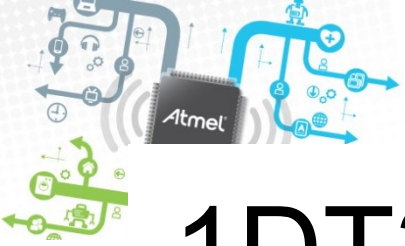
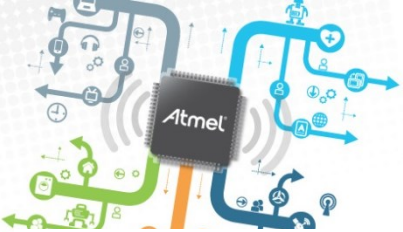


1DT301, Computer Technology

Tuesday, October 16, 2019

- Addressing mode
- Exam examples
- Questions

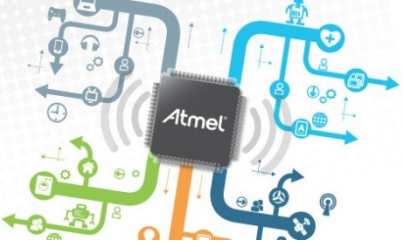


1.

a) Convert the hexadecimal number **$12B0_{16}$** and the decimal number **4784_{10}** to binary numbers and add the two binary numbers together. Give answer in hexadecimal! Show the calculations. **2p**

b) Write the octal number **4005210046_8** in hexadecimal form! **1p**

c) Write the decimal number -6 as a hexadecimal value in two's-complement form, 16 bits. **2p**



4784 to binary:

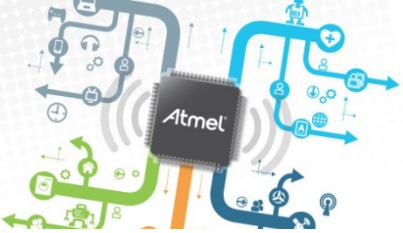
Division:	Quote:	Rest:		
4784/2	2392	0	lsb	= least significant bit
2392/2	1196	0		
1196/2	598	0		
598/2	299	0		
299/2	149	1		
149/2	74	1		
74/2	37	0		
37/2	18	1		
18/2	9	0		
9/2	4	1		
4/2	2	0		
2/2	1	0		
1/2	0	1	msb	= most significant bit

4 bits

4 bits

4 bits

The binary number is: 1 0010 1011 0000 = 0x12B0



1.

a) Convert the hexadecimal number **$12B0_{16}$** and the decimal number **4784_{10}** to binary numbers and add the two binary numbers together. Give answer in hexadecimal! Show the calculations. **2p**

b) Write the octal number **4005210046_8** in hexadecimal form! **1p**

c) Write the decimal number -6 as a hexadecimal value in two's-complement form, 16 bits. **2p**

a) Convert the hexadecimal number **$12B0_{16}$** and the decimal number **4784_{10}** to binary numbers and add the two binary numbers together. Give answer in hexadecimal! Show the calculations. **2p**

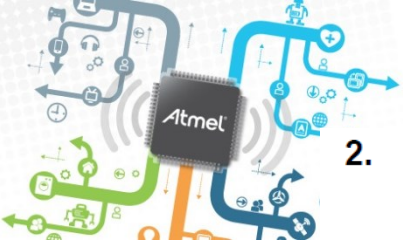
Answer: $0001\ 0010\ 1011\ 0000_2 + 0001\ 0010\ 1011\ 0000_2 = 0010\ 0101\ 0110\ 0000_2 = 2560_{16}$

b) Write the octal number **4005210046_8** in hexadecimal form! **1p**

Answer: $4005210046_8 = 100\ 000\ 000\ 101\ 010\ 001\ 000\ 000\ 100\ 110_2 = 20151026_{16}$

c) Write the decimal number -6 as a hexadecimal value in two's-complement form, 16 bits. **2p**

Answer: $+6_{10} = 0000\ 0000\ 0000\ 0110_2$ (16 bits)
1-complement = $1111\ 1111\ 1111\ 1001_2$
2-complement = 1-complement +1 = $1111\ 1111\ 1111\ 1010_2 = FFFA_{16}$



2. Machine code

Below is part of a progra
most significant byte first,
Example: The instruction
On lines B1 – B9 the ass

a) Recreate the assembler
Instruction Set Manual.

b) Make a flowchart of the p

```
+000000B1:   9731
+000000B2:   F7F1
+000000B3:   9508

+000000B4:   27FF
+000000B5:   E1EE
+000000B6:   CFFA

+000000B7:   E0F3
+000000B8:   EEE8
+000000B9:   940C00B1
```


SBIW – Subtract Immediate from Word

Description:

Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This ins
on the upper four register pairs, and is well suited for operations on the Pointer Registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd ← Rd+1:Rd - K

	Syntax:	Operands:	Program Counter:
(i)	SBIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0111	KKdd	KKKK

1001 0111 0011 0001

K=? K=00 0001

D=? d=11 = 3

Rd=R30

Rd+1: Rd=R31:R30

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

sbiw r31:r30, 1

S: N ⊕ V, For signed tests.

V: Rdh7 •$\overline{R15}$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15}$• $\overline{R14}$ •$\overline{R13}$ •$\overline{R12}$ •$\overline{R11}$• $\overline{R10}$• $\overline{R9}$• $\overline{R8}$• $\overline{R7}$• $\overline{R6}$ •$\overline{R5}$• $\overline{R4}$• $\overline{R3}$ •$\overline{R2}$• $\overline{R1}$• $\overline{R0}$
Set if the result is $0000; cleared otherwise.

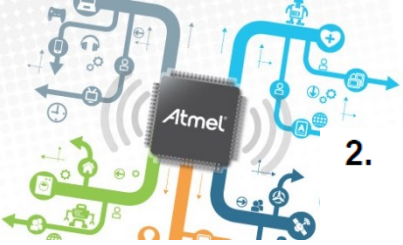
C: R15• $\overline{Rdh7}$
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
sbiw    r25:r24,1   ; Subtract 1 from r25:r24
sbiw    YH:YL,63    ; Subtract 63 from the Y-pointer(r29:r28)
```

Words: 1 (2 bytes)
Cycles: 2



2. Machine code

Below is part of a pr
most significant byte
Example: The instru
On lines B1 – B9 the

a) Recreate the assem
Instruction Set Manu

b) Make a flowchart of

```
+000000B1:    9731
+000000B2:    F7F1
+000000B3:    9508

+000000B4:    27FF
+000000B5:    E1EE
+000000B6:    CFFA

+000000B7:    E0F3
+000000B8:    EEE8
+000000B9:    940C00B1
```

BRNE – Branch if Not Equal

Description:

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is cleared. If the
cuted immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only
signed binary number represented in Rd was not equal to the unsigned or signed binary number repre
instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter
PC and is represented in two's complement form. (Equivalent to instruction BRBC 1,k).

Operation:

(i) If Rd ≠ Rr (Z = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRNE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k001

1111 0111 1111 0001

K=?

K=11 1111 0 =
111 1110

K = -2

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

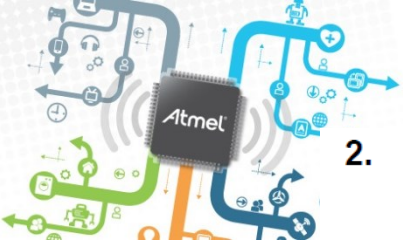
Example:

```
        eor     r27,r27   ; Clear r27
loop:   inc     r27       ; Increase r27
        ...
        cpi     r27,5     ; Compare r27 to 5
        brne    loop      ; Branch if r27<>5
        nop               ; Loop exit (do nothing)
```

PC = PC + k + 1= PC-1

brne pc-1

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true



2. Machine code

Below is part of a program, cut out from the disassembler. The machine code is printed with the most significant byte first, ie as it is described in the Instruction Set Manual.
Example: The instruction RET, the machine code is 9508.
On lines B1 – B9 the assembler code is removed.

a) Recreate the assembler code by interpreting machine code. Use the enclosed examples from the Instruction Set Manual.

b) Make a flowchart of the program and explain what the code is doing.

```
+000000B1:    9731

+000000B2:    F7F1

+000000B3:    9508        RET                    Subroutine return

+000000B4:    27FF

+000000B5:    E1EE

+000000B6:    CFFA

+000000B7:    E0F3

+000000B8:    EEE8

+000000B9:    940C00B1
```

2. Machine code

Below is part of a program, cut ou
most significant byte first, ie as it i
Example: The instruction RET, th
On lines B1 – B9 the assembler c

a) Recreate the assembler code by i
Instruction Set Manual.

b) Make a flowchart of the progra

```
+000000B1:    9731
+000000B2:    F7F1
+000000B3:    9508          RET

+000000B4:    27FF
+000000B5:    E1EE
+000000B6:    CFFA

+000000B7:    E0F3
+000000B8:    EEE8
+000000B9:    940C00B1
```

CLR – Clear Register

Description:

Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear register.

Operation:

(i) Rd ← Rd ⊕ Rd

	Syntax:	Operands:	Program Counter:
(i)	CLR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see EOR Rd,Rd)

0010	01dd	dddd	dddd

d = 11 1111 1111 = ?

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	0	0	0	1	–

S: 0
Cleared

V: 0
Cleared

N: 0
Cleared

Z: 1
Set

R (Result) equals Rd after the operation.

Example:

```
        clr   r18       ; clear r18
loop:   inc   r18       ; increase r18
        ...
        cpi   r18,$50   ; Compare r18 to $50
        brne  loop
```

Words: 1 (2 bytes)
Cycles: 1

2. Machine code

Below is part of a program, (
most significant byte first, ie
Example: The instruction RE
On lines B1 – B9 the assem

a) Recreate the assembler coc
Instruction Set Manual.

b) Make a flowchart of the prog

```
+000000B1:    9731
+000000B2:    F7F1
+000000B3:    9508

+000000B4:    27FF
+000000B5:    E1EE
+000000B6:    CFFA

+000000B7:    E0F3
+000000B8:    EEE8
+000000B9:    940C00B1
```

EOR – Exclusive OR

Description:

Performs the logical EOR between the contents of register Rd and register Rr and places th
ter Rd.

Operation:

(i) Rd ← Rd ⊕ Rr

	Syntax:	Operands:	Program Counter:
(i)	EOR Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	01rd	dddd	rrrr

d = 1 1111 d = 31
r = 1 1111 r = 31

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

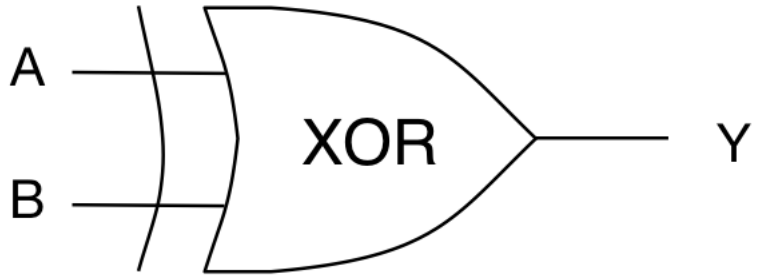
Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

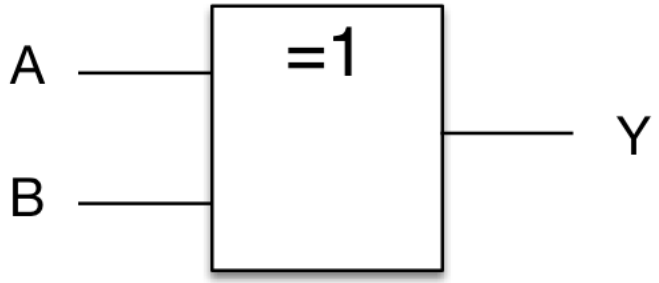
Example:

```
eor     r4,r4       ; Clear r4
```

Exclusive OR



A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0



1010 0101
1111 0000
0101 0101

1010 0101
1010 0101
0000 0000

xor r16, r16 ? = clr r16

2. Machine code

Below is part of a program, cut ou
most significant byte first, ie as it i
Example: The instruction RET, th
On lines B1 – B9 the assembler c

a) Recreate the assembler code by i
Instruction Set Manual.

b) Make a flowchart of the progra

```
+000000B1:    9731
+000000B2:    F7F1
+000000B3:    9508          RET

+000000B4:    27FF
+000000B5:    E1EE
+000000B6:    CFFA

+000000B7:    E0F3
+000000B8:    EEE8
+000000B9:    940C00B1
```

CLR – Clear Register

Description:

Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear register.

Operation:

(i) Rd ← Rd ⊕ Rd

	Syntax:	Operands:	Program Counter:
(i)	CLR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see EOR Rd,Rd)

0010	01dd	dddd	dddd

d = 11 1111 1111 = 31 =

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	0	0	0	1	–

S: 0
Cleared

V: 0
Cleared

N: 0
Cleared

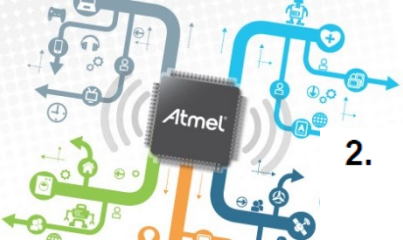
Z: 1
Set

R (Result) equals Rd after the operation.

Example:

```
        clr   r18       ; clear r18
loop:   inc   r18       ; increase r18
        ...
        cpi   r18,$50   ; Compare r18 to $50
        brne  loop
```

Words: 1 (2 bytes)
Cycles: 1



2. Machine code

Below is part of a pro
most significant byte f
Example: The instruc
On lines B1 – B9 the

a) Recreate the assemb
Instruction Set Manua

b) Make a flowchart of th

```
+000000B1:    9731
+000000B2:    F7F1
+000000B3:    9508

+000000B4:    27FF
+000000B5:    E1EE
+000000B6:    CFFA

+000000B7:    E0F3
+000000B8:    EEE8
+000000B9:    940C00B1
```

LDI – Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i) Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK
1110	0001	1110	1110

K=?

K=0001 1110

Status Register (SREG) and Boolean Formula:

dd = 30

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

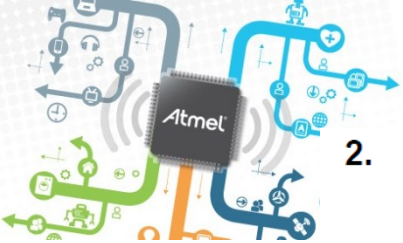
Example:

```
clr   r31       ; Clear Z high byte
ldi   r30,$F0   ; Set Z low byte to $F0
lpm             ; Load constant from Program
                ; memory pointed to by Z
```

PC = PC + k + 1= PC-1

brne pc-1

Words: 1 (2 bytes)
Cycles: 1



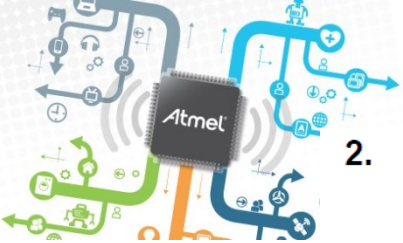
2. Machine code

Below is part of a program, cut out from the disassembler. The machine code is printed with the most significant byte first, ie as it is described in the Instruction Set Manual.
Example: The instruction RET, the machine code is 9508.
On lines B1 – B9 the assembler code is removed.

a) Recreate the assembler code by interpreting machine code. Use the enclosed examples from the Instruction Set Manual.

b) Make a flowchart of the program and explain what the code is doing.

```
+000000B1:   9731
+000000B2:   F7F1
+000000B3:   9508        RET                    Subroutine return

+000000B4:   27FF
+000000B5:   E1EE
+000000B6:   CFFA

+000000B7:   E0F3
+000000B8:   EEE8
+000000B9:   940C00B1
```



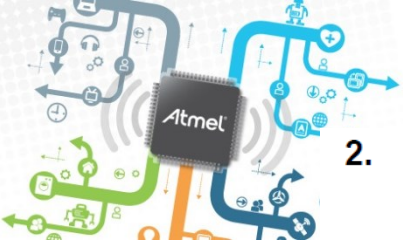
2. Machine code

Below is part of a program, cut out from the disassembler. The machine code is printed with the most significant byte first, ie as it is described in the Instruction Set Manual.
Example: The instruction RET, the machine code is 9508.
On lines B1 – B9 the assembler code is removed.

a) Recreate the assembler code by interpreting machine code. Use the enclosed examples from the Instruction Set Manual.

b) Make a flowchart of the program and explain what the code is doing.

```
+000000B1:   9731
+000000B2:   F7F1
+000000B3:   9508        RET                    Subroutine return

+000000B4:   27FF
+000000B5:   E1EE
+000000B6:   CFFA

+000000B7:   E0F3
+000000B8:   EEE8
+000000B9:   940C00B1
```



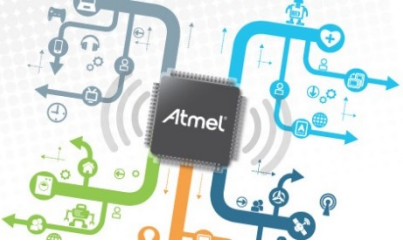
2. Machine code

Below is part of a program, cut out from the disassembler. The machine code is printed with the most significant byte first, ie as it is described in the Instruction Set Manual.
Example: The instruction RET, the machine code is 9508.
On lines B1 – B9 the assembler code is removed.

a) Recreate the assembler code by interpreting machine code. Use the enclosed examples from the Instruction Set Manual.

b) Make a flowchart of the program and explain what the code is doing.

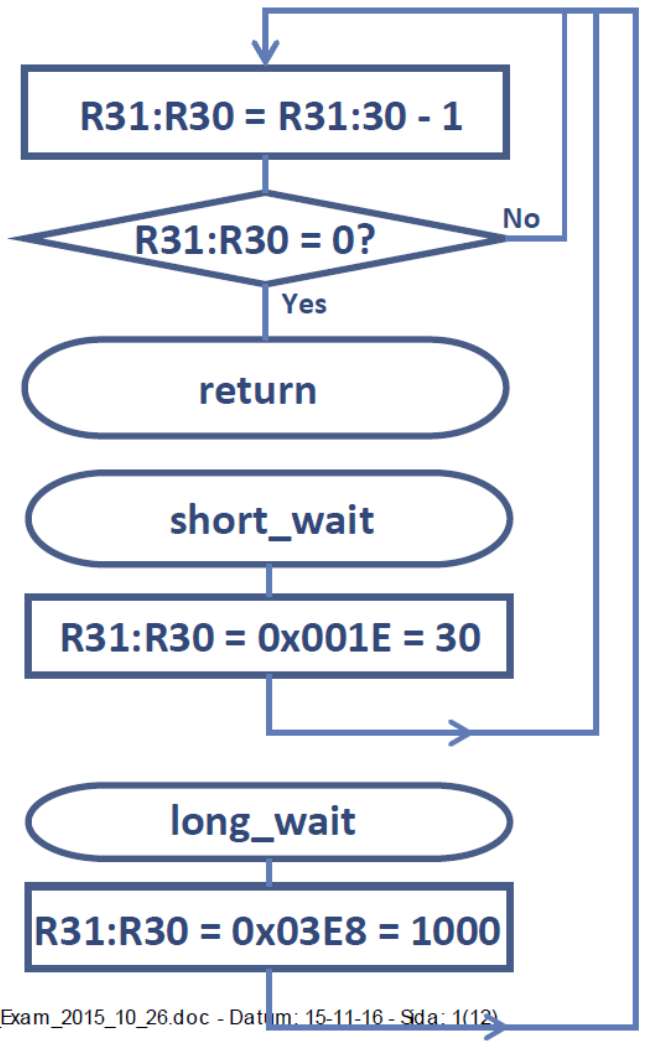
```
+000000B1:   9731
+000000B2:   F7F1
+000000B3:   9508        RET                    Subroutine return

+000000B4:   27FF
+000000B5:   E1EE
+000000B6:   CFFA

+000000B7:   E0F3
+000000B8:   EEE8
+000000B9:   940C00B1
```

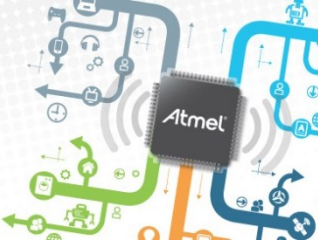


s doing. **5p**

b) Make a flowchart of the program and explain what the cc

```
@000000B1: wait_loop
125:    sbiw z, 1
+000000B1:    9731          SBIW      R30,0x01
126:    brne wait_loop
+000000B2:    F7F1          BRNE      PC-0x01
127:    ret
+000000B3:    9508          RET
@000000B4: short_wait
130:    clr zh
+000000B4:    27FF          CLR       R31
131:    ldi zl, 30
+000000B5:    E1EE          LDI       R30,0x1E
132:    rjmp wait_loop
+000000B6:    CFFA          RJMP      PC-0x0005
@000000B7: long_wait
134:    ldi zh, HIGH(1000)
+000000B7:    E0F3          LDI       R31,0x03
135:    ldi zl, LOW(1000)
+000000B8:    EEE8          LDI       R30,0xE8
136:    jmp wait_loop
+000000B9:    940C00B1      JMP       0x000000B1
```





4. Programming, assembler.

You shall write a subroutine that can be used in a bicycle speed computer. The subroutine will compute the speed in km/h. (kilometers per hour).
A magnetic sensor is hooked up to an interrupt input that gives an interrupt for each turn the front wheel rotates on the bicycle. The time that has elapsed since last interrupt is available in register pair R21, R20. The value is in ms, milliseconds, and is a value between 40 and 2000. If the value is above 2000, the speed shall be set to 0.

Formula:

$$v = \frac{d}{t} = \frac{d \cdot 10^{-2}}{t \cdot 10^{-3}} \cdot [m/s] = \frac{d \cdot 10}{t}[m/s] = \frac{d \cdot 10 \cdot 3{,}6}{t}[km/h] = \frac{d \cdot 36}{t}[km/h]$$

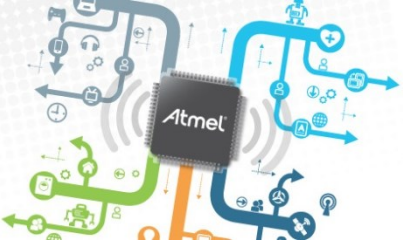
v= speed in km/h
d = wheel circumference in cm, centimeter (100 – 255)
t = rotation time in ms, milliseconds (40 – 2000)

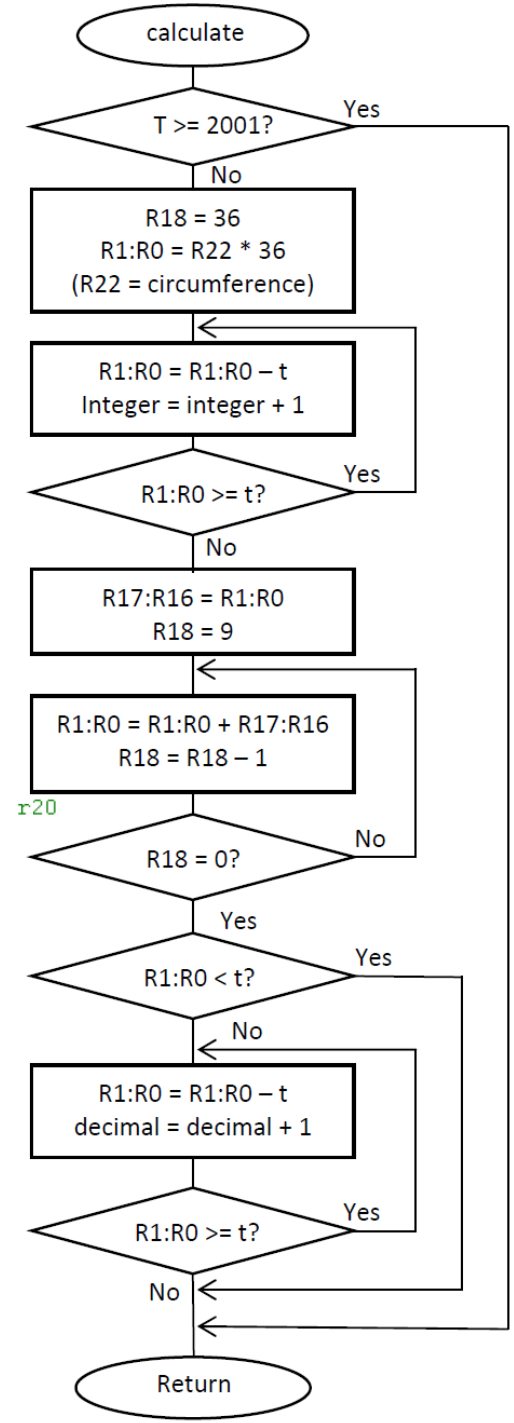
Your task is to make a program that will do the calculation above.
The CPU that will be used does not have instructions for division. Instead you have to use subtract instructions. Do the subtractions in a loop, and repeat until the reminder is close to zero. The calculation shall give one decimal digit, as in the example below:

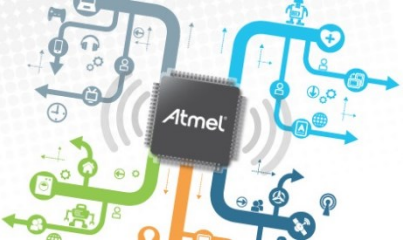
Example:
d = 220 cm
t = 302 ms

$$v = \frac{d}{t} = \frac{d \cdot 36}{t}[km/h] = \frac{220 \cdot 36}{302}[km/h] = 26{,}2[km/h]$$

Input values:
Time since last interrupt: R21, R20
Circumference in cm: R22
Output values:
Speed, integer part in R25
Speed, decimal part in R24

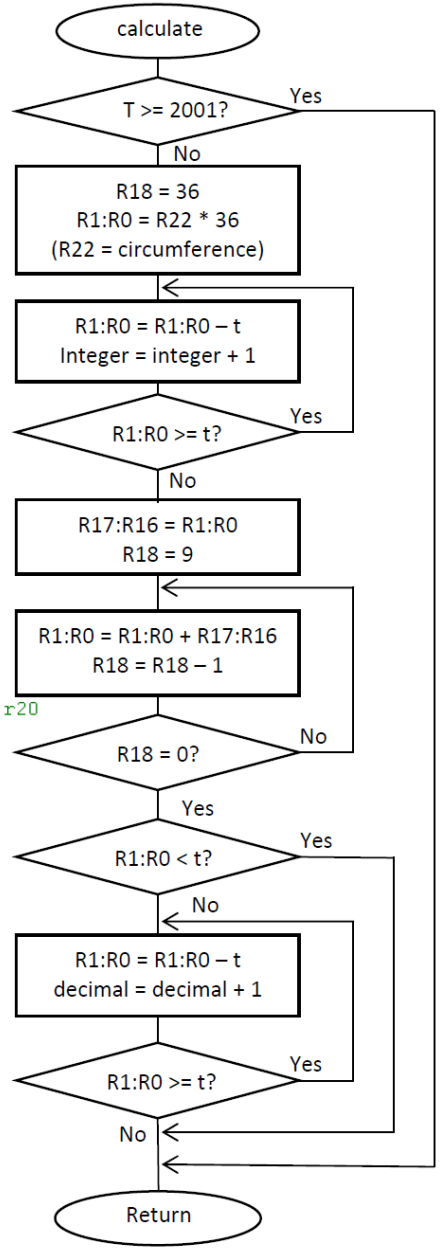

Atmel

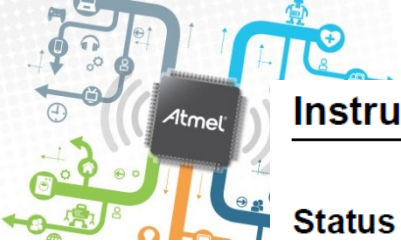

calculate
T >= 2001?
Yes
No
R18 = 36
R1:R0 = R22 * 36
(R22 = circumference)
R1:R0 = R1:R0 – t
Integer = integer + 1
R1:R0 >= t?
Yes
No
R17:R16 = R1:R0
R18 = 9
R1:R0 = R1:R0 + R17:R16
R18 = R18 – 1
r20
R18 = 0?
No
Yes
R1:R0 < t?
Yes
No
R1:R0 = R1:R0 – t
decimal = decimal + 1
R1:R0 >= t?
Yes
No
Return




```
speed_calculate:
;   Input values:
;       r21:r20  rotation time [ms]
;       r22  circumference [cm]
;   Output values:
;       r25(integer)  integer digit, speed [km/h]
;       r24(decimal)  decimal digit, speed [km/h]
;   Used registers:
;       r0, r1, r16, r17, r18
.def t_high = r21
.def t_low =r20
    push r0
    push r1
    push r16
    push r17
    push r18
    clr integer          ; integer digit
    clr decimal          ; decimal digit

compare_2000:
    ldi r17, HIGH(2001)
    ldi r16, LOW(2001)
    cp  t_low, r16       ; rotation time > 2000 ms?
    cpc t_high, r17
    brge end             ; return if >2000, speed = 0

    ldi r18, 36          ; constant for multiplication
    mul r22, r18         ; circumference x 36, result in r1:r0

subtract_integer:
    sub r0, t_low
    sbc r1, t_high       ; subtract registerpair, r1:r0 - r21:r20
    inc integer
    cp  r0, t_low        ; subtract until reminder is less than r21:r20
    cpc r1, t_high
    brge subtract_integer

; multiply reminder in r1:r0 with constant=10.
; copy value to r17:r16 and add 9 times (Instead of multiplication)

    mov r16, r0
    mov r17, r1
    ldi r18,9            ; r18 = counter for addition
add_9_times:
    add r0, r16
    adc r1, r17
    dec r18
    brne add_9_times
```

```
decimal_digit:
    cp  r0, t_low        ; check if reminder is less than t
    cpc r1, t_high       ; if so, decimal digit is 0
    brlt end             ; jump to end
subtract_decimal:
    sub r0, t_low        ; subtract until reminder is less than t
    sbc r1, t_high
    inc decimal          ; increment decimal digit
    cp  r0, t_low
    cpc r1, t_high
    brge subtract_decimal
end:
    pop r18
    pop r17
    pop r16
    pop r1
    pop r0
    ret
```





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)

ATMEL®

8-bit AVR® Instruction Set

I/O Registers

RAMPX, RAMPY, RAMPZ

Registers concatenated with the X-, Y-, and Z-registers enabling indirect addressing of the whole data space on MCUs with more than 64K bytes data space, and constant data fetch on MCUs with more than 64K bytes program space.

RAMPD

Register concatenated with the Z-register enabling direct addressing of the whole data space on MCUs with more than 64K bytes data space.

EIND

Register concatenated with the instruction word enabling indirect jump and call to the whole program space on MCUs with more than 64K bytes program space.

Stack

STACK: Stack for return address and pushed registers

SP: Stack Pointer to STACK

Flags

⇔: Flag affected by instruction

0: Flag cleared by instruction

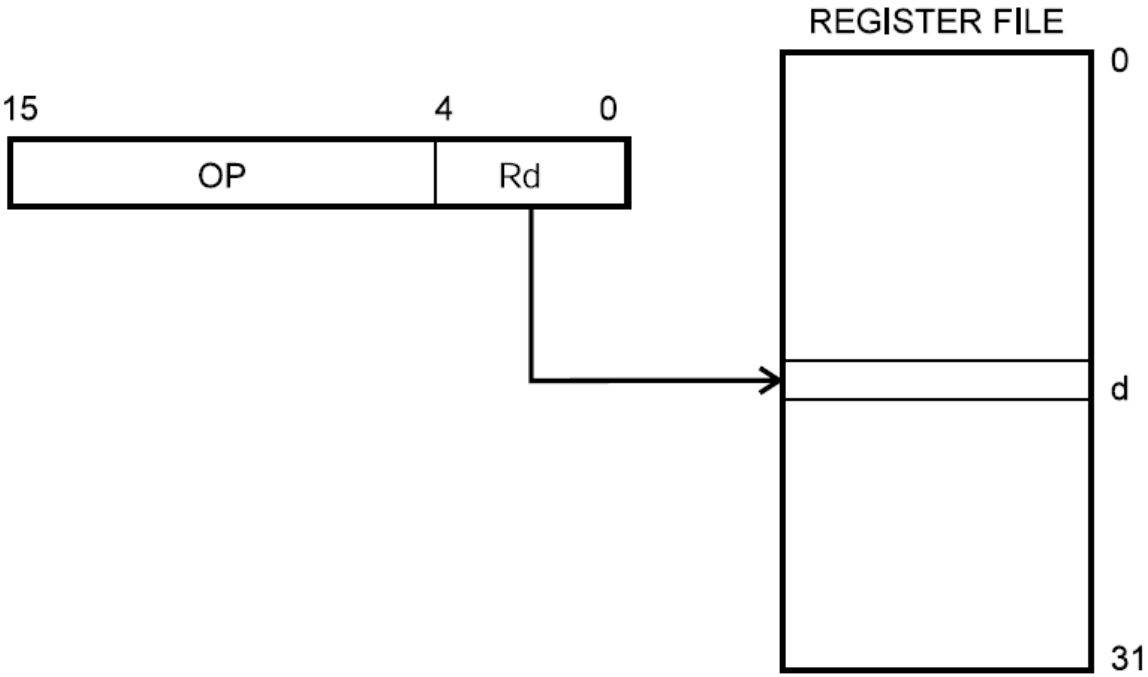
1: Flag set by instruction

-: Flag not affected by instruction

Register Direct, Single Register Rd

Register Direct, Single Register Rd

Figure 1. Direct Single Register Addressing



The operand is contained in register d (Rd).

Register Direct, Single Register Rd:

COM – One's Complement

Description:

This instruction performs a One's Complement of register Rd.

Operation:

(i) Rd ← $FF - Rd

	Syntax:	Operands:	Program Counter:
(i)	COM Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0000

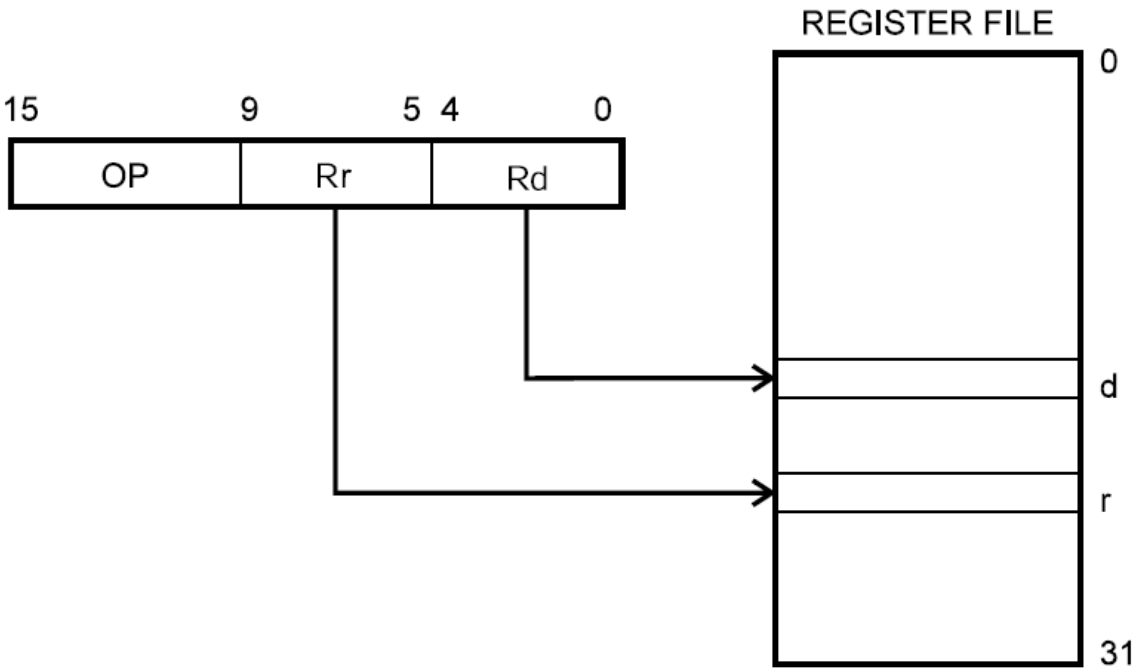
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	1

Register Direct, Two Registers Rd and Rr

Register Direct, Two Registers Rd and Rr

Figure 2. Direct Register Addressing, Two Registers



Operands are contained in register r (Rr) and d (Rd). The result is stored in register d (Rd).

Register Direct, Two Registers Rd and Rr:

ADD – Add without Carry

Description:

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

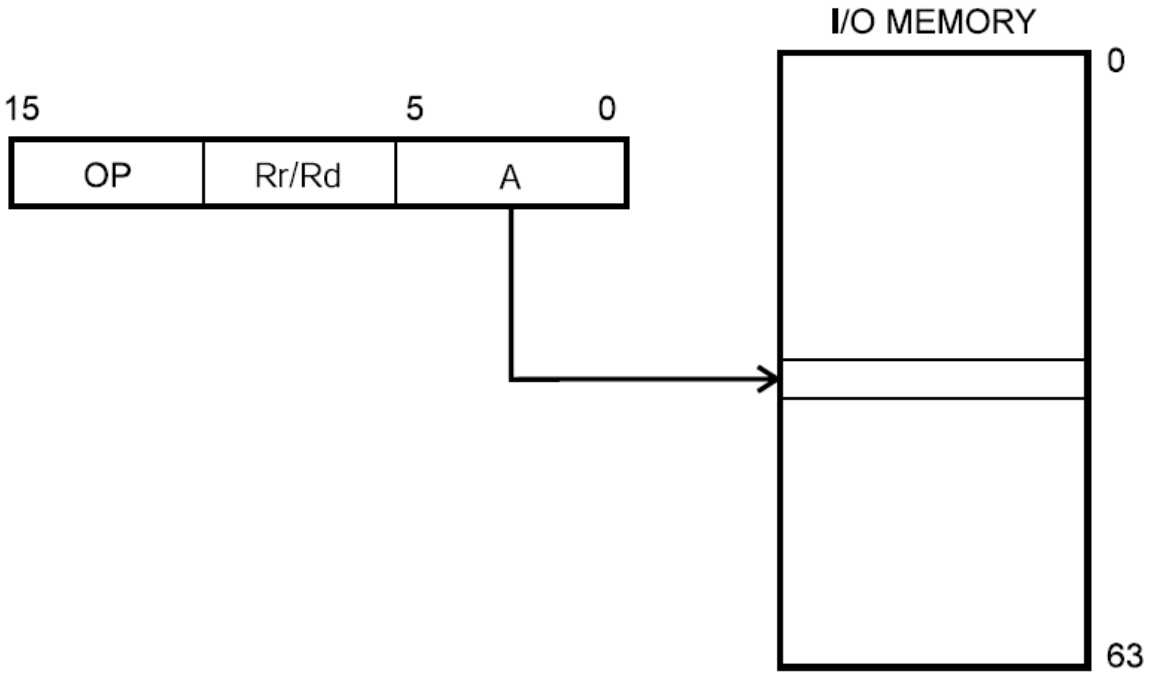
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

I/O Direct Addressing

I/O Direct

Figure 3. I/O Direct Addressing



I/O Direct Addressing.

OUT – Store Register to I/O Location

Description:

Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers etc.).

Operation:

(i) I/O(A) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	OUT A,Rr	$0 \le r \le 31, 0 \le A \le 63$	PC ← PC + 1

16-bit Opcode:

1011	1AAr	rrrr	AAAA

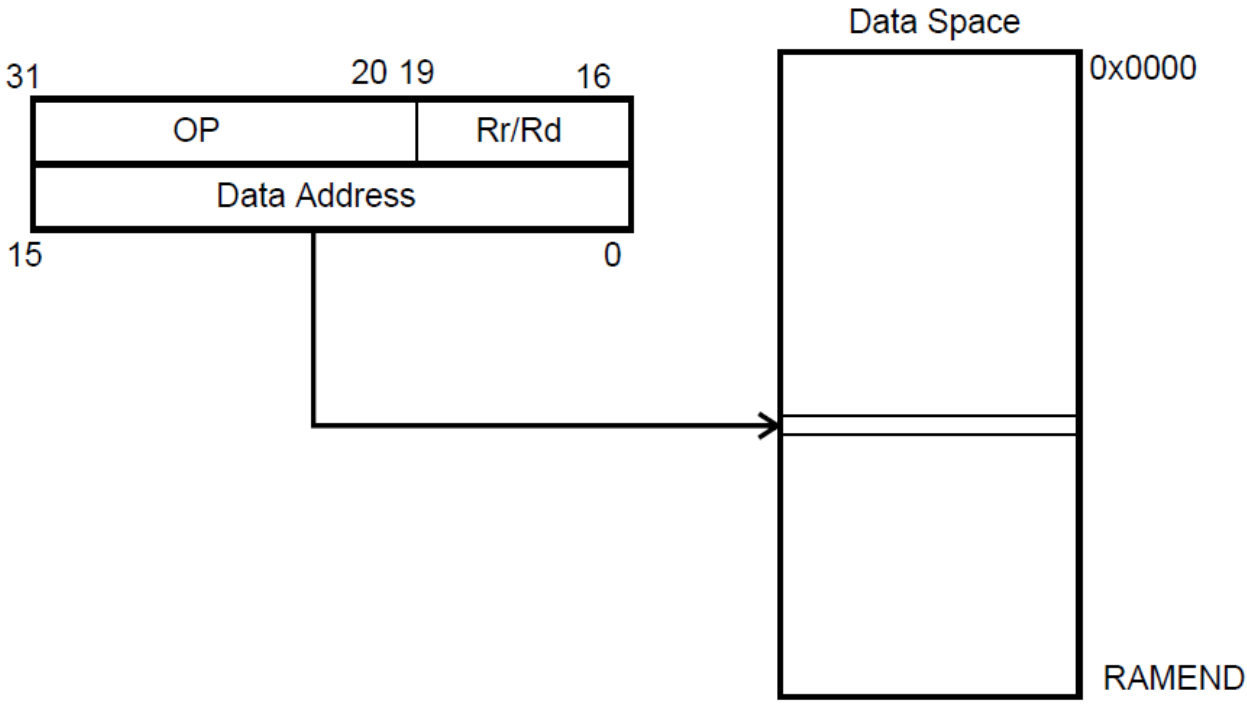
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Direct Data Addressing

Data Direct

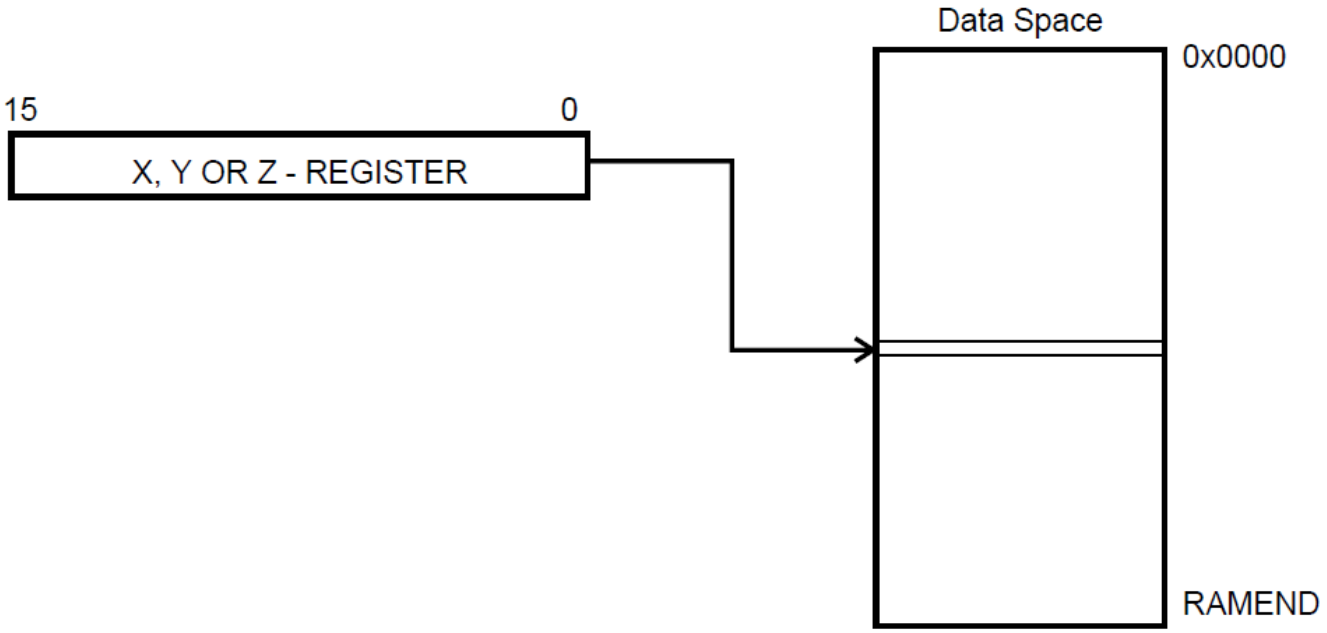
Figure 4. Direct Data Addressing



Data Indirect

Data Indirect

Figure 6. Data Indirect Addressing



Data Indirect

ST (STD) – Store Indirect From Register to Data Space using Index Z

Description:

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for Stack Pointer usage of the Z-pointer Register, however because the Z-pointer Register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y-pointer as a dedicated Stack Pointer. Note that only the low byte of the Z-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ Register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes Program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

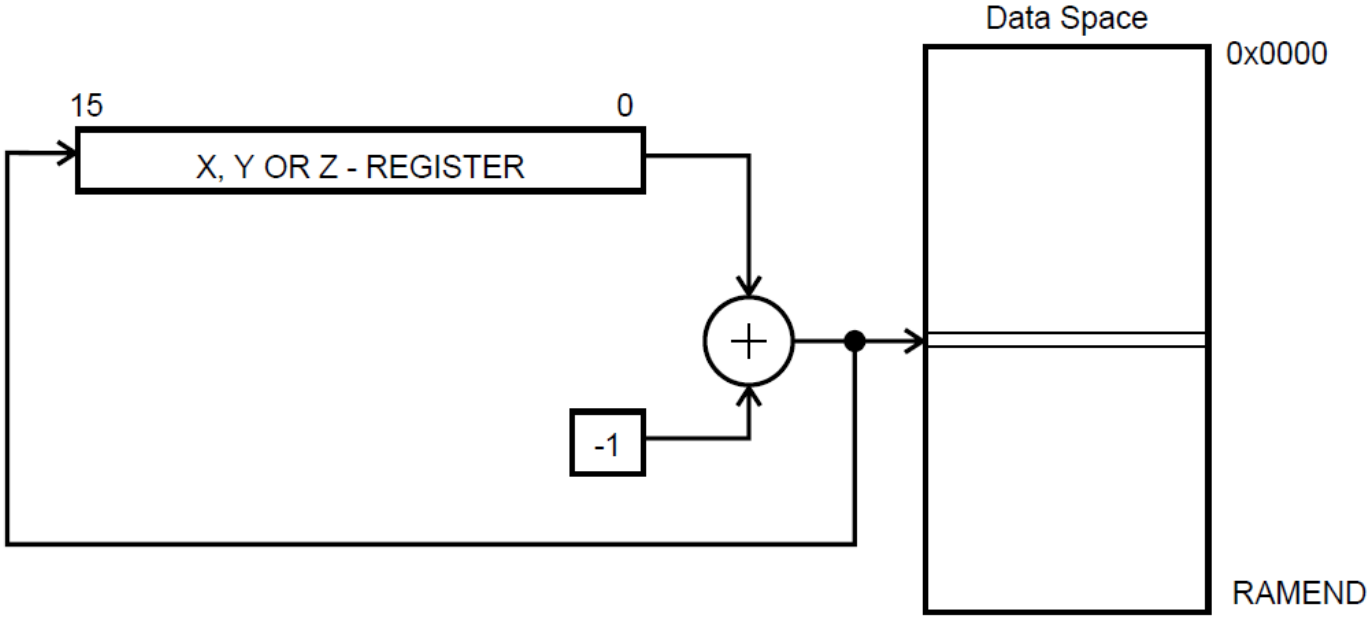
Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

	Syntax:	Operands:	Program Counter:
(i)	ST Z, Rr	$0 \le r \le 31$	PC ← PC + 1

Data Indirect with Pre-decrement

Data Indirect with Pre-decrement

Figure 7. Data Indirect Addressing with Pre-decrement



Data Indirect with Pre-decrement

ST (STD) – Store Indirect From Register to Data Space using Index Z

Description:

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for Stack Pointer usage of the Z-pointer Register, however because the Z-pointer Register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y-pointer as a dedicated Stack Pointer. Note that only the low byte of the Z-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ Register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes Program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

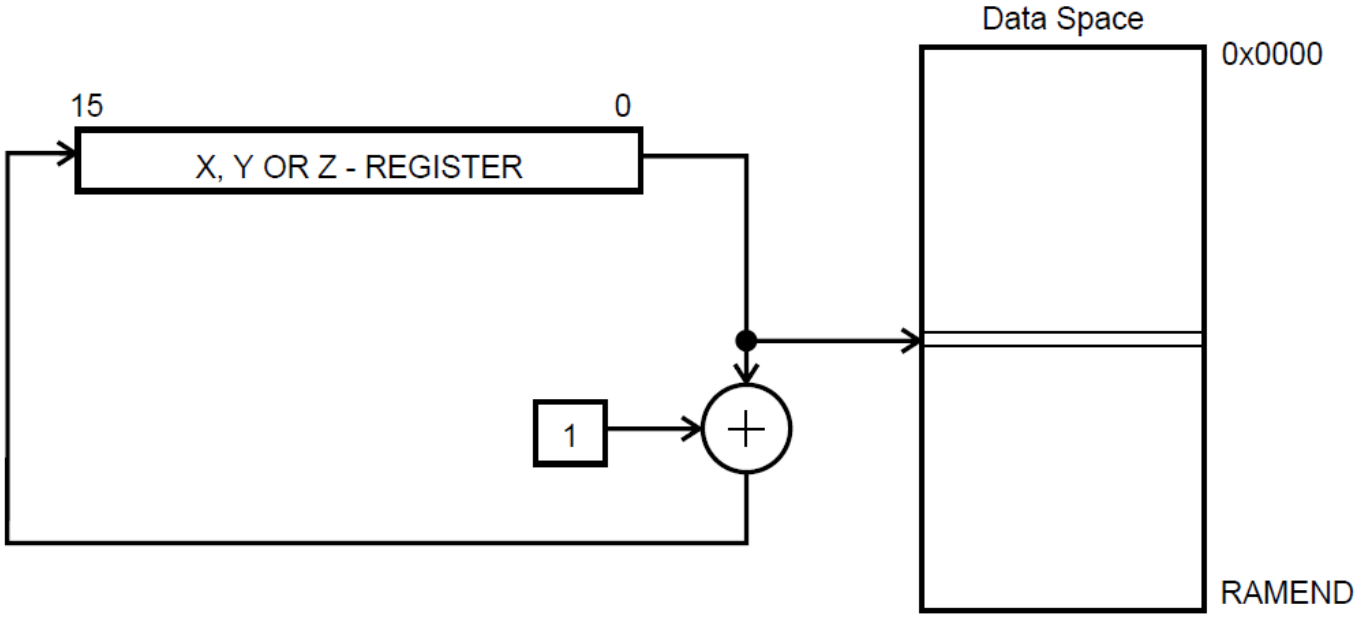
Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

	Syntax:	Operands:	Program Counter:
(iii)	ST -Z, Rr	$0 \le r \le 31$	PC ← PC + 1

Data Indirect with Post-increment

Data Indirect with Post-increment

Figure 8. Data Indirect Addressing with Post-increment



Data Indirect with Pre-decrement

ST (STD) – Store Indirect From Register to Data Space using Index Z

Description:

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for Stack Pointer usage of the Z-pointer Register, however because the Z-pointer Register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y-pointer as a dedicated Stack Pointer. Note that only the low byte of the Z-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ Register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes Program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

	Syntax:	Operands:	Program Counter:
(ii)	ST Z+, Rr	$0 \le r \le 31$	PC ← PC + 1

Direct Data Addressing:

STS – Store Direct to Data Space

Description:

Stores one byte from a Register to the data space. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64K bytes. The STS instruction uses the RAMPD Register to access memory above 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPD in register in the I/O area has to be changed.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) (k) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	STS k,Rr	$0 \le r \le 31, 0 \le k \le 65535$	PC ← PC + 2

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

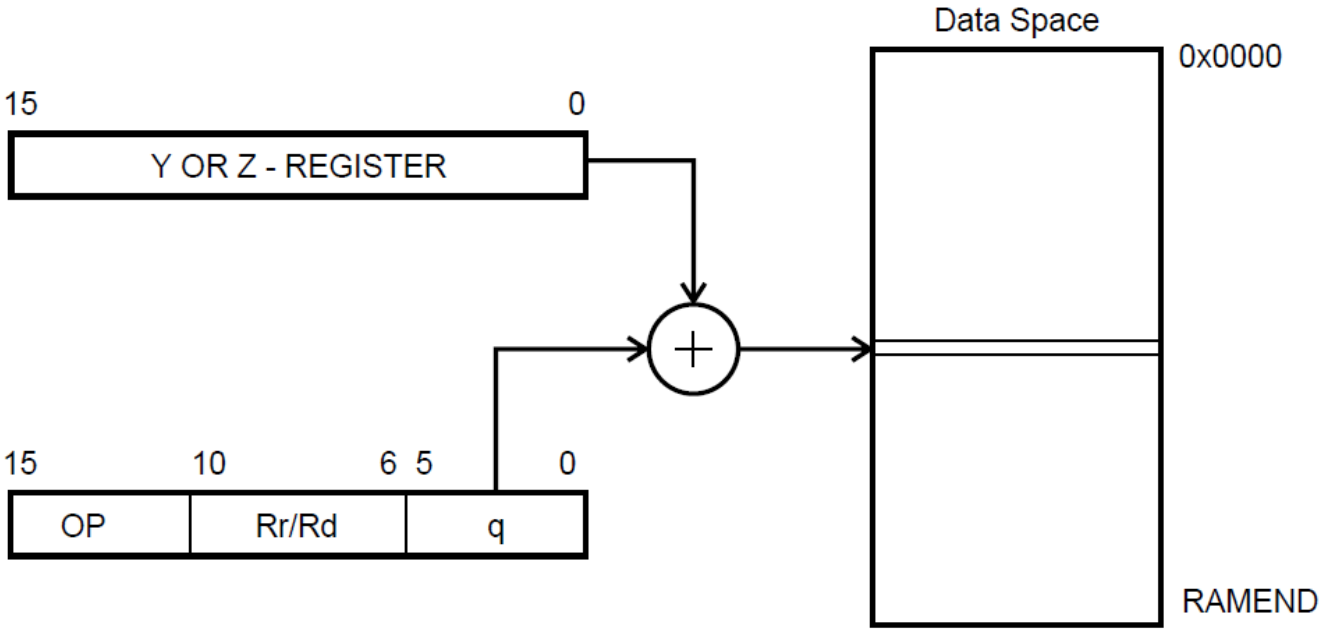
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Data Indirect with Displacement

Data Indirect with Displacement

Figure 5. Data Indirect with Displacement



Data Indirect with Displacement

ST (STD) – Store Indirect From Register to Data Space using Index Z

Description:

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for Stack Pointer usage of the Z-pointer Register, however because the Z-pointer Register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y-pointer as a dedicated Stack Pointer. Note that only the low byte of the Z-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ Register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes Program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

(iiii)	STD Z+q, Rr	$0 \le r \le 31, 0 \le q \le 63$	$PC \leftarrow PC + 1$

(STD) – Store Indirect From Register to Data Space using Index Y

ST Y+, r28
ST Y+, r29
ST -Y, r28
ST -Y, r29

Using the Y-pointer:

	Operation:		Comment:
(i)	(Y) ← Rr		Y: Unchanged
(ii)	(Y) ← Rr	Y ← Y+1	Y: Post incremented
(iii)	Y ← Y - 1	(Y) ← Rr	Y: Pre decremented
(iiii)	(Y+q) ← Rr		Y: Unchanged, q: Displacement

	Syntax:	Operands:	Program Counter:
(i)	ST Y, Rr	$0 \le r \le 31$	PC ← PC + 1
(ii)	ST Y+, Rr	$0 \le r \le 31$	PC ← PC + 1
(iii)	ST -Y, Rr	$0 \le r \le 31$	PC ← PC + 1
(iiii)	STD Y+q, Rr	$0 \le r \le 31, 0 \le q \le 63$	PC ← PC + 1

16-bit Opcode:

(i)	1000	001r	rrrr	1000
(ii)	1001	001r	rrrr	1001
(iii)	1001	001r	rrrr	1010
(iiii)	10q0	qq1r	rrrr	1qqq

(STD) – Store Indirect From Register to Data Space using Index Y

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr     r29         ; Clear Y high byte
ldi     r28,$60     ; Set Y low byte to $60
st      Y+,r0       ; Store r0 in data space loc. $60(Y post inc)
st      Y,r1        ; Store r1 in data space loc. $61
ldi     r28,$63     ; Set Y low byte to $63
st      Y,r2        ; Store r2 in data space loc. $63
st      -Y,r3       ; Store r3 in data space loc. $62(Y pre dec)
std     Y+2,r4      ; Store r4 in data space loc. $64
```

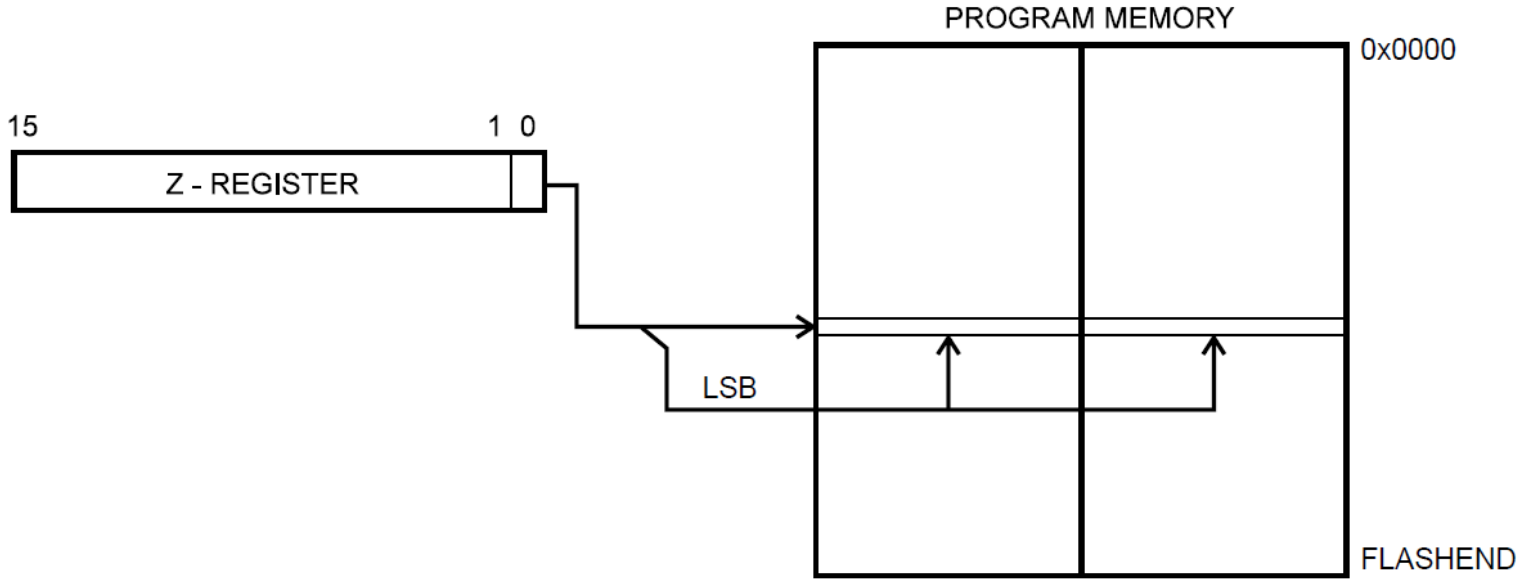
Words: 1 (2 bytes)

Cycles: 2

Program memory Constant Addressing using the LPM, ELPM and SPM Instructions.

Program Memory Constant Addressing using the LPM, ELPM, and SPM Instructions

Figure 9. Program Memory Constant Addressing



Program memory Constant Addressing using the LPM, ELPM and SPM Instructions.

LPM – Load Program Memory

Description:

Loads one byte pointed to by the Z-register into the destination register Rd. This instruction features a 100% space effective constant initialization or constant data fetch. The Program memory is organized in 16-bit words while the Z-pointer is a byte address. Thus, the least significant bit of the Z-pointer selects either low byte (Z_{LSB} = 0) or high byte (Z_{LSB} = 1). This instruction can address the first 64K bytes (32K words) of Program memory. The Z-pointer Register can either be left unchanged by the operation, or it can be incremented. The incrementation does not apply to the RAMPZ Register.

Devices with Self-Programming capability can use the LPM instruction to read the Fuse and Lock bit values. Refer to the device documentation for a detailed description.

Not all variants of the LPM instruction are available in all devices. Refer to the device specific instruction set summary. The LPM instruction is not implemented at all in the AT90S1200 device.

The result of these combinations is undefined:

	Syntax:	Operands:	Program Counter:
(i)	LPM	None, R0 implied	PC ← PC + 1
(ii)	LPM Rd, Z	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	LPM Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1

Program memory Constant Addressing using the LPM, ELPM and SPM Instructions.

(i)	LPM	None, R0 implied	PC ← PC + 1
(ii)	LPM Rd, Z	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	LPM Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

(i)	1001	0101	1100	1000
(ii)	1001	000d	dddd	0100
(iii)	1001	000d	dddd	0101

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

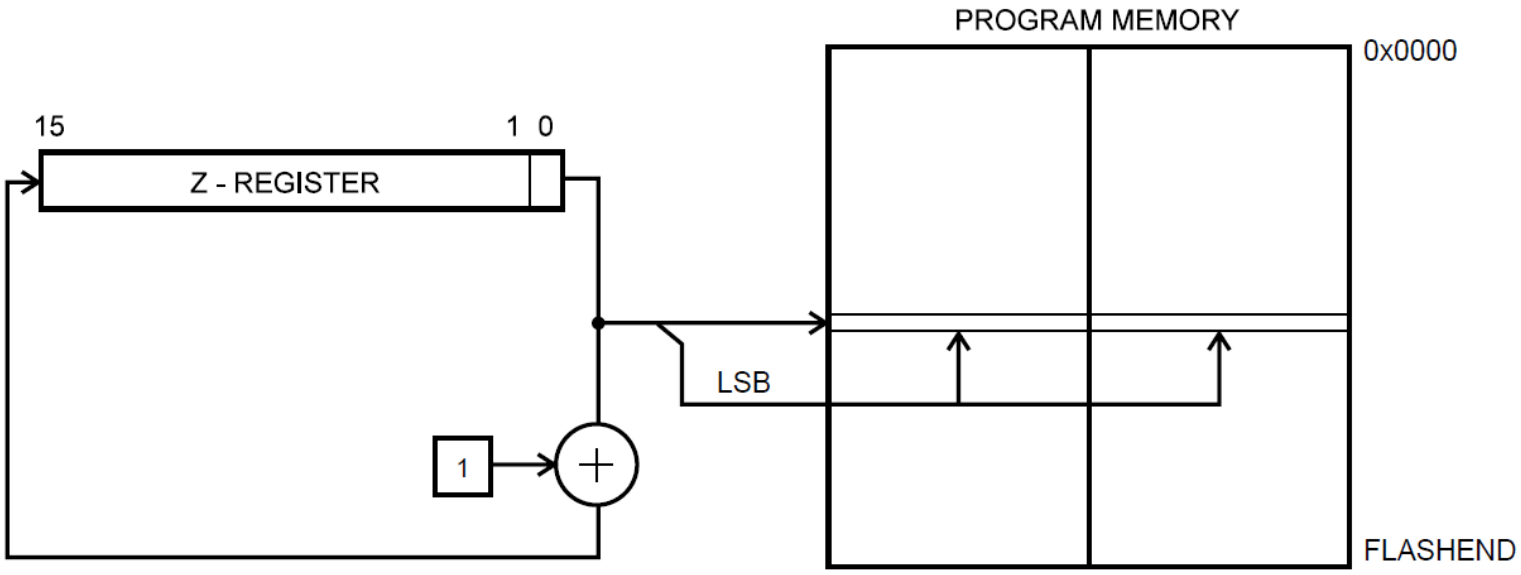
Example:

```
        ldi   ZH, high(Table_1<<1); Initialize Z-pointer
        ldi   ZL, low(Table_1<<1)
        lpm   r16, Z              ; Load constant from Program
                                  ; Memory pointed to by Z (r31:r30)
    ...
    Table_1:
    .dw 0x5876                    ; 0x76 is addresses when Z_LSB = 0
                                  ; 0x58 is addresses when Z_LSB = 1
    ...
```

Program memory with Post-increment using the LPM Z+ and ELPM Z+ Instruction.

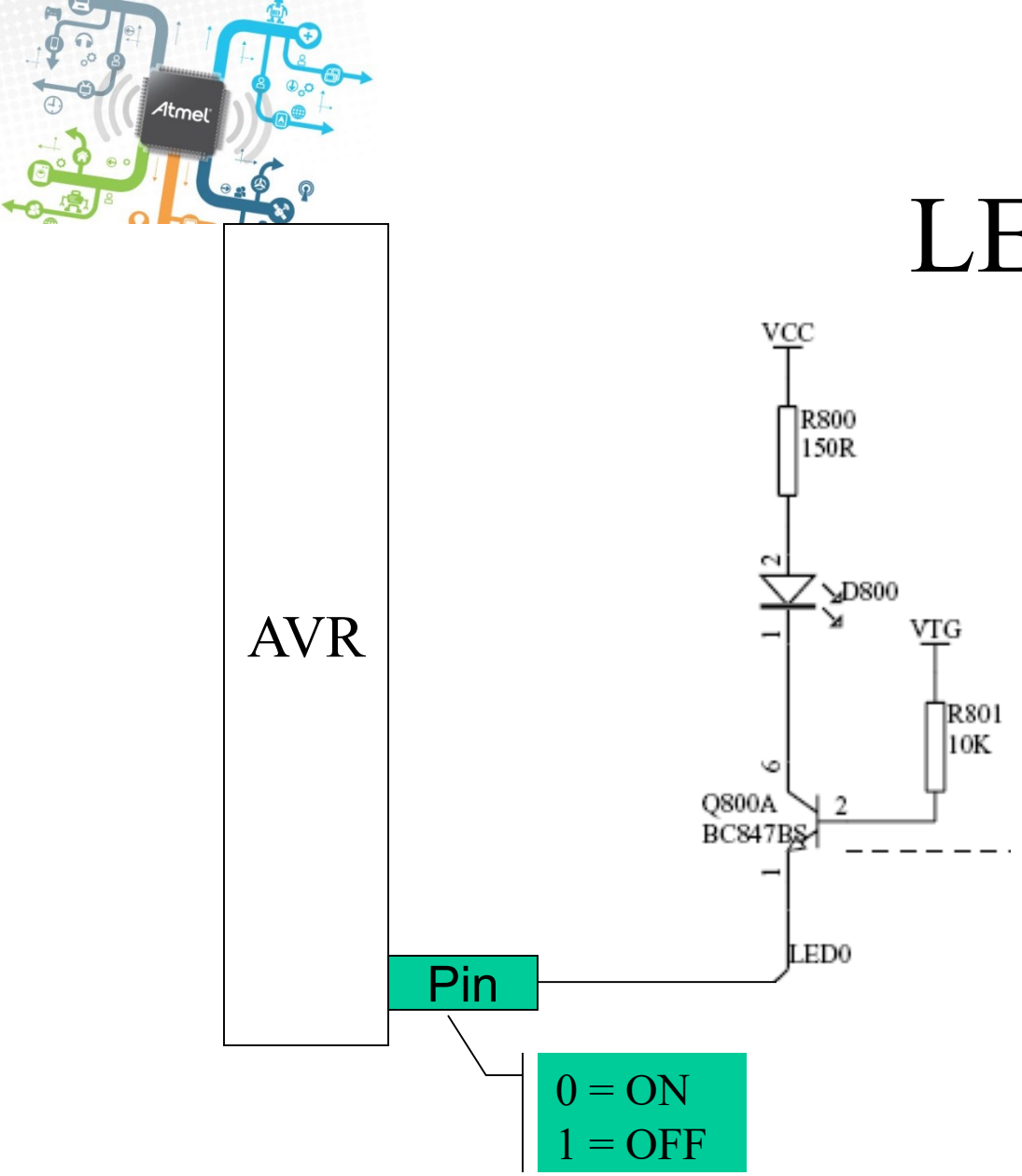
Program Memory with Post-increment using the LPM Z+ and ELPM Z+ Instruction

Figure 10. Program Memory Addressing with Post-increment



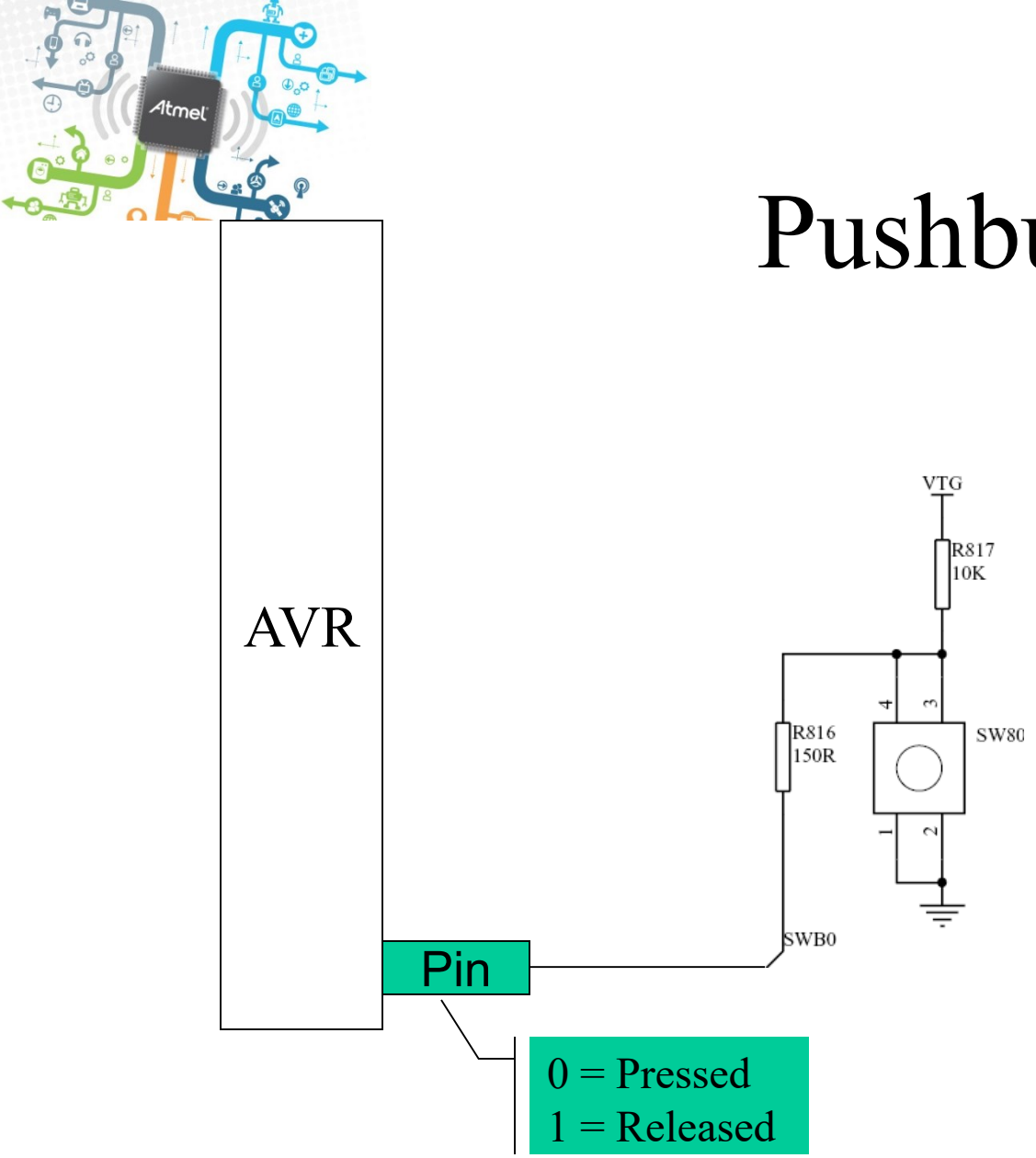
	Syntax:	Operands:	Program Counter:
(i)	LPM	None, R0 implied	PC ← PC + 1
(ii)	LPM Rd, Z	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	LPM Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1

LED



- An output pin is connected to the LED via a jumper
- There is an external pull-up
  - A transistor is used to provide constant brightness via Vcc regardless of Vtarget level (down to 1.8V)

Pushbutton



- Pressing the button pulls the pin to GND
  - The pin reads 0
- Releasing the button allows the 10K resistor to pull the signal to Vtarget
  - The pin reads 1