_____    lds   spi7SEG,imageD
_____    or    spi7SEG,work0
_____    call  spiTx
   :
_____    ret
```

PROGRAM ENCODING – TURN LEFT

```
; --------------------------
; ------- Turn Left --------
turnLeft:
_____   push  reg_F
_____   in    reg_F,SREG
   :
_____   lds   work0, dir        // x = work0 bit 1, y = work0 bit 0
_____   bst   work0,0           // store y    into T
_____   bld   work1,1           // load dir.1 from T (dir.1 = y)
_____   com   work0             // store /x    into T
_____   bst   work0,1
_____   bld   work1,0           // load dir.0 from T (dir.0 = /x)
_____   sts   dir, work1
   :
_____   out   SREG, reg_F
_____   pop   reg_F
_____   ret
```

Program Encoding – In Forest and spiTxWait

inForest:

```
Address Machine Instruction
0131    ______    ldi    ZL,low(table<<1)   // load address of look-up
   :
02e8    ______    lds    work0, row         // SRAM row address = 0101
02e9    ______
02ea    ______    cpi    work0, 0xFF
02eb    ______    breq   yes
02ec    ______    clr    cppReg             // Compare to eor  cppReg, cppReg
02ed    ______    rjmp   endForest
yes:
02ee    ______    ser    cppReg             // compare to ldi  cppReg, 0xFF
endForest:
   :
02f3    ______    ret
```

spiTxWait:

```
0112    ______    in     work0,SPSR
0113    ______    bst    work0,SPIF
0114    ______    brtc   spiTxWait
0115    ______    ret
```

Program Encoding – BCD to 7-Segment Display

- Program Memory Indirect is great for implementing look-up tables located in Flash program memory – including decoders (gray code → binary, hex → seven segment, …)

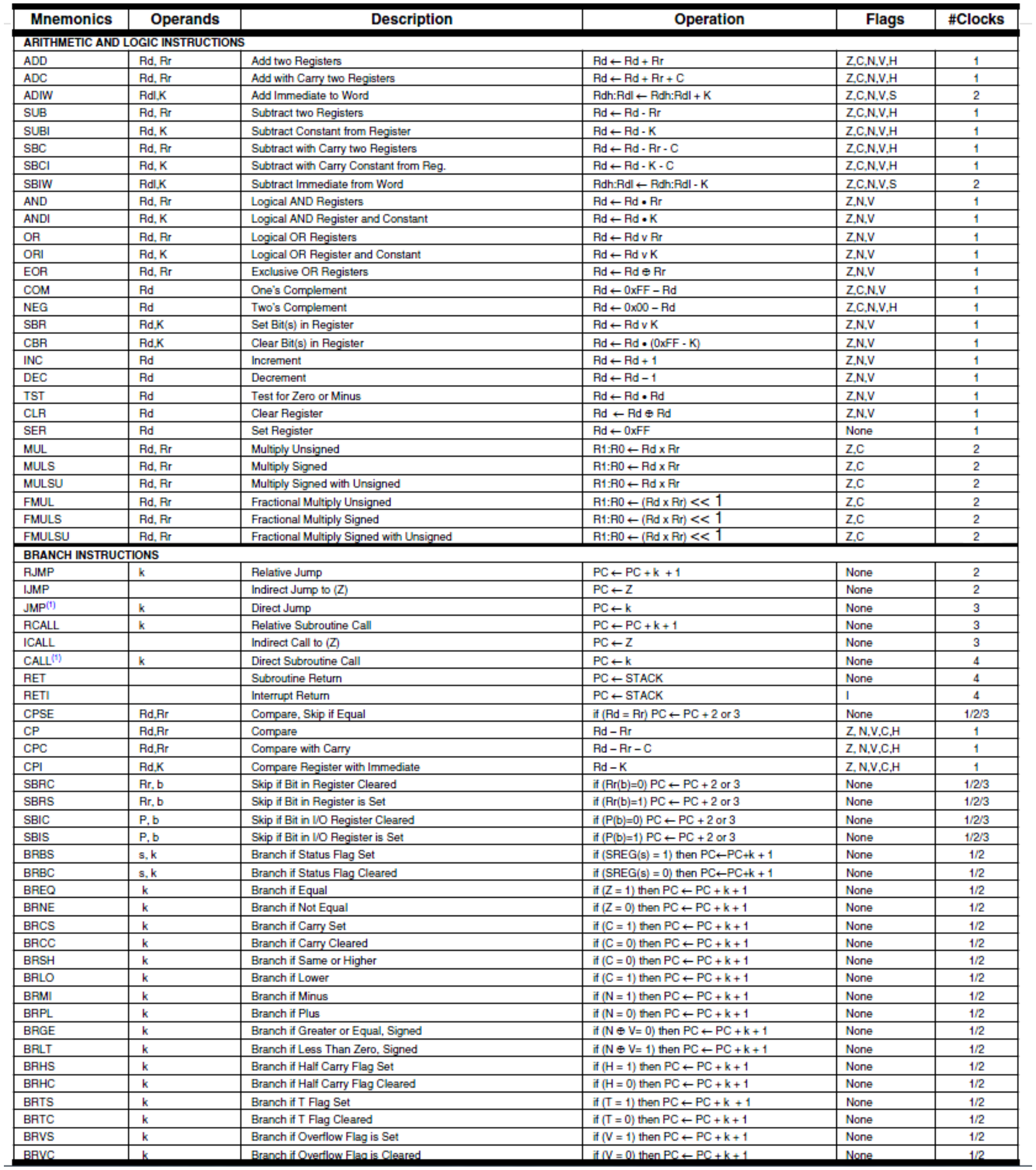
- In this example I build a 7-segment decoder in software.

```
BCD_to_7SEG:

Address  Machine Instruction

0131  _______    ldi   ZL,low(table<<1)   // load address of look-up

0132  _______    ldi   ZH,high(table<<1)

0133  _______    clr   r1

0134  _______    add   ZL, r16

0135  _______    adc   ZH, r1

0136  _______    lpm   spi7SEG, Z

0137  _______    ret

0138  _______    table: DB  0b01111110, 0b0110000, 0b1101101 …
```

PROGRAM DECODING – SRAM INDIRECT

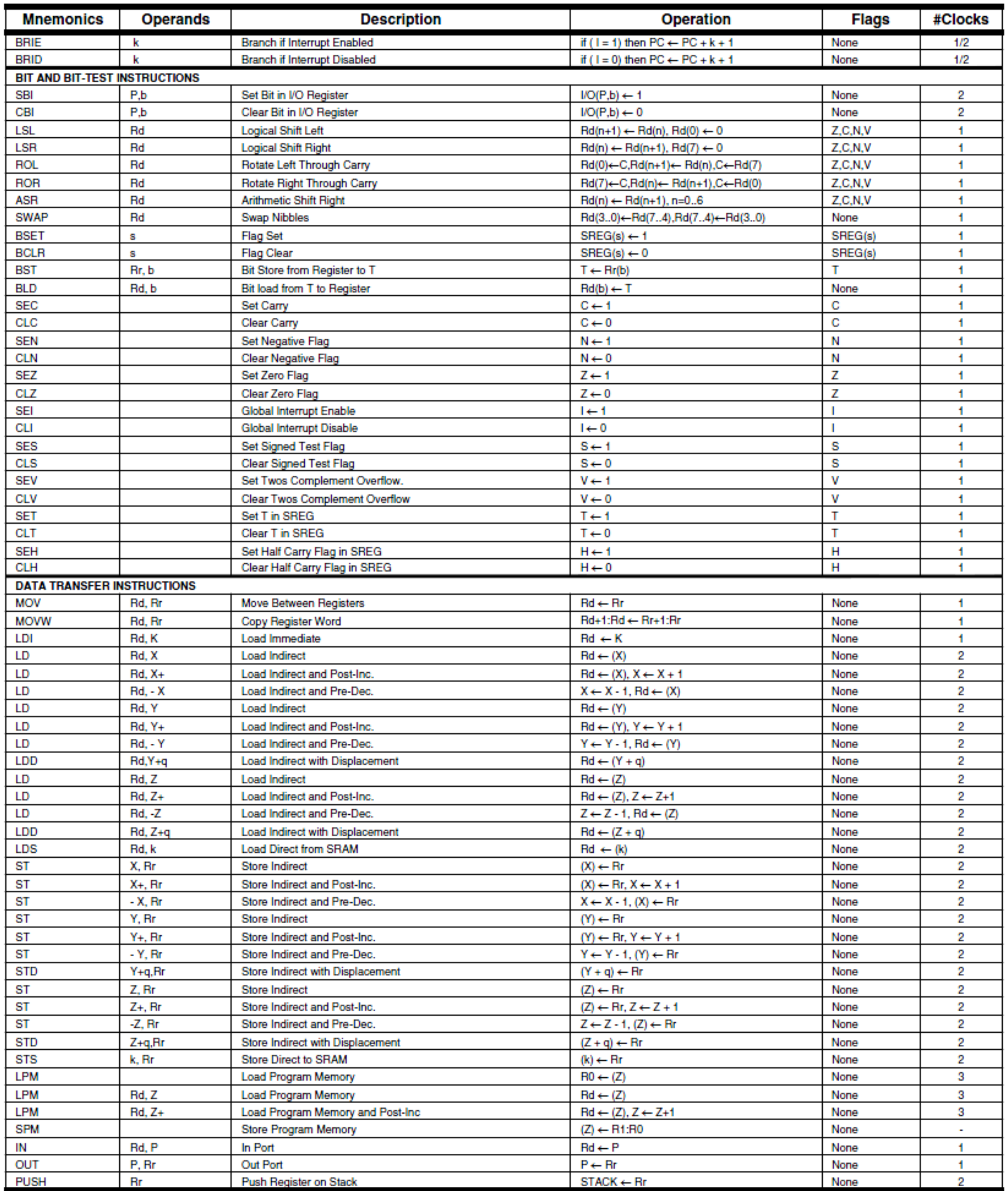
- Write and encode a program to set to ASCII Space Character (0x20), all the bytes in a 64-byte Buffer.

APPENDIX A – ATMEGA328P INSTRUCTION SET[1]

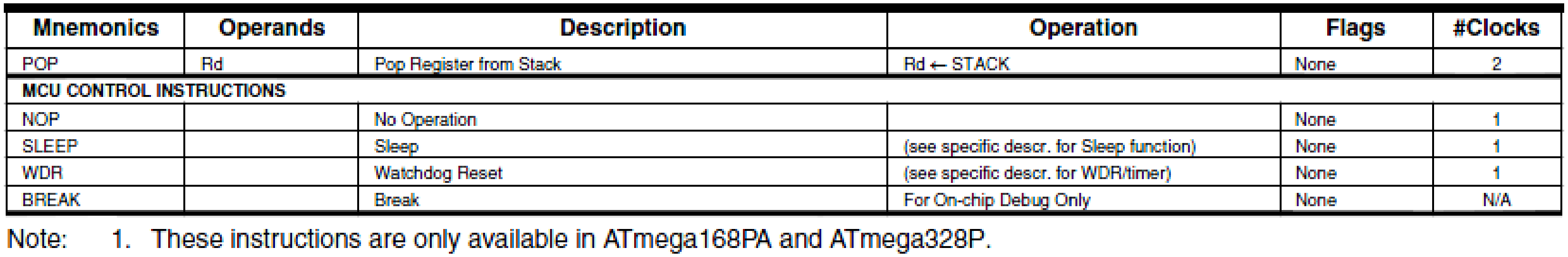


Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP(1)	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL(1)	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2

[1] Source: ATmega328P Data Sheet http://www.atmel.com/dyn/resources/prod_documents/8161S.pdf Chapter 31 Instruction Set Summary



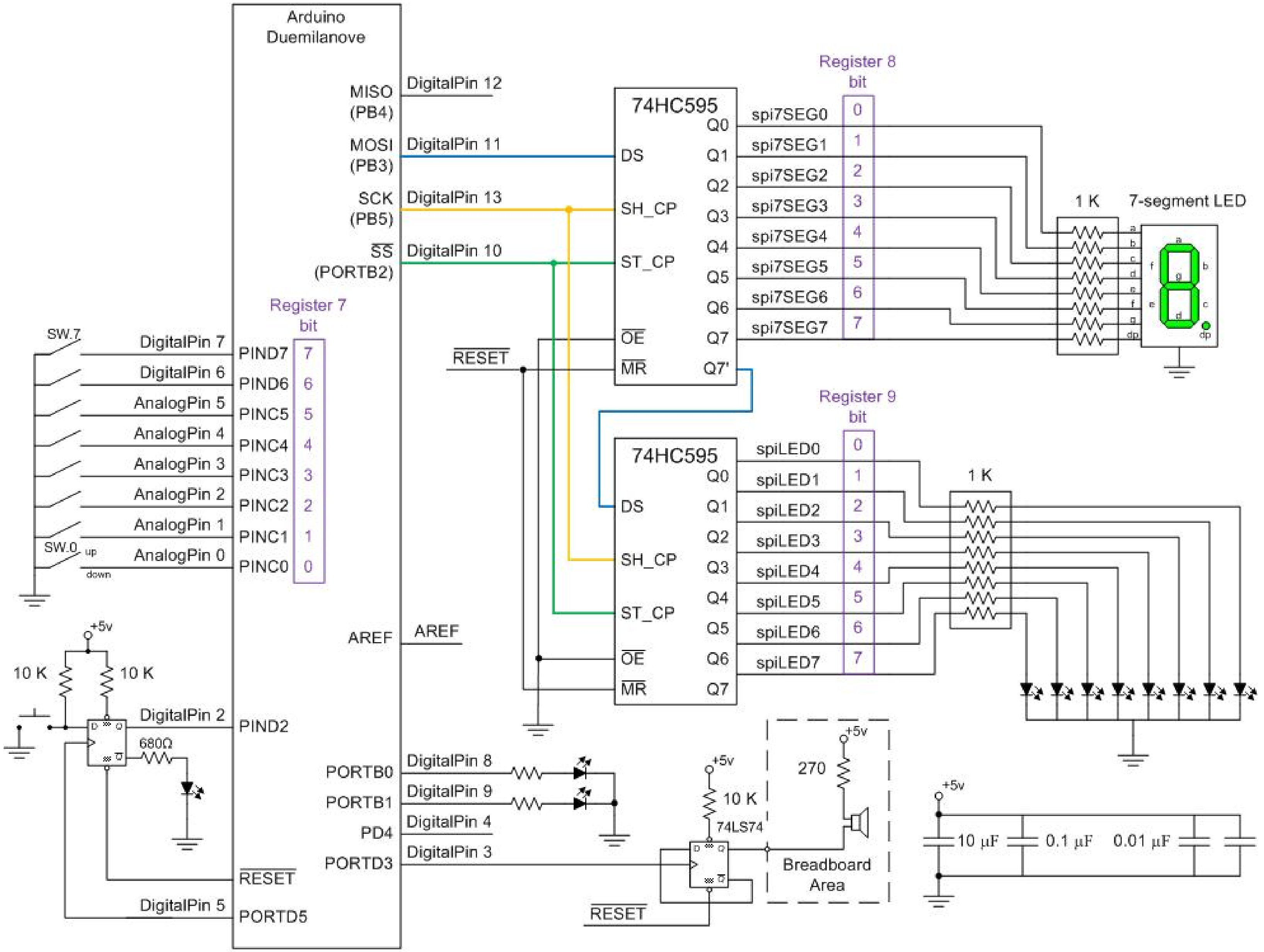
Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2



Mnemonics	Operands	Description	Operation	Flags	#Clocks
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

Note: 1. These instructions are only available in ATmega168PA and ATmega328P.

APPENDIX B – ARDUINO PROTO-SHIELD SCHEMATIC



1/11/10

SOLUTIONS

pulse: ← Who Am I #1

```
0204 9a5d      sbi  PORTD,dff_clk  // Set clock   (2 clock cycles)
0205 985d      cbi  PORTD,dff_clk  // Clear clock (2 clock cycles)
0206 9508      ret
```

hitWall: ← Who Am I #2

```
01f8 934f      push reg_F     // push any flags or registers modified
01f9 b74f      in   reg_F,SREG
01fa 930f      push work0
01fb 9180 0103 lds  cppReg,imageD
01fd 9100 0102 lds  work0,imageR
01ff 2380      and  cppReg,work0
0200 910f      pop  work0     // pop any flags or registers placed on the stack
0201 bf4f      out  SREG, reg_F
0202 914f      pop  reg_F
0203 9508      ret
```

display:

```
019a 934f      push reg_F
019b b74f      in   reg_F,SREG
019c 930f      push work0
```

```
019d 9100 0102 lds  work0,imageR

019f 9080 0103 lds  spi7SEG,imageD

01a1 2a80      or   spi7SEG,work0

01a2 940e 0109 call spiTx

01a4 910f      pop  work0

01a5 bf4f      out  SREG,reg_F

01a6 914f      pop  reg_F

01a7 9508      ret
```

turnLeft:

```
01b9 934f      push reg_F

01ba b74f      in   reg_F,SREG

01bb 930f      push work0

01bc 931f      push work1

01bd 9100 0100 lds  work0, dir     // x = work0 bit 1, y = work0 bit 0

01bf fb00      bst  work0,0   // store y    into T

01c0 f911      bld  work1,1   // load dir.1 from T (dir.1 = y)

01c1 9500      com  work0     // store /x    into T

01c2 fb01      bst  work0,1

01c3 f910      bld  work1,0   // load dir.0 from T (dir.0 = /x)

01c4 9310 0100 sts  dir, work1
```

```
01c6 911f      pop  work1

01c7 910f      pop  work0

01c8 bf4f      out  SREG, reg_F

01c9 914f      pop  reg_F

01ca 9508      ret

inForest:

02e5 92ff      push reg_F     // push any flags or registers modified

02e6 b6ff      in        reg_F,SREG

02e7 930f      push work0

02e8 9100 0101 lds  work0,row

02ea 3f0f      cpi  work0,0xFF

02eb f011      breq yes

02ec 2788      clr  cppReg    // no

02ed c001      rjmp endForest

               yes:

02ee ef8f      ser  cppReg

               endForest:

02ef 2799      clr  r25       // zero-extended to 16-bits for C++ call

02f0 910f      pop  work0     // pop any flags or registers placed on the stack

02f1 beff      out  SREG,reg_F
```

```
02f2 90ff      pop  reg_F

02f3 9508      ret

spiTxWait:

; Wait for transmission complete

0112 b50d      in   r16,SPSR

0113 fb07      bst  r16,SPIF

0114 f7ee      brtc spiTxWait

0115 9508      ret

BCD_to_7SEG:
0131 e7e0      ldi  ZL,low(table<<1)    // load address of look-up

0132 e0f2      ldi  ZH,high(table<<1)

0133 2411      clr  r1

0134 0fe0      add  ZL, r16

0135 1df1      adc  ZH, r1

0136 9084      lpm  spi7SEG, Z

0137 9508      ret

0138 307e

0139 6d6d      table: .DB  0b01111110, 0b0110000, 0b1101101, 0b1101101
```