

HANDOUT #4

010113027

# MICROPROCESSORS & EMBEDDED COMPUTER SYSTEMS

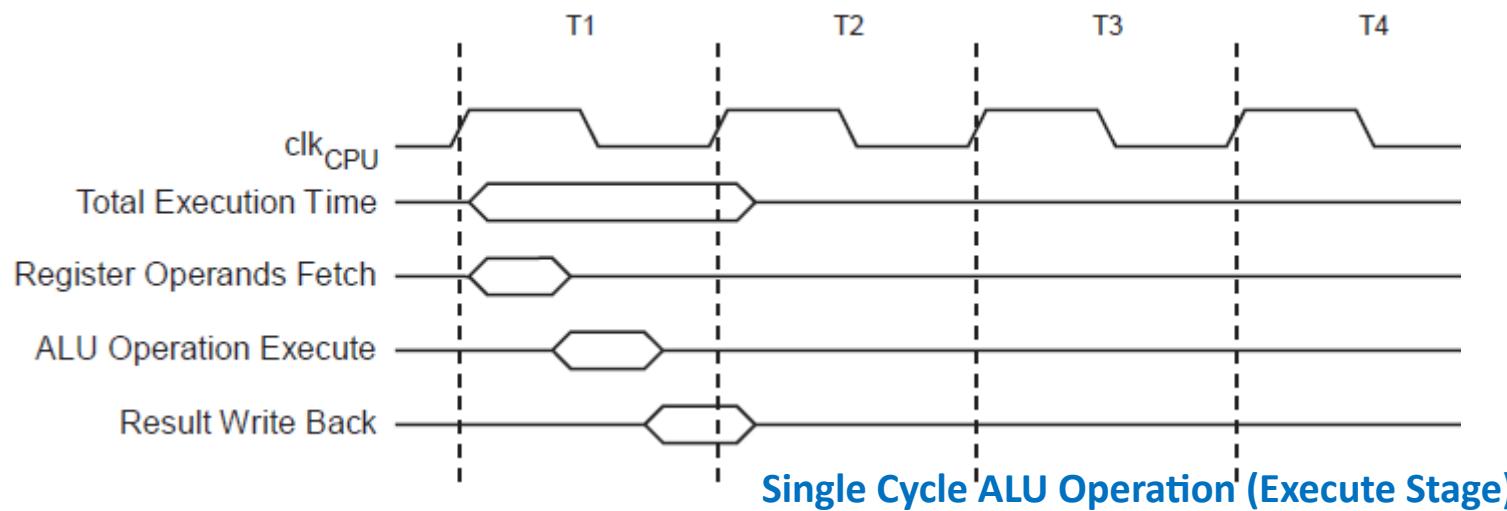
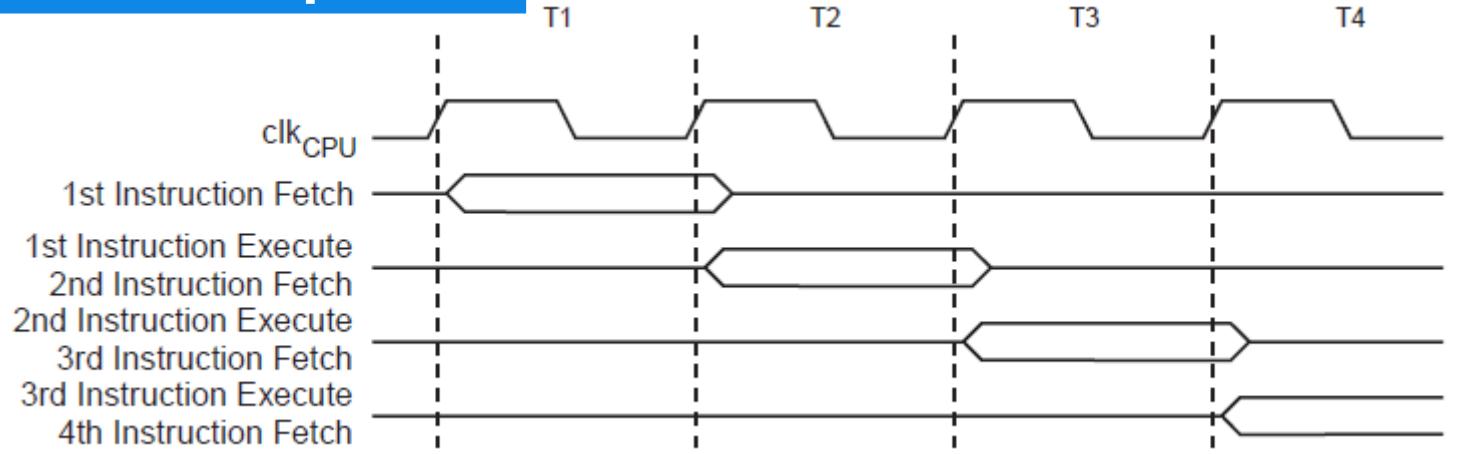
INSTRUCTOR: RSP ([rawat.s@eng.kmutnb.ac.th](mailto:rawat.s@eng.kmutnb.ac.th))

# Key Topics

- **Introduction to ATmega → ATmega328P**
- **AVR Memory Map**
- **AVR Registers and Memory Locations**
- **AVR Instruction Set**
- **AVR Memory Addressing Modes**
- **AVR Assembly Programming**
- **AVR Assembly Code Examples**
- **AVR Assembly Debugging with Microchip Studio IDE**
- **AVR Assemblers: AVR-GCC & AVRASM2**

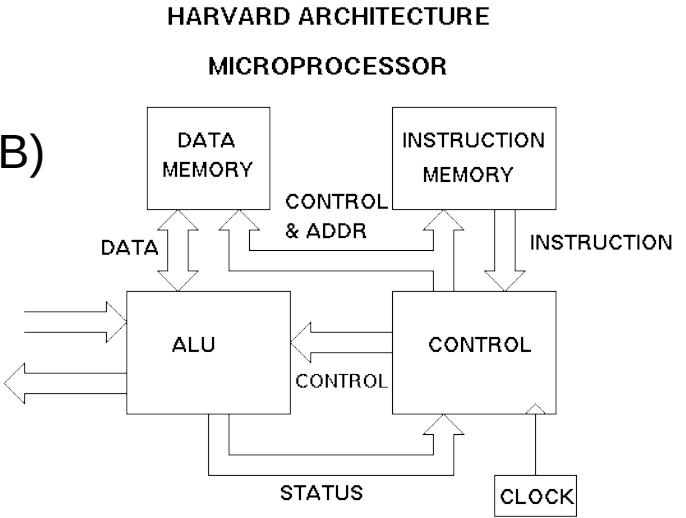
- AVR refers to an **8-bit microcontroller** architecture (originally from Atmel, now part of Microchip Technology).
- It uses a **modified Harvard architecture**: program memory (Flash) and data memory (SRAM & I/O) are separate.
- AVR is a **RISC-style microcontroller**: its instruction set is optimized so most instructions execute in 1 clock cycle (some in 2, a few more for memory or special operations).
- The ATmega architecture uses a **2-stage pipeline**, consisting of:
  - FETCH – Retrieve the next instruction from Flash.
  - EXECUTE – Execute the previously fetched instruction.

# AVR Instruction Pipeline



# ATmega328P

- **8-bit RISC CPU, Harvard Architecture**
- **SRAM (on-chip): 2048 (2 KB)**
- **Program Flash Memory (on-chip): 32768 Bytes (32 KB)**
- **EEPROM (on-chip): 1024 Bytes (1 KB)**
- **VCC range: 1.8V - 5.5V**
- **Max. CPU speed: 16MHz @ 4.5V - 5.5V**
- **ICSP (serial device programming), no JTAG pins**
- **Package Options**
  - 28-pin DIP (PDIP-28): Through-hole package
  - 32-pin TQFP (TQFP-32): Thin Quad Flat Package (0.8 mm pitch)
  - 32-pin QFN (QFN-32): Quad Flat No-lead Package



***Read the DATASHEET carefully, paying special attention to the registers, their addresses, and the function of each bit.***

# ATmega328 / 328P / 328PB

## ATmega328

- An older version, replaced by 328P
- Device signature: 0x1E 0x95 0x14

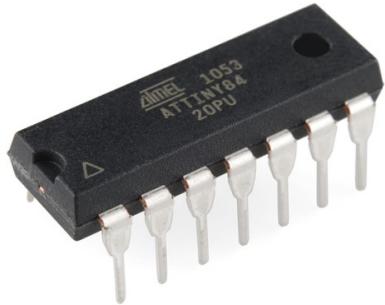
## ATmega328P (P = picoPower)

- Low-power optimized version of ATmega328 (power-saving features)
- Device signature: 0x1E 0x95 0x0F

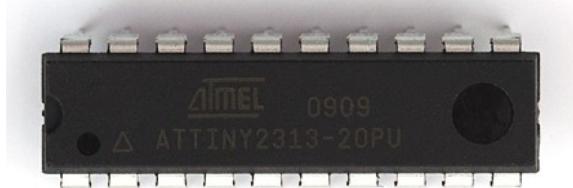
## ATmega328PB

- An enhanced model; a superset of ATmega328P
- Extra hardware and interfaces
  - e.g. two USARTs, two SPI modules and two TWI/I2C modules
- Device signature: 0x1E 0x95 0x16

# Examples of IC Packages



ATtiny84, PDIP-14,  
8KB Flash, 512B SRAM, 512B EEPROM,  
12 I/O lines, up to 20MHz



ATtiny2313, PDIP-20,  
2KB Flash, 128B SRAM, 128B EEPROM,  
15 I/O lines, up to 20MHz



ATmega328P, PDIP-28,  
32KB Flash, 2KB SRAM, 1KB EEPROM,  
23 I/O lines, up to 20MHz



ATmega328P-AU, TQFP-32,  
32KB Flash, 2KB SRAM, 1KB EEPROM,  
23 I/O lines, up to 20MHz

# ATmega328P: Pinout (PDIP-28)

<b>Active-low MCU reset</b>	(PCINT14/RESET) PC6	1	28	□ PC5 (ADC5/SCL/PCINT13)
	(PCINT16/RXD) PD0	2	27	□ PC4 (ADC4/SDA/PCINT12)
	(PCINT17/TXD) PD1	3	26	□ PC3 (ADC3/PCINT11)
	(PCINT18/INT0) PD2	4	25	□ PC2 (ADC2/PCINT10)
	(PCINT19/OC2B/INT1) PD3	5	24	□ PC1 (ADC1/PCINT9)
	(PCINT20/XCK/T0) PD4	6	23	□ PC0 (ADC0/PCINT8)
<b>VCC: Digital Supply Voltage (1.8V ~ 5.5V)</b>	VCC	7	22	□ GND <b>GND AVCC</b>
<b>GND: Ground</b>	GND	8	21	□ AREF <b>AREF: Analog Reference Voltage, usually AVCC</b>
	(PCINT6/XTAL1/TOSC1) PB6	9	20	□ AVCC <b>AVCC: Analog Supply Voltage</b>
	(PCINT7/XTAL2/TOSC2) PB7	10	19	□ PB5 (SCK/PCINT5)
	(PCINT21/OC0B/T1) PD5	11	18	□ PB4 (MISO/PCINT4)
	(PCINT22/OC0A/AIN0) PD6	12	17	□ PB3 (MOSI/OC2A/PCINT3)
	(PCINT23/AIN1) PD7	13	16	□ PB2 ( <u>SS</u> /OC1B/PCINT2)
	(PCINT0/CLKO/ICP1) PB0	14	15	□ PB1 (OC1A/PCINT1)

## Port B, Port C, Port D:

- General Purpose 8-Bit I/O Ports,
- optional internal pullup-resistors when configured as input
- output source capability: 20mA

## Special functions of ports are also available:

- Port D: UART (RXD,TXD), External Interrupts (INTx), Analog Comparator (AINx), Output Compare (OCx)
- Port B: External Oscillator/Crystal (XTALx, TOSCx), SPI (MISO, MOSI, SCK, /SS)
- Port C: A/D converters (ADCx), TWI (SCL, SDA)

# ATmega328P: System Functions

- Power-on Reset (POR)
- Brown-out Detection (BOR)
- Watchdog Timer (WDT)
- Interrupt Handling & Interrupt Vectors
- External Interrupts (INT0, INT1)
- Pin Change Interrupts (PCINT0–23 in 3 groups)
- Power Management and Sleep Modes:
  - Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
  - MCU wakeup sources: WDT, Pin Change Interrupts, Ext. Interrupts, ...
- Clock Sources and Clock Selection Unit
- Internal Calibrated Oscillator
- Fuse bits & Lock Bits

# ATmega328P: Peripherals

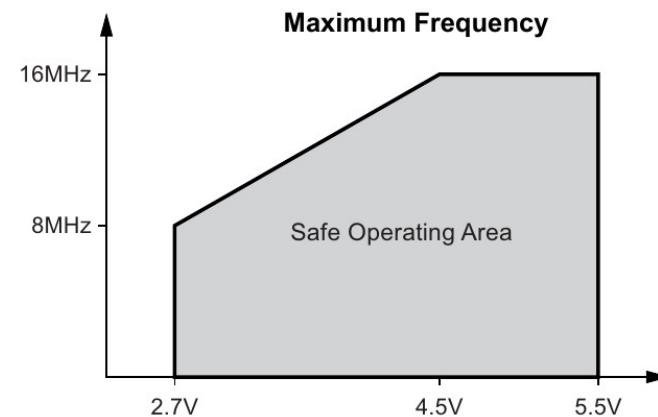
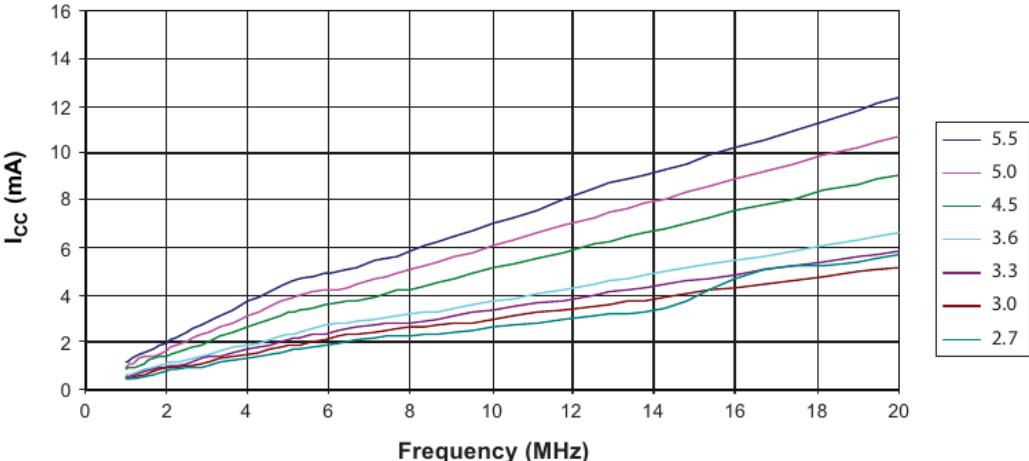
- I/O ports (GPIO): Port B, C, D
- 1x USART
- Timers/Counters
  - 2x 8-bit Timer/Counters (Timer0, Timer2)
  - 1x 16-bit Timer/Counter (Timer1)
  - 6x PWM outputs (OC0A, OC0B, OC1A, OC1B, OC2A, OC2B pins)
- 1x Master/Slave SPI (Serial Peripheral Interface)
- 1x TWI (Two-Wire Interface) / I2C (Inter-Chip Communication)
- 10-bit ADC: 6 channels (DIP) or 8 channels (TQFP/QFN)
- 1x Analog Comparator
- No DAC, I2S, CAN, USB, Ethernet, TRNG (True Random Number Generator), WiFi, BLE, ...

# ATmega328P Characteristics

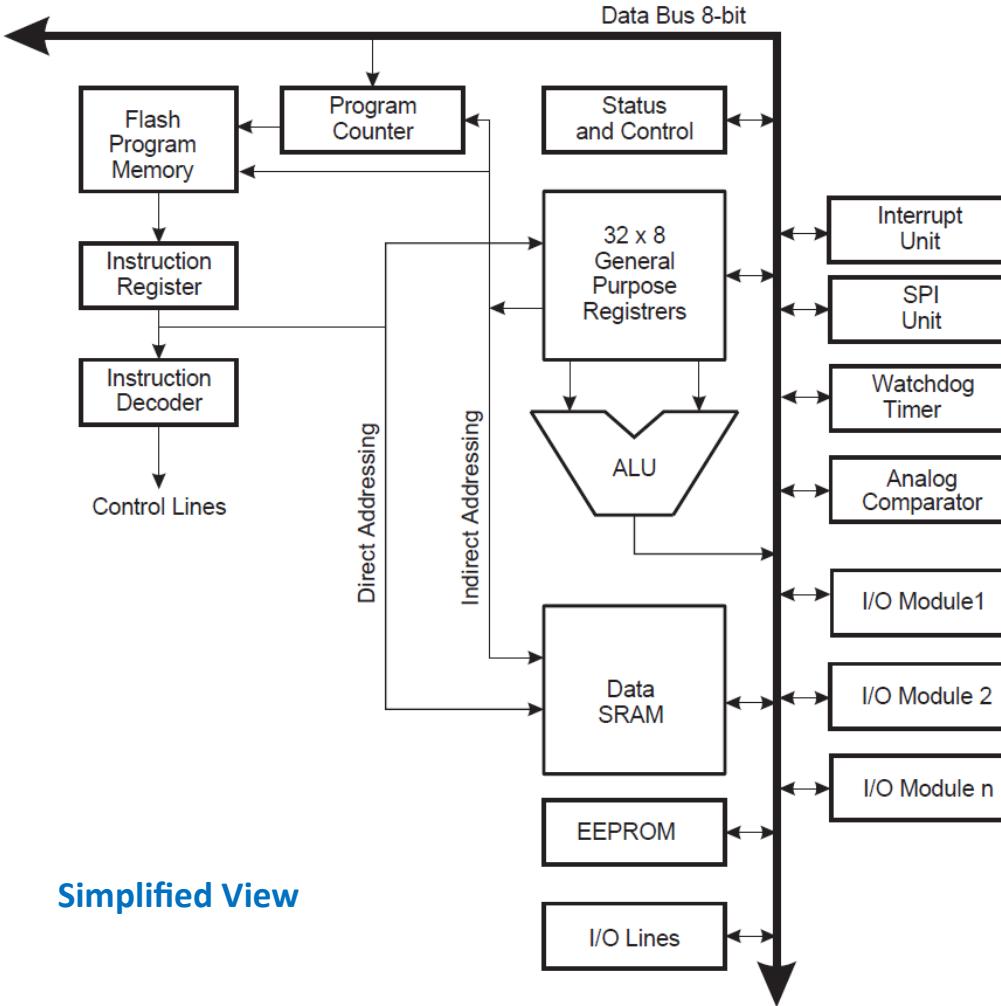
Some device characteristics of ATmega328P:

- **Speed Grade:**
  - up to 10MHz @ 2.7V - 4.5V
  - up to 16MHz (20MHz) @ 4.5V - 5.5V
- **Temperature range**
  - -40°C to +125°C (Automotive)
- **Current consumption:**
  - Active, 16MHz, VCC = 5V:  $I_{CC} = 14\text{mA}$  (typ.)
  - Idle, 16MHz, VCC = 5V:  $I_{CC} = 2.8\text{mA}$  (typ.)
- **Output voltage of I/O pins:**
  - VOL: 0.8V (max.) @VCC = 5V,  $I_{OL} = 20\text{mA}$  (sink)
  - VOH: 4.1V (max.) @VCC = 5V,  $I_{OH} = -20\text{mA}$  (source)

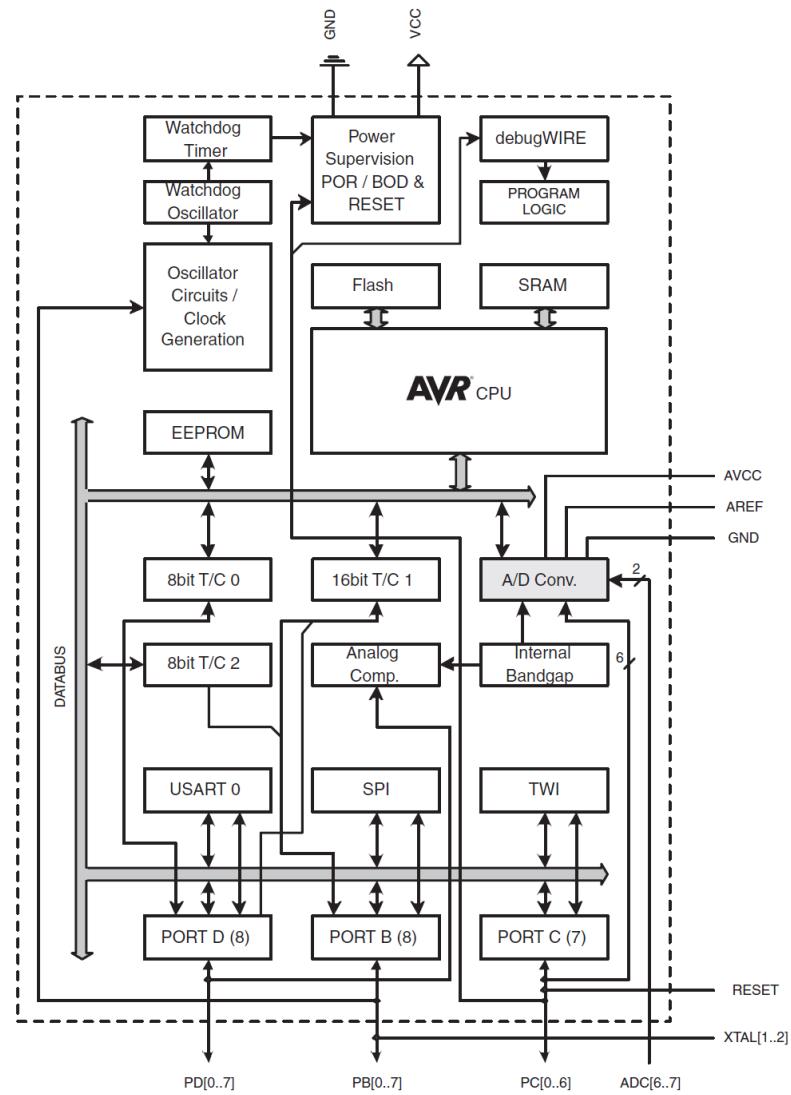
Active Supply Current versus Frequency



# ATmega328P Block Diagram

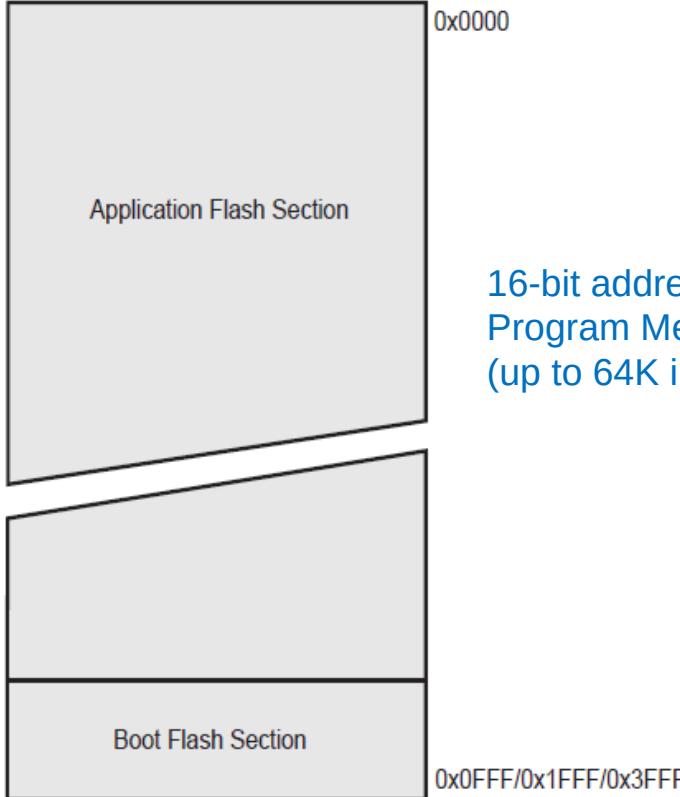


Simplified View



# AVR Memory Maps

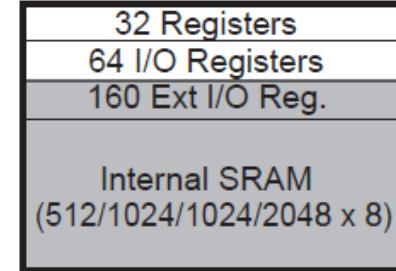
## Program Flash Memory



Accessible via special instructions: **LPM, SPM**

- Boot Loader support: Boot Flash Section
- The **SPM** instruction can be executed only from Boot Flash.

## Data Memory (byte-addressable)



0x0000 - 0x001F  
0x0020 - 0x005F  
0x0060 - 0x00FF  
0x0100

0x04FF/0x04FF/0x0FF/0x08FF

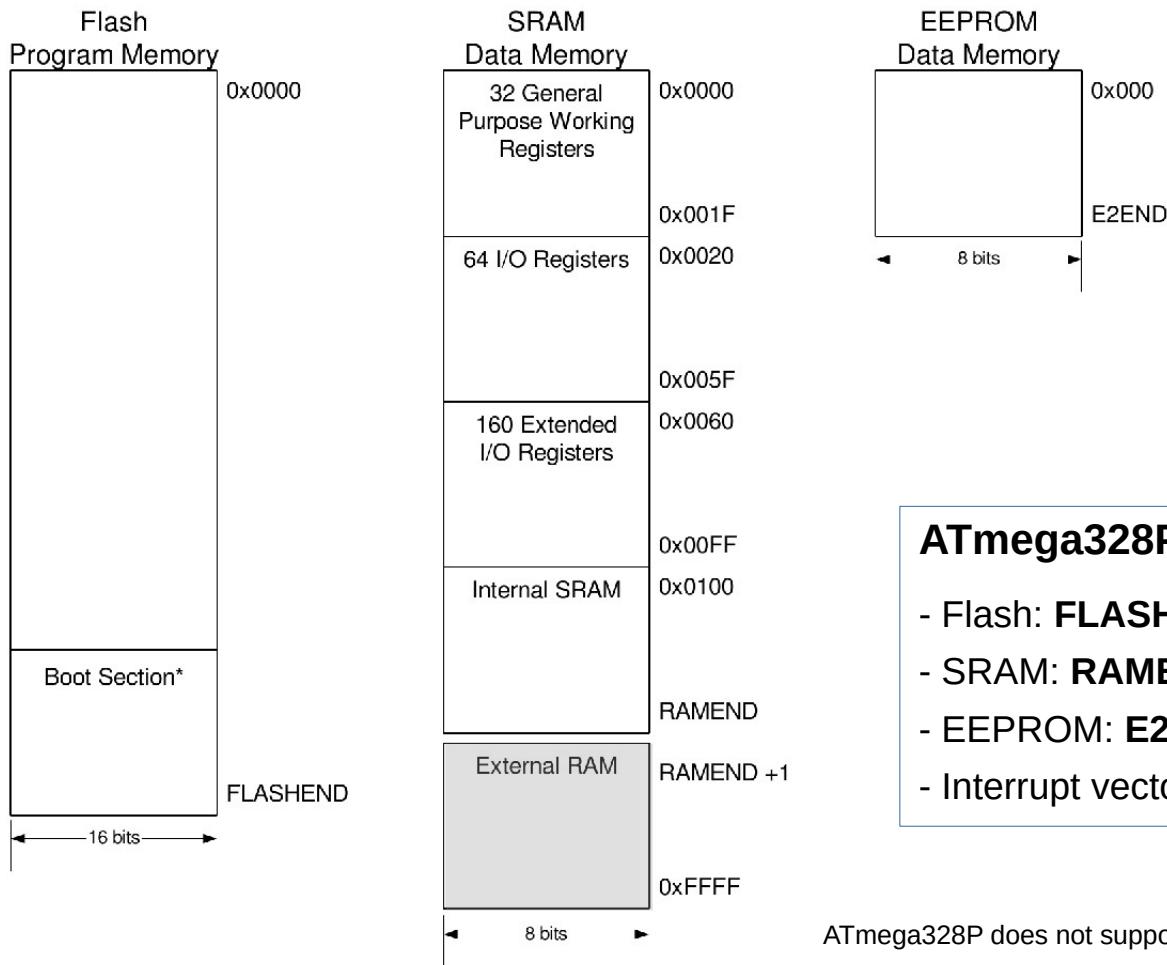
## Data Memory Map;

- 1) General-purpose working registers (GPRs)
- 2) I/O registers
- 3) Extended I/O registers
- 4) Internal SRAM
- 5) External SRAM (none for ATmega328P)

## AVR Memory Maps

- 1) Program Memory Map
- 2) Data Memory Map
- 3) EEPROM Memory Map

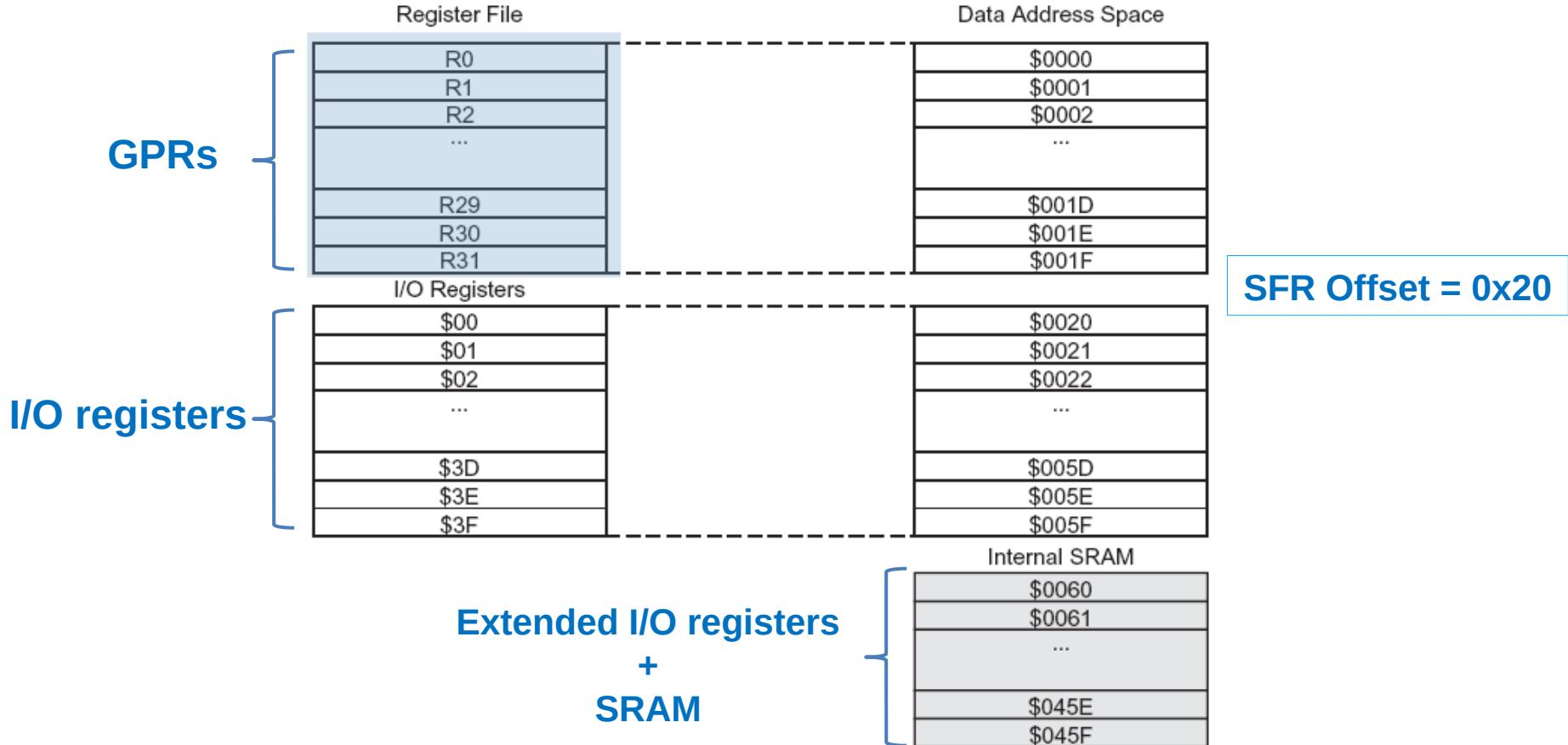
# AVR Memory Maps



## ATmega328P

- Flash: **FLASHEND**: 0x3FFF (32K bytes or 16K words)
- SRAM: **RAMEND**: 0x08FF (2K bytes)
- EEPROM: **E2END**: 0x3FF (1024 bytes)
- Interrupt vector size: 2 words (4 bytes)

# AVR Data Memory Map



# AVR General-Purpose Registers

- There are **32 8-bit general-purpose registers (R0–R31)**.
- The **last three register pairs** can be used as **pointer registers (16-bit)**: X = R27:R26, Y = R29:R28, Z = R31:R30
  - They can be used for **indirect memory addressing**, including **post-increment and pre-decrement addressing modes**.
  - Y and Z also support a **6-bit positive displacement (offset) addressing mode**.
- **ADIW** and **SBIW** work only on the X, Y, Z paired registers.
- Instructions that accept an **8-bit immediate value** are limited to registers **R16–R31** (e.g., **LDI**, **CPI**, **ORI**, **ANDI**, etc.).
- **16-bit immediate operations (ADIW, SBIW)** apply only to four register pairs **R25:R24**, **R27:R26**, **R29:R28**, and **R31:R30**.
- Some variants of the **multiply instructions** are restricted to registers **R16–R23**.
- The **R0–R31** registers, the status register and some I/O registers are **bit-addressable**.

# AVR General-Purpose Registers

7	0
R0	
R1	
R2	
...	
R13	
R14	
R15	
R16	
R17	
...	
R26	
R27	
R28	
R29	
R30	
R31	

Addr.

0x00

0x01

0x02

0x0D

0x0E

0x0F

0x10

0x11

0x1A

0x1B

0x1C

0x1D

0x1E

0x1F

The addresses of registers  
in the Data Memory Space

Three register pairs can be used  
for indirect memory addressing.

X = (R27:R26)

Y = (R29:R28)

Z = (R31:R30)

X-register Low Byte

X-register High Byte

Y-register Low Byte

Y-register High Byte

Z-register Low Byte

Z-register High Byte

Low(X) = R26

High(X) = R27

Low(Y) = R28

High(Y) = R29

Low(Z) = R30

High(Z) = R31

General-purpose working registers

17

# Examples of Special Registers

In addition to general-purpose registers, the CPU has a few **special-purpose registers**:

- **PC**: program counter
- **SP**: 16-bit stack pointer for ATmega series → **SPH** (high byte) : **SPL** (low byte)
- **SREG**: 8-bit status register

Note: The **PC width** depends on **FLASH size**:

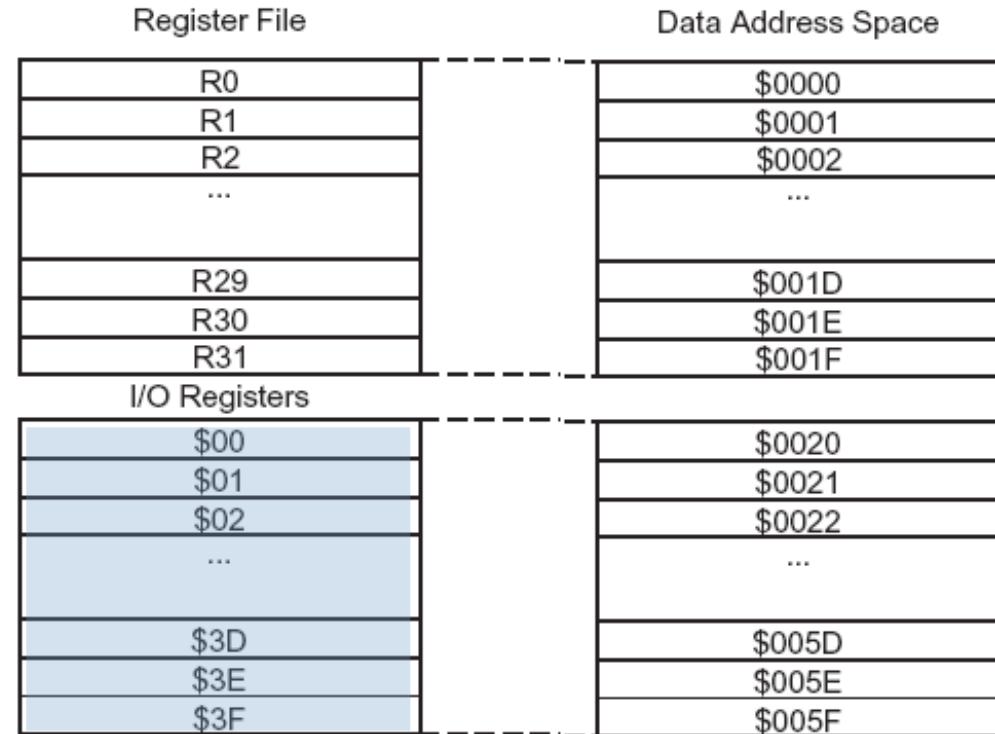
- ≤ 8 KB: 12 bits
- ≤ 64 KB: 16 bits
- > 64 KB: 22 bits (uses **RAMP** registers)

Only available in AVR MCUs with large address spaces.

- **RAMPX**, **RAMPY**, **RAMPZ**, **RAMPD** and **EIND** registers
- Used to form extended 24-bit addresses for program and data memory beyond 64 KB.

# Accessing I/O Registers

- The **first 64 I/O registers** are accessible through both the I/O space and the data address space.
- The **only first 32 I/O registers** are **bit-addressable**.
- They have therefore two different addresses, usually written as "0x00 (0x20)" through "0x3F (0x5F)", where the first item is the **I/O address** and the second, in parentheses, the **data address**.
- The **special-purpose CPU registers**, with the exception of **PC**, can be accessed as **I/O registers**.



# Status Register

## SREG – AVR Status Register

The AVR status register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

### Status Register (SREG)

SREG: Status Register

C: Carry Flag

Z: Zero Flag

N: Negative Flag

V: Two's complement overflow indicator

S:  $N \oplus V$ , For signed tests

H: Half Carry Flag

T: Transfer bit used by BLD and BST instructions

I: Global Interrupt Enable/Disable Flag

# Stack Pointer

## SPH and SPL – Stack Pointer High and Stack Pointer Low Register

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
0x3D (0x5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W								
	R/W								
Initial Value	RAMEND								
	RAMEND								

# Stack and Stack Pointer

- The stack is implemented using the **Data SRAM**, where data is stored such as:
  - the **return address** from the **Program Counter (PC)** when a **subroutine** is called
  - **local variables** used inside functions.
- The **Stack Pointer (SP) register** is used to point to the **top of the stack**.
  - At initialization, the **SP** must be set to the last address (the highest address) of the SRAM.
- Changes to the **SP** occur automatically when:
  - The **SP** decreases when executing instructions such as **PUSH, CALL, ICALL, RCALL**.
  - The **SP** increases when executing instructions such as **POP, RET** (return from subroutine), and **RETI** (return from interrupt).

Instruction	Stack pointer	Description
PUSH	Decrement by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decrement by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

# AVR Data Memory Access

All addresses in the Data Memory Map can be accessed using instructions for load/store data such as **LD/LDS/LDD** and **ST/STS/STD**.

32 Registers
64 I/O Registers
160 Ext I/O Reg.
Internal SRAM (512/1024/1024/2048 x 8)

0x0000 - 0x001F  
0x0020 - 0x005F  
0x0060 - 0x00FF  
0x0100  
  
0x04FF/0x04FF/0x0FF/0x08FF

These locations can be accessed via **SBI/SBIS** and **CBI/SBIC** instructions,

These locations can be accessed via **IN / OUT** instructions.

There are **5 addressing modes** for the Data Memory Map:

- Direct
- Indirect with Displacement
- Indirect
- Indirect with Pre-decrement
- Indirect with Post-increment

# AVR Data Memory Access

Instruction	Direction	Address Type
LD	Memory → Register	Indirect (via X, Y, Z pointer registers)
LDS	Memory → Register	Direct address (full 16-bit address)
LDD	Memory → Register	Indirect + displacement (Y or Z + offset)
ST	Register → Memory	Indirect (via X, Y, Z pointer registers)
STS	Register → Memory	Direct address (full 16-bit address)
STD	Register → Memory	Indirect + displacement (Y or Z + offset)



# AVR Data Memory Access

```
.INCLUDE "m328pdef.inc"
.DSEG                                ; Data Memory Section
.ORG 0x0100                            ; Set SRAM location for the variable
VAR1: .BYTE 1                           ; Reserve 1 byte in SRAM for VAR1

.CSEG                                ; Program Memory Section
.ORG 0x0000                            ; Program start address (Reset Vector)
RJMP MAIN                             ; The first instruction at 0x0000

MAIN:
    LDS R16, VAR1      ; R16 ← RAM[0x0100]
    INC R16            ; R16 ← R16 + 1
    STS VAR1, R16     ; RAM[0x0100] ← R16

    LDI ZH, HIGH(VAR1) ;
    LDI ZL, LOW(VAR1)  ;
    LD R16, Z          ; R16 ← RAM[Z]
    INC R16            ; R16 ← R16 + 1
    ST Z, R16          ; RAM[Z] ← R16

LOOP: RJMP LOOP
```

# AVR Memory Addressing

There are several ways to access data / memory / program memory:

- **Register-direct**: operations that act on registers (e.g. add, logical ops, shifts).
- **I/O direct**: instructions that read/write to I/O registers (e.g. IN, OUT) using small I/O address space.
- **Data direct / Data indirect / Data indirect with displacement**: load/store from RAM via **absolute address** or via **pointer registers** (X, Y, Z).
- **Program memory access**: There are instructions to read from program memory: e.g. LPM, ELPM (extended), possibly with post-increment (LPM Z+). These load constants or data stored in flash into registers.
- **Jump/Call instructions**: to transfer control within program memory, either relative (e.g. RJMP, RCALL) or absolute (e.g. JMP, CALL).

# I/O Space

I/O space contains control registers for all peripherals (GPIO, timers, ADC, UART, SPI, etc.).

## Lower I/O Registers

- Address range: **0x0020 – 0x005F**
- Size: 64 bytes
- Optimized for speed (**fast access**)
- Example:
  - **DDRB, PORTB, PINB**
- These can be accessed by instructions:
  - **IN / OUT** (fast, single-cycle)
  - **SBI / CBI** (bit set/clear, 2 cycles)
  - **LD / ST** (slower, 2 cycles)

## Extended I/O Registers

- Address range: **0x0060 – 0x00FF**
- Size: 160 bytes
- Extended I/O space holds the majority of **peripheral control registers (SFRs)**.
- These are accessed only by:
  - **LD / ST** (Load / store from SRAM, indirect addressing)
  - **LDS / STS** (Load / store from SRAM, absolute addressing)
  - No IN / OUT instructions

# Examples of I/O Registers

- **PORTD, DDRD, PIND**
  - These are the three registers associated with **I/O Port D**.
- **PORTC, DDRC, PINC**
  - These are the three registers associated with **I/O Port C**.
- **PORTB, DDRB, PINB**
  - These are the three registers associated with **I/O Port B**.
- **SP (Stack Pointer)**
  - The Stack Pointer is a 16-bit register, consisting of **SPH:SPL**, and is used to control stack operations.
- **SREG (Status Register)**
  - The Status Register contains the flag bits—such as **Carry (C)**, **Zero (Z)**, **Negative (N)**, and others—that reflect the result of ALU operations executed by each instruction.

- The instruction set contains ~131 instructions ([AVRe+ for ATmega328](#)), which can be **grouped** as follows:
  - 1) Arithmetic and Logic Instructions
  - 2) Control flow (Branches, Jumps, Calls) Instructions
  - 3) Data Transfer, Memory & I/O Instructions
  - 4) Bit, Bit-test and Bit Shift Instructions
- Most instructions are **16 bits (2 bytes) in size**, although some instructions are 32 bits.
- Most instructions execute in a **single clock cycle**. However, some instructions require **two cycles**, such as:
  - Instructions that access bytes in SRAM memory
  - 8×8-bit multiply instructions
  - Instructions related to stack operations
- Some instructions take **four cycles**, such as **CALL** and **RET**, which are used for function (subroutine) calls.

# Carry Flag: Unsigned Addition

Addition of two 8-bit unsigned numbers  
=> **Carry Bit is 1.**

$$\begin{array}{r} 11111111 \text{ (+255)} \\ + 00000001 \text{ (+1)} \\ \hline 100000000 \Rightarrow C=1 \end{array}$$

$$\begin{array}{r} 01000000 \text{ (+64)} \\ + 11000000 \text{ (+192)} \\ \hline 100000000 \Rightarrow C=1 \end{array}$$

$$\begin{array}{r} 11111111 \text{ (+255)} \\ + 11111111 \text{ (+255)} \\ \hline 111111110 \Rightarrow C=1 \end{array}$$

Addition of two 8-bit unsigned numbers  
=> **Carry Bit is 0.**

$$\begin{array}{r} 01111111 \text{ (+127)} \\ + 01000000 \text{ (+64)} \\ \hline 010111111 \Rightarrow C=0 \end{array}$$

$$\begin{array}{r} 01111111 \text{ (+127)} \\ + 01111111 \text{ (+127)} \\ \hline 011111110 \Rightarrow C=0 \end{array}$$

The CPU has a status register (**SREG**) which holds flags that many instructions update.

**Carry flag** in the SREG is set if an arithmetic operation resulted in a carry (or borrow) out.

# Carry Flag: Unsigned Subtraction

$$\begin{array}{r} 00000000 \text{ (0)} \\ - 00000001 \text{ (+1)} \\ \hline \end{array}$$

**111111111 => C=1 (Borrow Flag)**

$$\begin{array}{r} 00000000 \text{ (0)} \\ - 11111111 \text{ (+255)} \\ \hline \end{array}$$

**100000001 => C=1 (Borrow Flag)**

$$\begin{array}{r} 10000000 \text{ (+128)} \\ - 01111111 \text{ (+127)} \\ \hline \end{array}$$

**00000001 => C=0 (No Borrow Flag)**

# 16-bit Unsigned Addition / Subtraction

A=3101h, B=00FFh, X=A+B

$$\text{LOW}(A) = \text{AL} = 01h$$

$$\text{HIGH}(A) = \text{AH} = 31h$$

$$\text{LOW}(B) = \text{BL} = FFh$$

$$\text{HIGH}(B) = \text{BH} = 00h$$

A=3101h, B=00FFh, X=A-B

$$\text{LOW}(A) = \text{AL} = 01h$$

$$\text{HIGH}(A) = \text{AH} = 31h$$

$$\text{LOW}(B) = \text{BL} = FFh$$

$$\text{HIGH}(B) = \text{BH} = 00h$$

$$XL = \text{AL} + \text{BL}$$

$$= 01h + FFh = 00h \rightarrow C=1$$

$$XH = \text{AH} + \text{BH} + C \quad \leftarrow$$

$$= 31h + 00h + 1$$

$$= 32h \rightarrow C=0$$

$$X = 3200h$$

Add with carry

$$XL = \text{AL} - \text{BL}$$

$$= 01h - FFh = 02h \rightarrow C=1$$

$$XH = \text{AH} - \text{BH} - C \quad \leftarrow$$

$$= 31h - 00h - 1$$

$$= 30h \rightarrow C=0$$

$$X = 3002h$$

Subtract with carry (borrow)

# Overflow Flag: 2's Complement (Signed) Numbers

## Signed Overflow (V=1)

Positive + Positive => Negative  
Negative + Negative => Positive

Positive - Negative => Negative  
Negative - Positive => Positive

## Signed Comparison Flag (S)

$$S = N \oplus V$$

## Overflow (V=1)

$$\begin{array}{r} 01111111 \text{ (+127 dec)} \\ + 00000001 \text{ (+1 dec)} \\ \hline 010000000 \text{ (-128)} \end{array} \Rightarrow C=0, V=1, N=1, S=0, Z=0$$

## No Overflow (V=0)

$$\begin{array}{r} 10000000 \text{ (-128 dec)} \\ + 00000001 \text{ (+1 dec)} \\ \hline 010000001 \text{ (-127)} \end{array} \Rightarrow C=0, V=0, N=1, S=1, Z=0$$

## Signed Overflow (V):

$$A = A_7 \dots A_0 \text{ and } B = B_7 \dots B_0 \text{ (MSB..LSB)}$$

$$\text{Add (R=A+B): } V = \sim(A_7 \oplus B_7) \cdot (A_7 \oplus R_7)$$

$$\text{Sub (R=A-B): } V = (A_7 \oplus B_7) \cdot (A_7 \oplus R_7)$$

## Carry / Borrow Flag (C)

$$\text{Add (R=A+B): } C = A_7 \cdot B_7 + (A_7 \oplus B_7) \cdot \sim R_7$$

$$\text{Sub (R=A-B): } C = \sim A_7 \cdot B_7 + \sim (A_7 \oplus B_7) \cdot R_7$$

# Overflow Flag: 2's Complement (Signed) Numbers

## Signed Overflow (V):

$A = A_7 \dots A_0$  and  $B=B_7 \dots B_0$  (MSB..LSB)

Add:  $V = \sim(A_7 \oplus B_7) \cdot (A_7 \oplus R_7)$

Sub:  $V = (A_7 \oplus B_7) \cdot (A_7 \oplus R_7)$

## Carry / Borrow Flag (C)

Add:  $C = A_7 \cdot B_7 + (A_7 \oplus B_7) \cdot \sim R_7$

Sub:  $C = \sim A_7 \cdot B_7 + \sim (A_7 \oplus B_7) \cdot R_7$

## Signed Comparison Flag (S)

$S = N \oplus V$

$10000000$  (-128 dec)

+  $10000000$  (-128 dec)

-----

$100000000$  (0)  $\Rightarrow C=1, V=1, N=0, S=1, Z=1$

$00000001$  (+1 dec)

+  $11111111$  (-1 dec)

-----

$100000000$  (0)  $\Rightarrow C=1, V=0, N=0, S=0, Z=1$

$01000000$  (64 dec)

-  $01000001$  (65 dec)

-----

$11111111$  (-1 dec)  $\Rightarrow N=1, V=0, S=1, Z=0$

# Definitions of Symbols of the AVR ISA

<b>Rd</b>	Destination (and source) register: R0..R31 or R16..R31 (depending on instructions)
<b>Rr</b>	Source register: R0..R31
<b>K</b>	constant data (8-bit) between 0..255
<b>k</b>	constant (program memory location)
<b>b</b>	bit in register or I/O register: (3-bit) constant between 0..7
<b>s</b>	bit in the status register (SREG): (3-bit) constant between 0..7
<b>X,Y,Z</b>	indirect address register
<b>P</b>	I/O register address: (6-bit) constant between 0..63
<b>q</b>	displacement for direct addressing: (6-bit) constant between 0..63
<b>STACK</b>	stack pointer register
<b>PC</b>	program counter

# Examples of AVR ALU Instructions

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
ADD	Rd, Rr	Add without Carry	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	Rd, K	Add Immediate to Word	$Rd+1:Rd \leftarrow Rd+1:Rd + K$	Z,C,N,V	2
SUB	Rd, Rr	Subtract without Carry	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Immediate	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Immediate with Carry	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW	Rd, K	Subtract Immediate from Word	$Rd+1:Rd \leftarrow Rd+1:Rd - K$	Z,C,N,V	2

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
AND	Rd, Rr	Logical AND	$Rd \leftarrow Rd \bullet Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND with Immediate	$Rd \leftarrow Rd \bullet K$	Z,N,V	1
OR	Rd, Rr	Logical OR	$Rd \leftarrow Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR with Immediate	$Rd \leftarrow Rd \vee K$	Z,N,V	1
EOR	Rd, Rr	Exclusive OR	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1

Examples of AVR instruction with the 8-bit immediate of 8 bits:

**ADIW, SUBI, SBCI, SBIW, ANDI, ORI, CPI, LDI** (Rd must be from R16..R31).

# Examples of AVR ALU Instructions

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
COM	Rd	One's Complement	$Rd \leftarrow \$FF - Rd$	Z,C,N,V	1
NEG	Rd	Two's Complement	$Rd \leftarrow \$00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd \vee K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd \bullet (\$FFh - K)$	Z,N,V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \bullet Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd \leftarrow \$FF$	None	1
MUL	Rd,Rr	Multiply Unsigned	$R1, R0 \leftarrow Rd \times Rr$	C	2 <sup>(1)</sup>

# Examples of AVR Instructions for Data Transfer

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
MOV	Rd, Rr	Copy Register	$Rd \leftarrow Rr$	None	1
LDI	Rd, K	Load Immediate	$Rd \leftarrow K$	None	1
LDS	Rd, k	Load Direct from SRAM	$Rd \leftarrow (k)$	None	3
LD	Rd, X	Load Indirect	$Rd \leftarrow (X)$	None	2
LD	Rd, X+	Load Indirect and Post-Increment	$Rd \leftarrow (X), X \leftarrow X + 1$	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	$X \leftarrow X - 1, Rd \leftarrow (X)$	None	2
LD	Rd, Y	Load Indirect	$Rd \leftarrow (Y)$	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	$Rd \leftarrow (Y), Y \leftarrow Y + 1$	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	$Y \leftarrow Y - 1, Rd \leftarrow (Y)$	None	2
LDD	Rd, Y+q	Load Indirect with Displacement	$Rd \leftarrow (Y + q)$	None	2
LD	Rd, Z	Load Indirect	$Rd \leftarrow (Z)$	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	$Rd \leftarrow (Z), Z \leftarrow Z + 1$	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	$Z \leftarrow Z - 1, Rd \leftarrow (Z)$	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	$Rd \leftarrow (Z + q)$	None	2

# Examples of AVR Instructions for Data Transfer

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
STS	k, Rr	Store Direct to SRAM	$(k) \leftarrow Rr$	None	3
ST	X, Rr	Store Indirect	$(X) \leftarrow Rr$	None	2
ST	X+, Rr	Store Indirect and Post-Increment	$(X) \leftarrow Rr, X \leftarrow X + 1$	None	2
ST	-X, Rr	Store Indirect and Pre-Decrement	$X \leftarrow X - 1, (X) \leftarrow Rr$	None	2
ST	Y, Rr	Store Indirect	$(Y) \leftarrow Rr$	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	$(Y) \leftarrow Rr, Y \leftarrow Y + 1$	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	$Y \leftarrow Y - 1, (Y) \leftarrow Rr$	None	2
STD	Y+q,Rr	Store Indirect with Displacement	$(Y + q) \leftarrow Rr$	None	2
ST	Z, Rr	Store Indirect	$(Z) \leftarrow Rr$	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	$(Z) \leftarrow Rr, Z \leftarrow Z + 1$	None	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	$Z \leftarrow Z - 1, (Z) \leftarrow Rr$	None	2
STD	Z+q,Rr	Store Indirect with Displacement	$(Z + q) \leftarrow Rr$	None	2

# Examples of AVR Instructions for Data Transfer

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
LPM		Load Program Memory	$R0 \leftarrow (Z)$	None	3
IN	Rd, P	In Port	$Rd \leftarrow P$	None	1
OUT	P, Rr	Out Port	$P \leftarrow Rr$	None	1
PUSH	Rr	Push Register on Stack	$STACK \leftarrow Rr$	None	2
POP	Rd	Pop Register from Stack	$Rd \leftarrow STACK$	None	2

The IN and OUT instructions can only be used with the I/O registers. P must be an I/O address.

Examples of address definitions for I/O registers (PORT B) In "m328Pdef.inc"

```
.equ PORTB = 0x05
.equ DDRB = 0x04
.equ PINB = 0x03
```

# Examples of AVR Instructions for Bit Operations

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
LSL	Rd	Logical Shift Left	$Rd(n+1) \leftarrow Rd(n), Rd(0) \leftarrow 0, C \leftarrow Rd(7)$	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	$Rd(n) \leftarrow Rd(n+1), Rd(7) \leftarrow 0, C \leftarrow Rd(0)$	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	$Rd(0) \leftarrow C, Rd(n+1) \leftarrow Rd(n), C \leftarrow Rd(7)$	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	$Rd(7) \leftarrow C, Rd(n) \leftarrow Rd(n+1), C \leftarrow Rd(0)$	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	$Rd(n) \leftarrow Rd(n+1), n=0..6$	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	$Rd(3..0) \leftrightarrow Rd(7..4)$	None	1
BSET	s	Flag Set	$SREG(s) \leftarrow 1$	SREG(s)	1
BCLR	s	Flag Clear	$SREG(s) \leftarrow 0$	SREG(s)	1
SBI	P, b	Set Bit in I/O Register	$I/O(P, b) \leftarrow 1$	None	2
CBI	P, b	Clear Bit in I/O Register	$I/O(P, b) \leftarrow 0$	None	2

# Examples of AVR Instructions for Bit Operations

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
BST	Rr, b	Bit Store from Register to T	$T \leftarrow Rr(b)$	T	1
BLD	Rd, b	Bit load from T to Register	$Rd(b) \leftarrow T$	None	1
SEC		Set Carry	$C \leftarrow 1$	C	1
CLC		Clear Carry	$C \leftarrow 0$	C	1
SEN		Set Negative Flag	$N \leftarrow 1$	N	1
CLN		Clear Negative Flag	$N \leftarrow 0$	N	1
SEZ		Set Zero Flag	$Z \leftarrow 1$	Z	1
CLZ		Clear Zero Flag	$Z \leftarrow 0$	Z	1
SEI		Global Interrupt Enable	$I \leftarrow 1$	I	1
CLI		Global Interrupt Disable	$I \leftarrow 0$	I	1

# Examples of AVR Instructions for Bit Operations

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
SES		Set Signed Test Flag	$S \leftarrow 1$	S	1
CLS		Clear Signed Test Flag	$S \leftarrow 0$	S	1
SEV		Set Two's Complement Overflow	$V \leftarrow 1$	V	1
CLV		Clear Two's Complement Overflow	$V \leftarrow 0$	V	1
SET		Set T in SREG	$T \leftarrow 1$	T	1
CLT		Clear T in SREG	$T \leftarrow 0$	T	1
SEH		Set Half Carry Flag in SREG	$H \leftarrow 1$	H	1
CLH		Clear Half Carry Flag in SREG	$H \leftarrow 0$	H	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1

# Examples of AVR Branch Instructions

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
RJMP	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
IJMP		Indirect Jump to (Z)	$PC \leftarrow Z$	None	2
JMP	k	Jump	$PC \leftarrow k$	None	3
RCALL	k	Relative Call Subroutine	$PC \leftarrow PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC \leftarrow Z$	None	3
CALL	k	Call Subroutine	$PC \leftarrow k$	None	4
RET		Subroutine Return	$PC \leftarrow STACK$	None	4
RETI		Interrupt Return	$PC \leftarrow STACK$	I	4

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
CPSE	Rd,Rr	Compare, Skip if Equal	if ( $Rd = Rr$ ) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	$Rd - Rr$	Z,C,N,V,H	1
CPC	Rd,Rr	Compare with Carry	$Rd - Rr - C$	Z,C,N,V,H	1
CPI	Rd,K	Compare with Immediate	$Rd - K$	Z,C,N,V,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if ( $Rr(b)=0$ ) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register Set	if ( $Rr(b)=1$ ) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if ( $I/O(P,b)=0$ ) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register Set	if ( $I/O(P,b)=1$ ) $PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3

# Examples of AVR Branch Instructions

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
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

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
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, 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

# Examples of AVR Branch Instructions

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
BRTS	k	Branch if T Flag Set	if ( $T = 1$ ) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if ( $T = 0$ ) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if ( $V = 1$ ) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if ( $V = 0$ ) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if ( $I = 1$ ) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if ( $I = 0$ ) then $PC \leftarrow PC + k + 1$	None	1 / 2

# AVR Instruction: ADC (Add with Carry)

## ADC – Add with Carry

---

### Description:

Adds two registers and the contents of the C Flag and places the result in the destination register Rd.

#### Operation:

(i)  $Rd \leftarrow Rd + Rr + C$

#### Syntax:

(i)  $ADC\ Rd,Rr$

#### Operands:

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

**Any register from R0..R31**

#### 16-bit Opcode:

0001	11rd	dddd	rrrr
------	------	------	------

**16-bit instruction size**

#### Status Register (SREG) Boolean Formula:

**These flags in SREG may be changed after executing this instruction.**



# AVR Instruction: ADC (Add with Carry)

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

Set if there was a carry from bit 3; cleared otherwise

Half-Carry

$$H = A_3 \cdot B_3 + \overline{R_3} \cdot (A_3 \oplus B_3)$$

$$H = A_3 \cdot B_3 \downarrow \overline{R_3} \cdot (A_3 + B_3)$$

S:  $N \oplus V$ , For signed tests.

$$V = \overline{(A_7 \oplus B_7)} \cdot (A_7 \oplus R_7)$$

$$V = (A_7 \cdot B_7 \cdot \overline{R_7}) + (\overline{A_7} \cdot \overline{B_7} \cdot R_7)$$

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

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

N: R7

N=1 if the MSB of R is 1.

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

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

Z=1 if all bits of R are 0.

Set if the result is \$00; cleared otherwise.

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

Carry

Set if there was carry from the MSB of the result; cleared otherwise.

$$C = A_7 \cdot B_7 + \overline{R_7} \cdot (A_7 \oplus B_7)$$

$$C = A_7 \cdot B_7 + \overline{R_7} \cdot (A_7 + B_7)$$

R (Result) equals Rd after the operation.

Example:

; Add R1:R0 to R3:R2

add r2,r0 ; Add low byte

adc r3,r1 ; Add with carry high byte

Words: 1 (2 bytes)

Cycles: 1

# AVR Instruction: Subtract with Carry

## SBC – Subtract with Carry

---

### Description:

Subtracts two registers and subtracts with the C Flag and places the result in the destination register Rd.

#### Operation:

(i)  $Rd \leftarrow Rd - Rr - C$

#### Syntax:

(i) SBC Rd,Rr

#### Operands:

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

#### Program Counter:

$PC \leftarrow PC + 1$

#### 16-bit Opcode:

0000	10rd	dddd	rrrr
------	------	------	------

### Status Register and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

# AVR Instruction: Subtract with Carry

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

Set if there was a borrow from bit 3; cleared otherwise

$$H = \overline{A_3} \cdot B_3 + R_3 \cdot (\overline{A_3} + B_3)$$

$$H = \overline{A_3} \cdot B_3 + R_3 \cdot (\overline{A_3} \oplus B_3)$$

S:  $N \oplus V$ , For signed tests.

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

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

$$V = (A_7 \oplus B_7) \cdot (A_7 \oplus R_7)$$

$$V = (A_7 \cdot \overline{B_7} \cdot \overline{R_7}) + (\overline{A_7} \cdot B_7 \cdot R_7)$$

N: R7

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

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

Previous value remains unchanged when the result is zero; cleared otherwise.

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

Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of the Rd; cleared otherwise.

R (Result) equals Rd after the operation.

$$C = \overline{A_7} \cdot B_7 + R_7 \cdot (\overline{A_7} \oplus B_7)$$

$$C = \overline{A_7} \cdot B_7 + R_7 \cdot (\overline{A_7} + B_7)$$

**Example:**

sub r2,r0	; Subtract r1:r0 from r3:r2
sbc r3,r1	; Subtract low byte
	; Subtract with carry high byte

**Words:** 1 (2 bytes)

**Cycles:** 1

# AVR Instruction: Subtract Immediate with Carry

## SBCI – Subtract Immediate with Carry

---

### Description:

Subtracts a constant from a register and subtracts with the C Flag and places the result in the destination register Rd.

### Operation:

(i)  $Rd \leftarrow Rd - K - C$

### Syntax:

(i) SBCI Rd,K

### Operands:

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

### Program Counter:

$PC \leftarrow PC + 1$

### 16-bit Opcode:

0100	KKKK	dddd	KKKK
------	------	------	------

### Status Register and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

# AVR Instruction: Subtract Immediate with Carry

- H:  $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$   
Set if there was a borrow from bit 3; cleared otherwise.
- 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; cleared otherwise.
- N: R7  
Set if MSB of the result is set; cleared otherwise.
- Z:  $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$   
Previous value remains unchanged when the result is zero; cleared otherwise.
- C:  $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$   
Set if the absolute value of the constant plus previous carry is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

## Example:

```
subi r16,$23      ; Subtract $4F23 from r17:r16
sbc r17,$4F       ; Subtract low byte
sbc r17,$4F       ; Subtract with carry high byte
```

Words: 1 (2 bytes)

Cycles: 1

# AVR Instruction: Load Immediate

## LDI – Load Immediate

### Description:

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

#### Operation:

(i)  $Rd \leftarrow K$

#### Syntax:

(i) LDI Rd,K

#### Operands:

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

#### Program Counter:

$PC \leftarrow PC + 1$

#### 16-bit Opcode:

1110	KKKK	dddd	KKKK
------	------	------	------

### Status Register (SREG) and Boolean Formula:

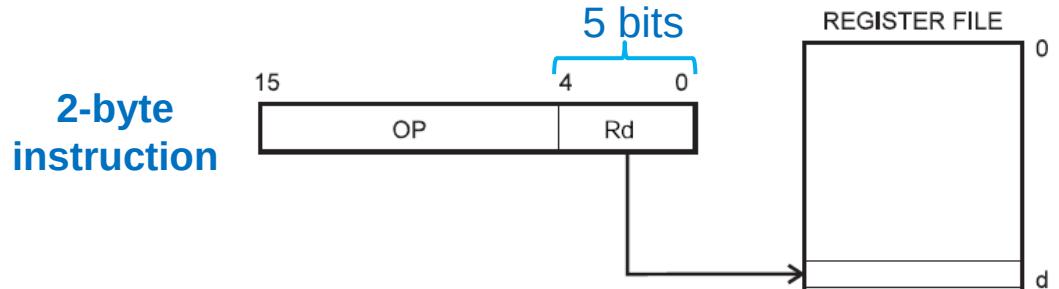
I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Words: 1 (2 bytes)

Cycles: 1

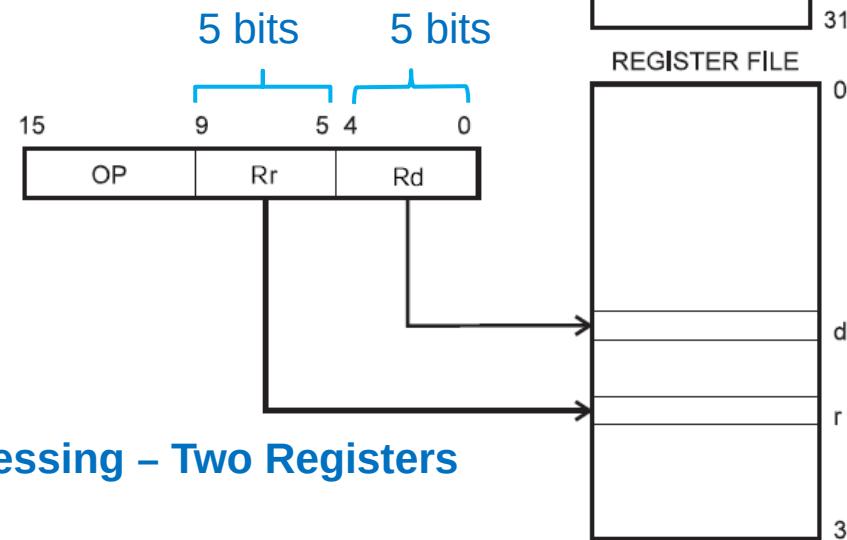
No impact on any flags in SREG

# AVR Data Memory Addressing



No SRAM or I/O memory is accessed; Instructions use only registers as operands and no immediate.

## Direct Register Addressing – One Register



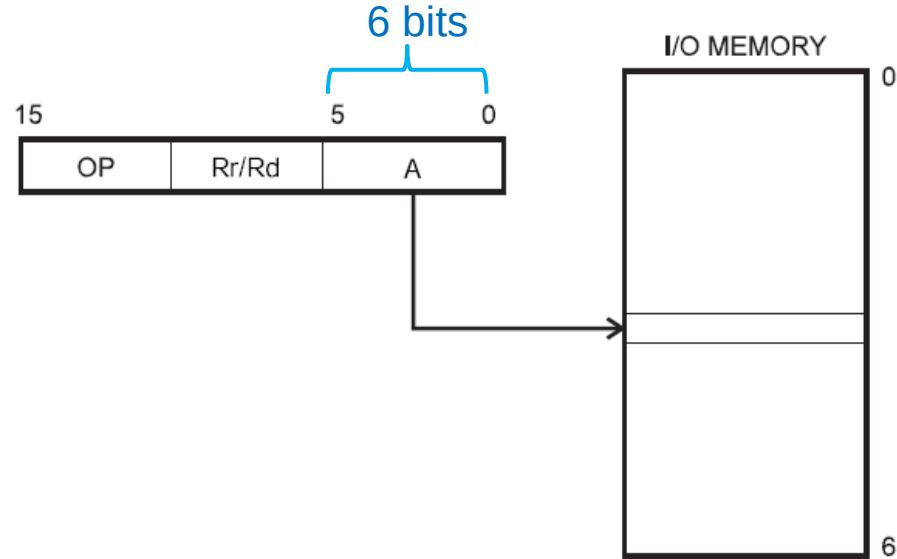
## Direct Register Addressing – Two Registers

# AVR Data Memory Addressing

<b>INC Rd</b>	; Increment: $Rd \leftarrow Rd + 1$
<b>DEC Rd</b>	; Decrement: $Rd \leftarrow Rd - 1$
<b>NEG Rd</b>	; Two's complement (negate): $Rd \leftarrow 0x00 - Rd$
<b>COM Rd</b>	; One's complement (invert): $Rd \leftarrow \text{NOT } Rd$ (or $0xFF - Rd$ )
<b>LSR Rd</b>	; Logical shift right: ; $Rd(6..0) \leftarrow Rd(7..1)$ , $Rd(7) \leftarrow 0$ , $C \leftarrow Rd(0)$
<b>ADD Rd, Rr</b>	; Add: $Rd \leftarrow Rd + Rr$
<b>ADC Rd Rr</b>	; Add with Carry: $Rd \leftarrow Rd + Rr + C$
<b>SUB Rd, Rr</b>	; Subtract: $Rd \leftarrow Rd - Rr$
<b>SUBI Rd, K</b>	; Subtract Immediate: $Rd \leftarrow Rd - K$
<b>AND Rd, Rr</b>	; Bitwise AND: $Rd \leftarrow Rd \text{ AND } Rr$
<b>OR Rd, Rr</b>	; Bitwise OR : $Rd \leftarrow Rd \text{ OR } Rr$
<b>EOR Rd, Rr</b>	; Bitwise XOR: $Rd \leftarrow Rd \text{ XOR } Rr$
<b>MOV Rd, Rr</b>	; Copy: $Rd \leftarrow Rr$

# AVR Data Memory Addressing

## I/O Direct Addressing



```
IN Rd, A ; Load the value from I/O register A into CPU register Rd
```

```
OUT A, Rr ; Store the value from CPU register Rr into I/O register A
```

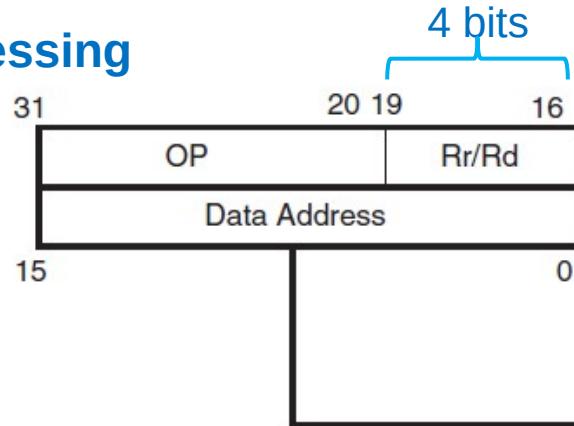
Rd / Rr = CPU register (R0–R31)

A = I/O register address (0–63 for classic I/O space)

# AVR Data Memory Addressing

## Direct Data Addressing

4-byte  
instruction



Data Space

0x0000

RAMEND

Only single registers can be loaded/stored per instruction ( $Rd/Rr = R0-R31$ ). The memory address is embedded directly in the instruction.

**LDS Rd, k ; Load Direct  $Rd \leftarrow RAM[k]$**

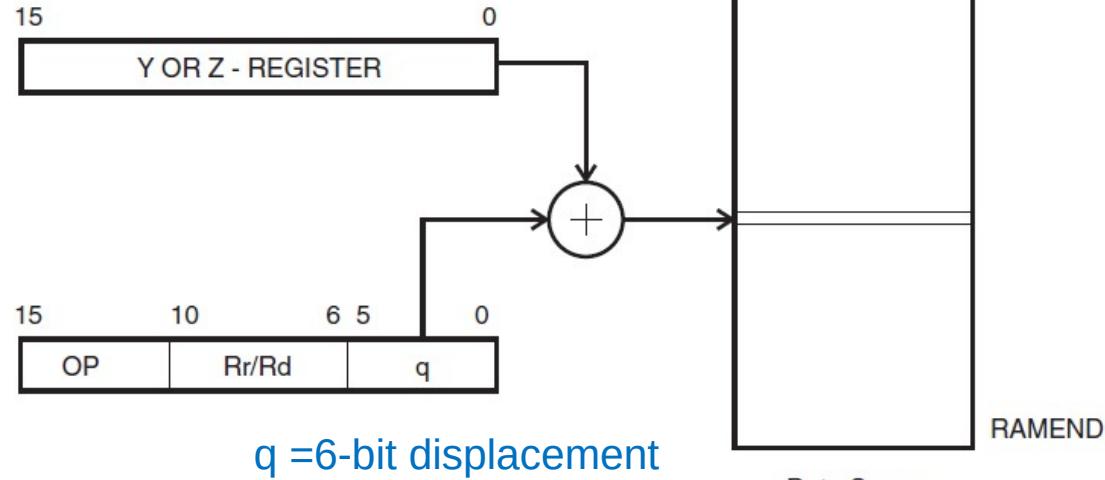
**STS k, Rr ; Store Direct  $RAM[k] \leftarrow Rr$**

**k** is a 16-bit constant SRAM address (data address).

# AVR Data Memory Addressing

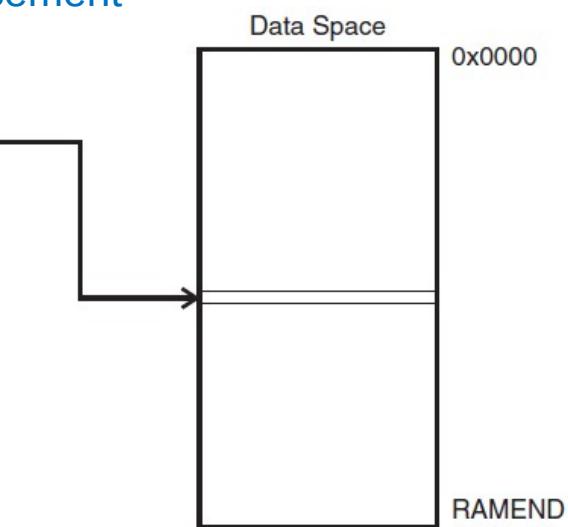
## Data Indirect with Displacement

2-byte instruction



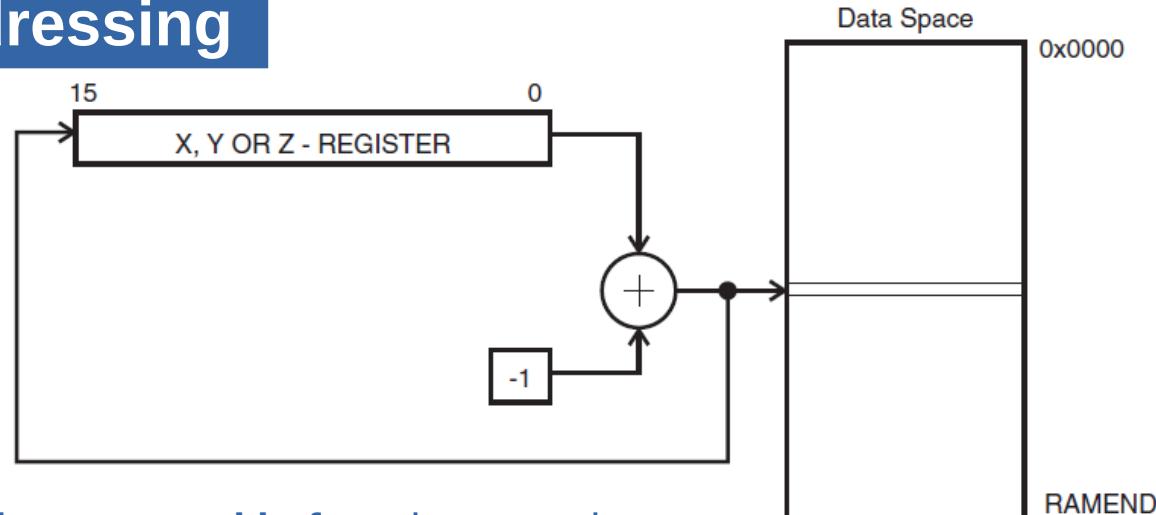
## Data Indirect Addressing

Register pairs are used as a pointer (with optional pre / post increment). This address mode is useful for accessing arrays in SRAM.



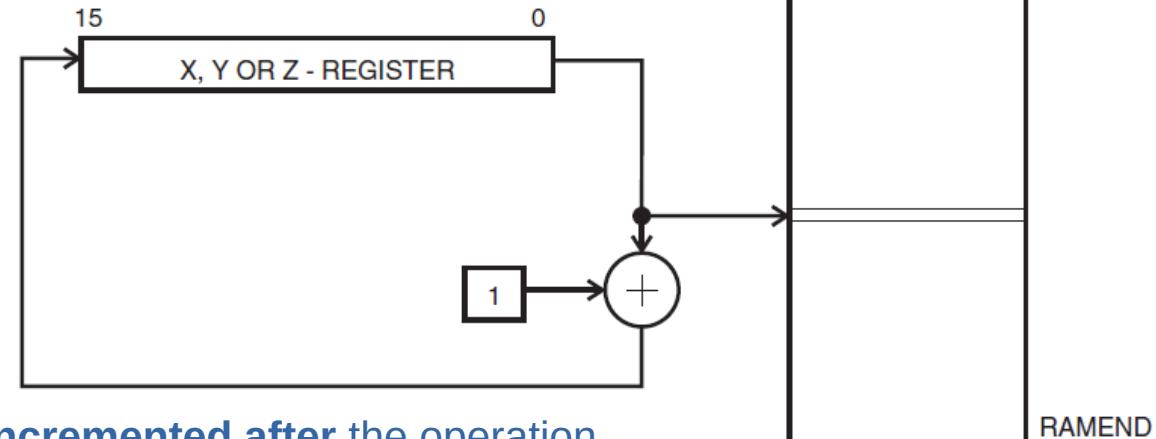
# AVR Data Memory Addressing

## Data Indirect Addressing with Pre-decrement



The X-, Y-, or the Z-register is **decremented before** the operation.

## Data Indirect Addressing with Post-increment



The X-, Y-, or the Z-register is **incremented after** the operation.

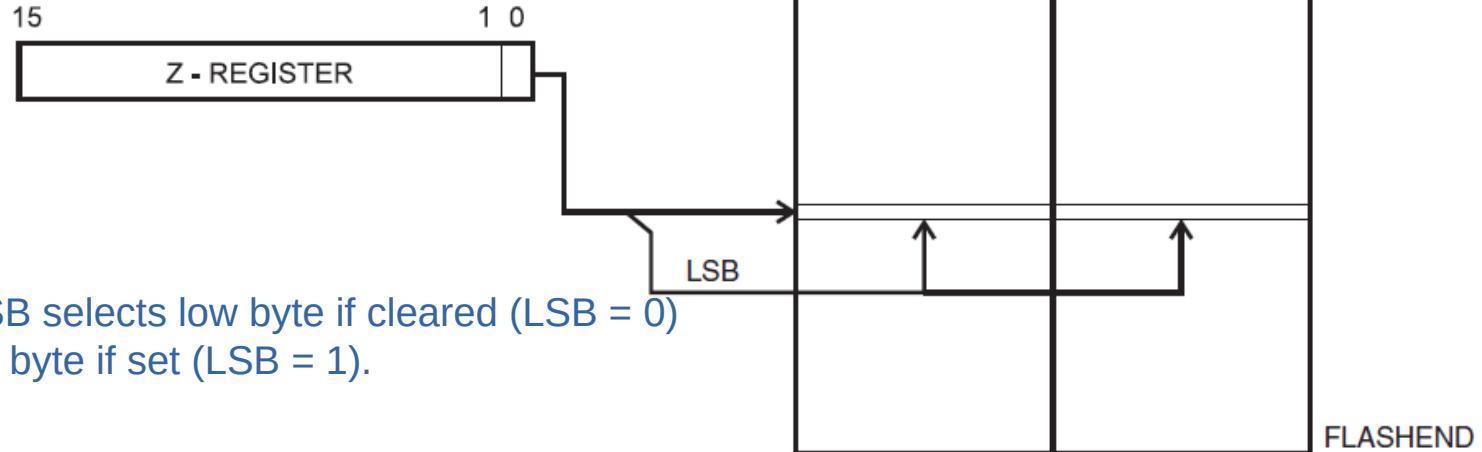
# AVR Data Memory Addressing

```
LDD Rd, Y+q ; Load Indirect with Displacement Rd from SRAM: Rd ← MEM[Y+q]
LDD Rd, Z+q ; Load Indirect with Displacement Rd from SRAM: Rd ← MEM[Z+q]
STD Y+q, Rr ; Store Indirect Rr into SRAM: MEM[Y + q] ← Rr
STD Z+q, Rr ; Store Indirect Rr into SRAM: MEM[Z + q] ← Rr
LD Rd, X ; Load Indirect Rd from SRAM: Rd ← MEM[X]
LD Rd, Y ; Load Indirect Rd from SRAM: Rd ← MEM[Y]
LD Rd, Z ; Load Indirect Rd from SRAM: Rd ← MEM[Z]
ST X, Rr ; Store Indirect Rr into SRAM: MEM[X] ← Rr
ST Y, Rr ; Store Indirect Rr into SRAM: MEM[Y] ← Rr
ST Z, Rr ; Store Indirect Rr into SRAM: MEM[Z] ← Rr
LD Rd, X+ ; Load Indirect with post-increment: Rd ← MEM[X], X ← X+1
ST X+, Rr ; Store Indirect with post-increment: MEM[X] ← Rr, X ← X+1
LD Rd, -X ; Load Indirect with pre-decrement: X ← X-1, Rd ← MEM[X]
ST -X, Rr ; Store Indirect with pre-decrement: X ← X-1, MEM[X] ← Rr
```

# AVR Program Memory Addressing

## Load / Store Program Memory Constant Addressing

### LPM and SPM Instructions

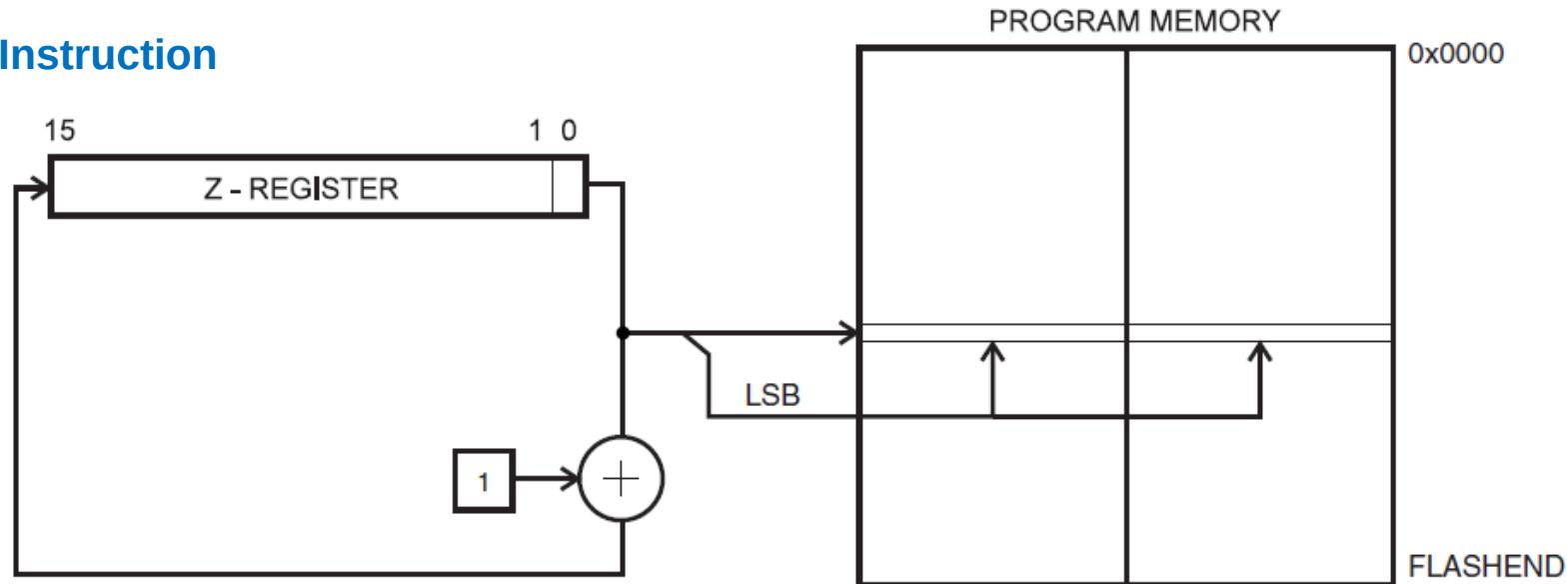


**ELPM / EELPM instructions** are available for devices with Flash size larger than 64 KB, like XMEGA and **not used on ATmega328P**.

# AVR Program Memory Addressing

## Load Program Memory with Post-increment

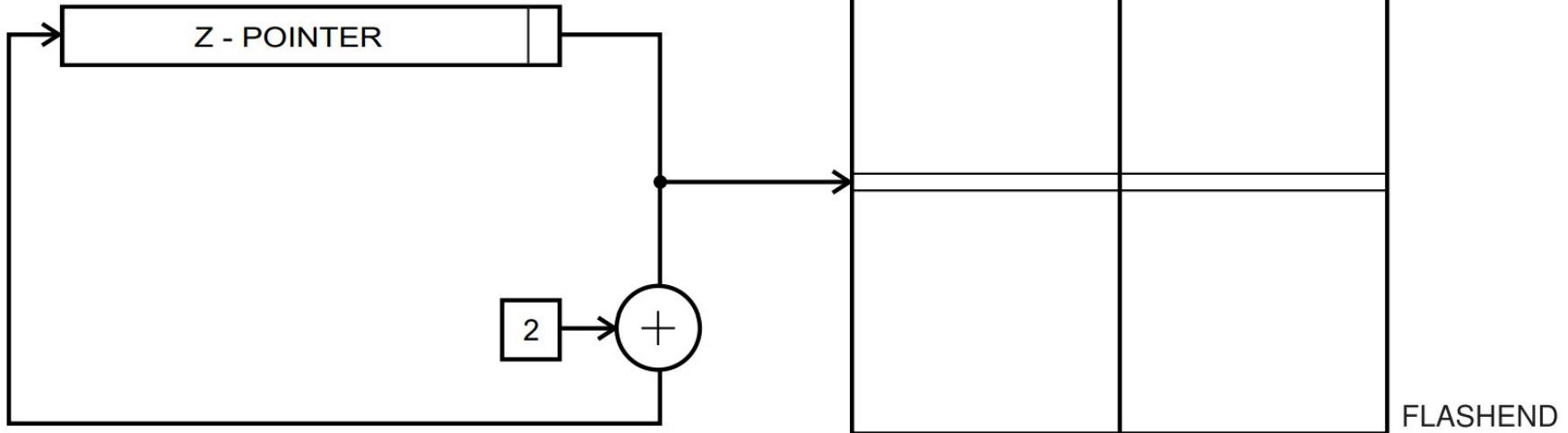
### LPM Z+ Instruction



# AVR Program Memory Addressing

## Store Program Memory with Post-increment

### SPM Z+ Instruction

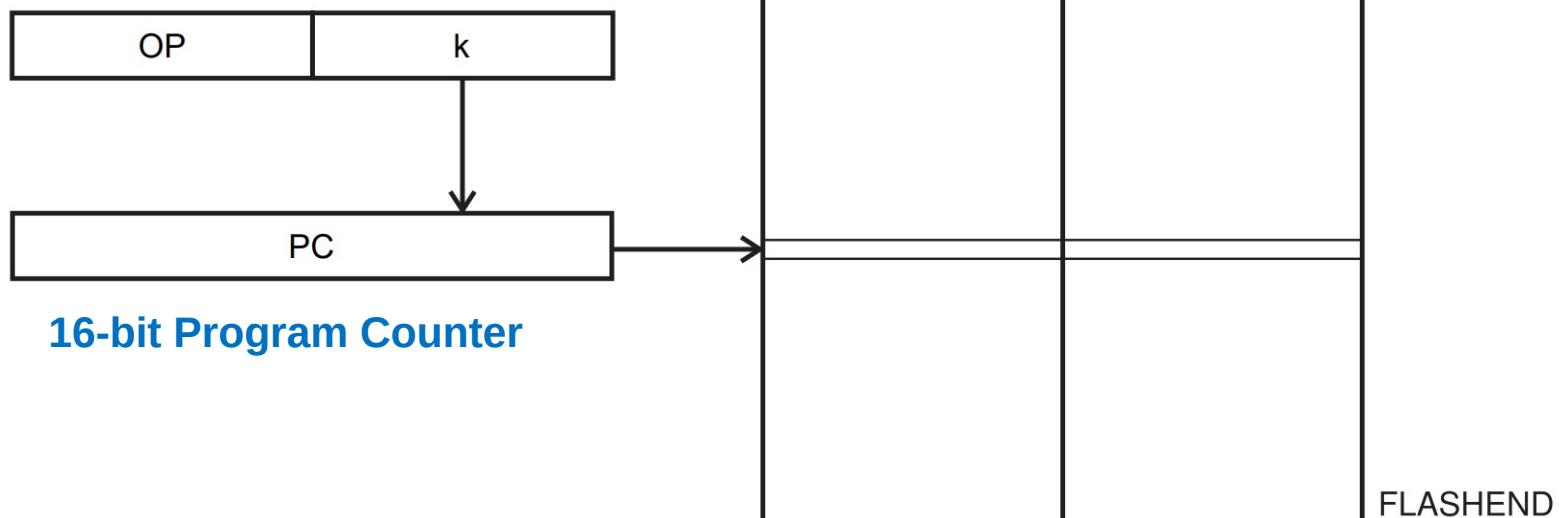


The Z-pointer is incremented by 2 after the operation. Constant byte address is specified by the Z-pointer contents before incrementing. The 15 MSbs select word address and the LSb should be left cleared.

# AVR Program Memory Addressing

## Direct Program Memory Addressing

### JMP and CALL instructions

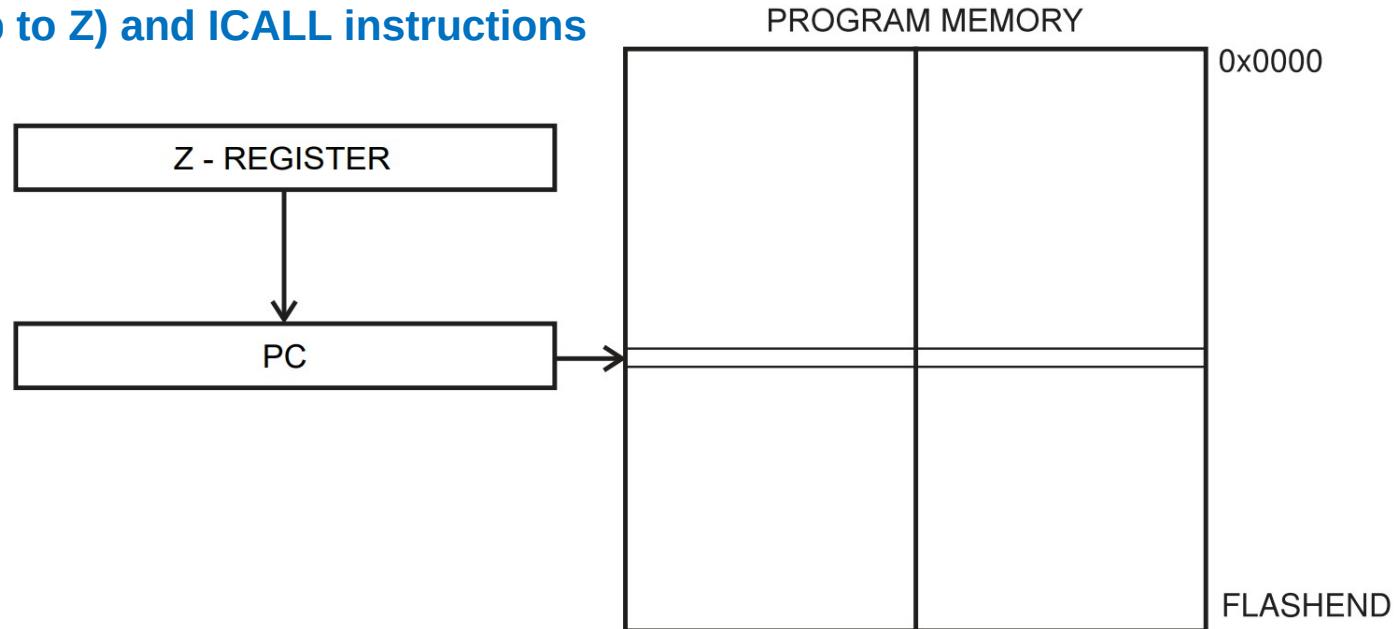


Program execution continues at the address immediate in the instruction word.

# AVR Program Memory Addressing

## Indirect Program Memory Addressing

IJMP (Indirect Jump to Z) and ICALL instructions

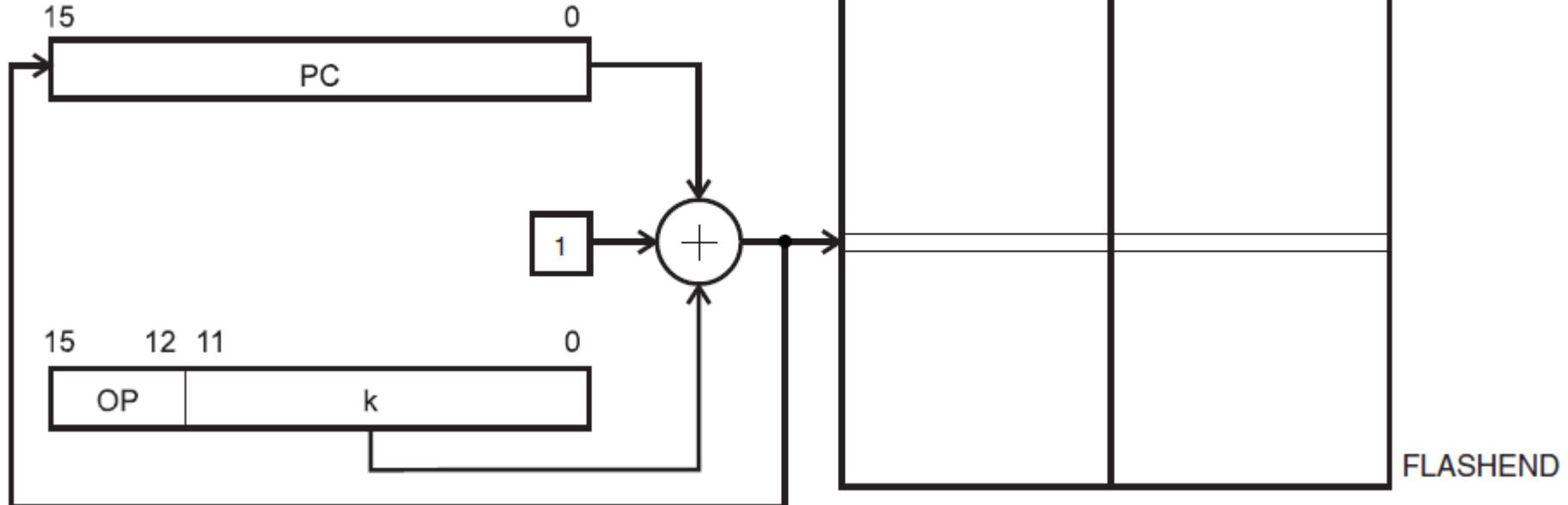


Program execution continues at the address contained by the Z-register (i.e., the PC is loaded with the contents of the Z-register).

# AVR Program Memory Addressing

## Relative Program Memory Addressing

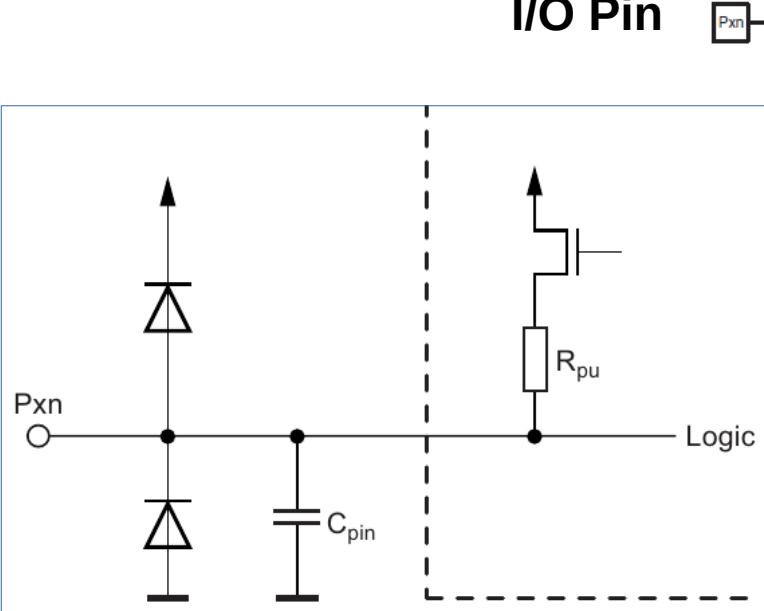
### RCALL (Relative Call Subroutine) instruction



Program execution continues at address  $PC + k + 1$ .  
The relative address  $k$  is from -2048 to 2047.

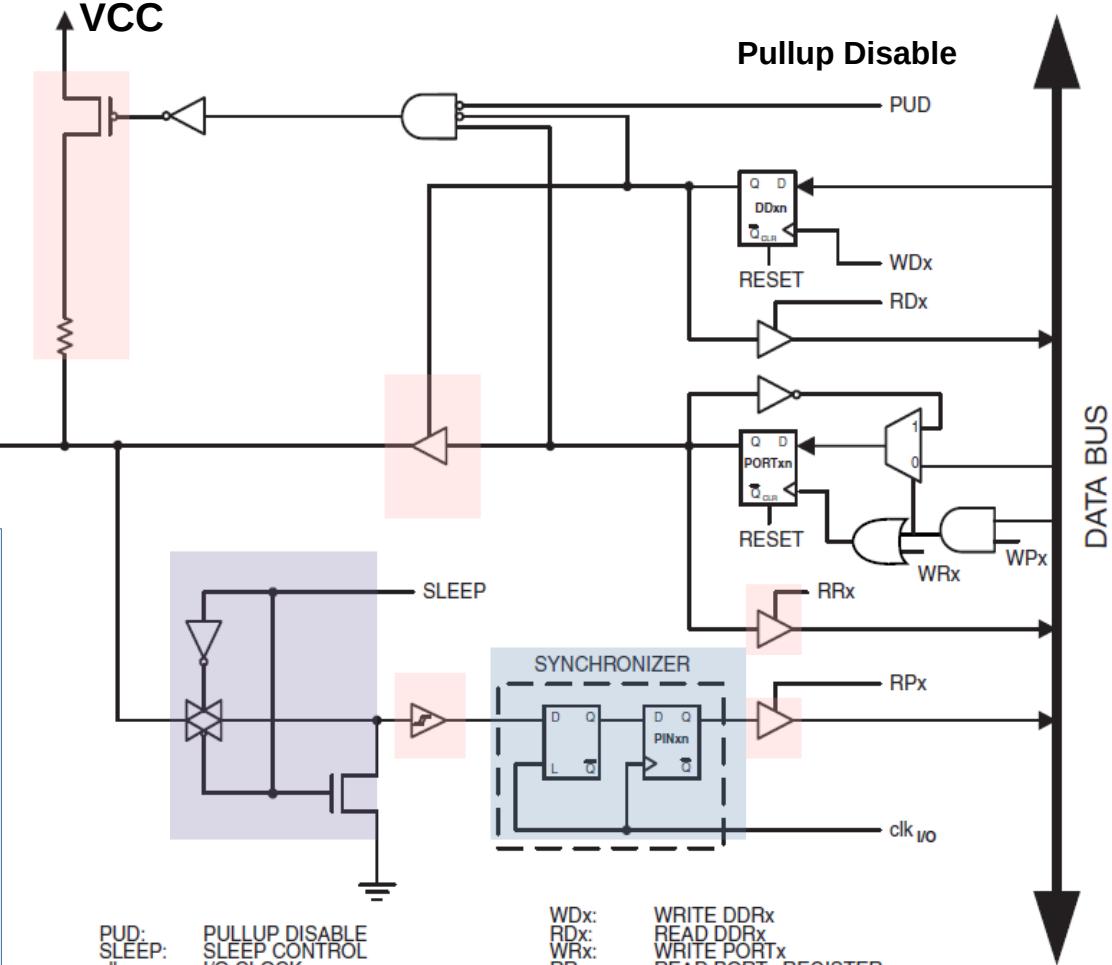
# I/O Block (ATmega328P)

Note: Data paths and control logic for **Alternate Functions** of the I/O pin are not shown.



I/O Pin

$P_{xn}$



PUD:  
SLEEP:  
clk<sub>I/O</sub>:

PULLUP DISABLE  
SLEEP CONTROL  
I/O CLOCK

WDx:  
RDx:  
WRx:  
RRx:  
RPx:  
WPx:

WRITE DDRx  
READ DDRx  
WRITE PORTx  
READ PORTx REGISTER  
READ PORTx PIN  
WRITE PINx REGISTER

# Registers Associated with Each I/O Port

Each 8-bit I/O port has the following registers:

## 1. Data Register – PORTx (Read/Write)

- When pin is configured as output, writing to **PORTx** writes the output value.
- When pin is input, writing a ‘1’ to a bit enables the **internal pull-up resistor**.

## 2. Data Direction Register – DDRx (Read/Write)

- Controls input/output direction (1 = Output, 0 = Input)
- Example: `DDRB |= (1<<PB0); // PB0 as output`

## 3. Port Input Pins Register – PINx (Read-only)

- Reading **PINx** returns the current logic level on the pin.
- Writing a ‘1’ to a bit in **PINx** toggles the corresponding output bit (for **fast toggling**).
- Example: `uint8_t value = PINC & (1<<PC2); // read PC2 bit`

# I/O Port Pin Configuration

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output low (sink)
1	1	X	Output	No	Output high (source)

```
DDRB |= (1<<3); // Configure PB3 as output (set bit)
PORTB |= (1<<3); // Set PB3 high for output (set bit)
```

```
uint8_t value;
DDRD &= ~(1<<2); // Set PD2 as input (clear bit)
PORTD |= (1<<2); // Enable internal pull-up on PD2 (set bit)
value = PIND & (1<<2); // Read logical level on PD2 (masked bit)
```

# Registers Associated with Each I/O Port

## Read-Modify-Write Behavior

AVR architecture supports bit-level changes without affecting other bits in the register.

- AVR instructions like **SBI**, **CBI**, and bit-masked writes perform an **atomic read-modify-write operation** on a single bit.
- Writing to any bit in **PORTx**, **DDRx**, or **PINx** does not modify the other bits in the same register.
- Example (safe bit set):

```
// Sets PD3 only; no effect on other PORTD bits  
PORTD |= (1<<PD3);
```

# Registers Associated with Each I/O Port

Two addresses:  
- I/O register address  
- Address in data memory space

## PORTE – The Port E Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	<b>PORTE7</b>	<b>PORTE6</b>	<b>PORTE5</b>	<b>PORTE4</b>	<b>PORTE3</b>	<b>PORTE2</b>	<b>PORTE1</b>	<b>PORTE0</b>	<b>PORTE</b>
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

## DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	<b>DDRB7</b>	<b>DDRB6</b>	<b>DDRB5</b>	<b>DDRB4</b>	<b>DDRB3</b>	<b>DDRB2</b>	<b>DDRB1</b>	<b>DDRB0</b>	<b>DDRB</b>
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

## PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	<b>PINB7</b>	<b>PINB6</b>	<b>PINB5</b>	<b>PINB4</b>	<b>PINB3</b>	<b>PINB2</b>	<b>PINB1</b>	<b>PINB0</b>	<b>PINB</b>
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A								

**Read-only**

# Registers Associated with Each I/O Port

## PORTC – The Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	–	<b>PORTC6</b>	<b>PORTC5</b>	<b>PORTC4</b>	<b>PORTC3</b>	<b>PORTC2</b>	<b>PORTC1</b>	<b>PORTC0</b>	<b>PORTC</b>
Read/Write	R	R/W							
Initial Value	0	0	0	0	0	0	0	0	

## DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	<b>DDC6</b>	<b>DDC5</b>	<b>DDC4</b>	<b>DDC3</b>	<b>DDC2</b>	<b>DDC1</b>	<b>DDC0</b>	<b>DDRC</b>
Read/Write	R	R/W							
Initial Value	0	0	0	0	0	0	0	0	

## PINC – The Port C Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	–	<b>PINC6</b>	<b>PINC5</b>	<b>PINC4</b>	<b>PINC3</b>	<b>PINC2</b>	<b>PINC1</b>	<b>PINC0</b>	<b>PINC</b>
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	N/A							

# Registers Associated with Each I/O Port

## PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	<b>PORTD7</b>	<b>PORTD6</b>	<b>PORTD5</b>	<b>PORTD4</b>	<b>PORTD3</b>	<b>PORTD2</b>	<b>PORTD1</b>	<b>PORTD0</b>	PORTD
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

## DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	<b>DDD7</b>	<b>DDD6</b>	<b>DDD5</b>	<b>DDD4</b>	<b>DDD3</b>	<b>DDD2</b>	<b>DDD1</b>	<b>DDD0</b>	DDRD
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

## PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	<b>PIND7</b>	<b>PIND6</b>	<b>PIND5</b>	<b>PIND4</b>	<b>PIND3</b>	<b>PIND2</b>	<b>PIND1</b>	<b>PIND0</b>	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A								

# MCUCR – PUD Bit

## MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS	BODSE	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

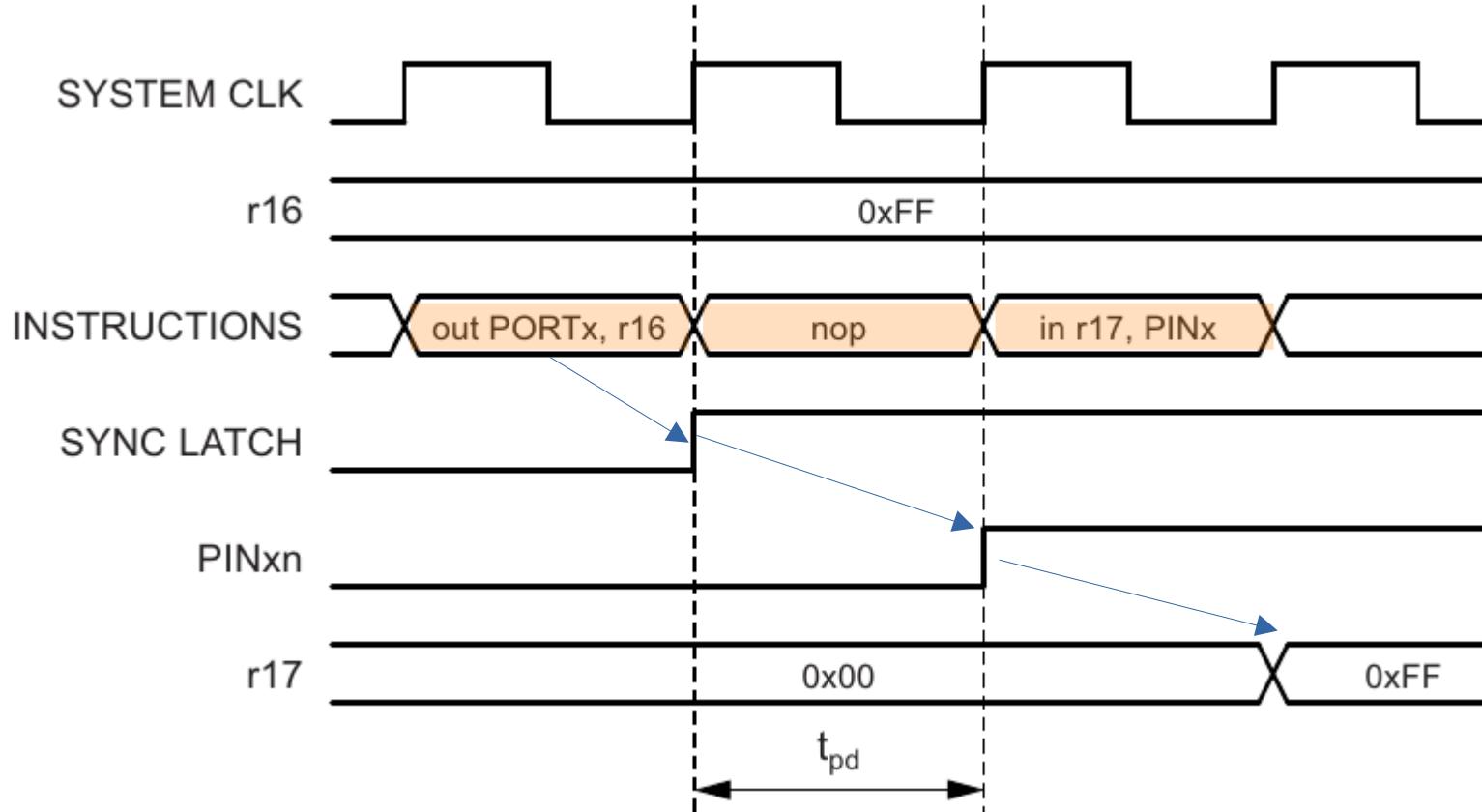
- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn registers are configured to enable the pull-ups ( $\{DDxn, PORTxn\} = 0b01$ ).  
details about this feature.

# I/O Ports and Pull-up Resistors

- Each digital input pin of all I/O ports has a **bit-selectable internal pull-up resistor** (approximately 20–50 kΩ) that can be enabled or disabled through software.
- Each pull-up resistor is controlled in software by writing a 1 to the corresponding bit in **PORTx** while the pin is configured as input (**DDRx = 0**).
- ATmega328P has no programmable pull-down resistors.
- Each I/O pin has internal **protection diodes (clamp diodes)** to VCC and GND. They are used to protect against voltage excursions beyond VCC + 0.5V and below GND – 0.5V.
- Maximum recommended current through the diode: **I/O pin current** must stay within ±40 mA per pin (max, or **±20mA for safe**), otherwise damage may occur.

# I/O Bit Write / Read Operations



When reading back a software assigned pin value, a **nop** instruction must be inserted.

# I/O Operations: C and Assembly Code Examples

```
...
; Define pull-ups and set outputs high
; Define directions for port pins
ldi    r16, (1<<PB7) | (1<<PB6) | (1<<PB1) | (1<<PB0)
ldi    r17, (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0)
out   PORTB, r16
out   DDRB, r17
; Insert nop for synchronization
nop
; Read port pins
in    r16, PINB
...
```

```
unsigned char i;
...
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7) | (1<<PB6) | (1<<PB1) | (1<<PB0);
DDRB = (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0);
/* Insert nop for synchronization*/
no_operation();
/* Read port pins */
i = PINB;
...
```

- Set port B pins **0** and **1** high, **2** and **3** low.
- Define the port B pins from **4** to **7** as **input**, with **pull-ups** assigned to port B pins **6** and **7**.

PORTB = 0b**11000011**  
DDRB = 0b**00001111**  
inputs   outputs

# AVR Assembly Code Example (1)

## AVRASM2 Version

```
.EQU DDRB = 0x04          ; Data Direction Register for Port B
.EQU PORTB = 0x05         ; Port B Data Register (controls output)
.CSEG                      ; Code Segment (Program Memory Section)
.ORG 0x0000
RJMP MAIN                  ; Reset Vector

MAIN:
    ; Configure PB5 as output (read-modify-write)
    LDI R17, 0x20           ; R17 = 0x20
    IN  R16, DDRB            ; Read current DDRB
    OR  R16, R17              ; R16 ← R16 or R17
    OUT DDRB, R16             ; Write back to DDRB

LOOP:
    ; Toggle PB5 using OUT (read-modify-write)
    IN  R16, PORTB           ; Read current PORTB value
    EOR R16, R17              ; R16 ← R16 xor R17
    OUT PORTB, R16             ; Write back new toggle state
    RJMP LOOP                 ; Repeat forever
```

# AVR Assembly Code Example (2)

AVRASM2 Version

```
.EQU DDRB = 0x04          ; Data Direction Register for Port B
.EQU PORTB = 0x05         ; Port B Data Register (controls output)
.CSEG                      ; Code Segment (Program Memory Section)
.ORG 0x0000
RJMP MAIN                  ; Reset Vector

MAIN:
    ; Configure PB5 as output
    SBI DDRB, 5           ; Set bit 5 (PB5 output)
LOOP:
    SBI PORTB, 5          ; Output PB5 = 1
    CBI PORTB, 5          ; Output PB5 = 0
    RJMP LOOP              ; Repeat forever
```

# AVR Assembly Code Example (3)

## AVRASM2 Version

```
.EQU DDRB = 0x04          ; Data Direction Reg B
.EQU PORTB = 0x05          ; Port B Output Register
.EQU DDRD = 0x0A          ; Data Direction Reg D
.EQU PORTD = 0x0B          ; Port D Output Register
.EQU PIND = 0x09          ; Port D Input Pins Register

.CSEG
.ORG 0x0000                ; Program start address (Reset Vector)
RJMP MAIN                  ; The first instruction at 0x0000

MAIN:    CBI DDRD, 2        ; Clear DDRD2 bit → PD2 input
         SBI PORTD, 2        ; Set PORTD2 bit → enable pull-up
         SBI DDRB, 5          ; Set DDRB5 bit → PB5 output
         CBI PORTB, 5          ; Clear PB5 output (LED OFF)
LOOP:    SBIC PIND, 2        ; Skip next instruction if PD2 = 0
         RJMP LED_OFF        ; If PD2 = 1 → go turn LED OFF
LED_ON:   SBI PORTB, 5        ; PB5 HIGH (turn LED ON)
         RJMP LOOP
LED_OFF: CBI PORTB, 5        ; PB5 LOW (turn LED OFF)
         RJMP LOOP
```

# AVR Instruction: IN

## IN - Load an I/O Location to Register

### Description

Loads data from the I/O space into register Rd in the Register File.

Operation:

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

Syntax:

(i) IN Rd,A

Operands:

$0 \leq d \leq 31, 0 \leq A \leq 63$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

1011	0AA <sub>d</sub>	dd <sub>dd</sub>	AA <sub>AA</sub>
------	------------------	------------------	------------------

### Status Register (SREG) and Boolean Formula

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

# AVR Instruction: OUT

## OUT – Store Register to I/O Location

### Description

Stores data from register Rr in the Register File to I/O space.

Operation:

(i)  $I/O(A) \leftarrow Rr$

Syntax:

(i) OUT A,Rr

Operands:

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

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

1011	1AAr	rrrr	AAAA
------	------	------	------

### Status Register (SREG) and Boolean Formula

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

# AVR Instruction: OR

## OR – Logical OR

### Description

Performs the logical OR between the contents of register Rd and register Rr, and places the result in the destination register Rd.

Operation:

(i)  $Rd \leftarrow Rd \vee Rr$

Syntax:

(i) OR Rd,Rr

Operands:

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

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0010	10rd	dddd	rrrr
------	------	------	------

### Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
-	-	-	$\Leftrightarrow$	0	$\Leftrightarrow$	$\Leftrightarrow$	-

# AVR Instruction: EOR

## EOR – Exclusive OR

### Description

Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i)  $Rd \leftarrow Rd \oplus Rr$

Syntax:

(i) EOR Rd,Rr

Operands:

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

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0010	01rd	dddd	rrrr
------	------	------	------

### Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
-	-	-	$\Leftrightarrow$	0	$\Leftrightarrow$	$\Leftrightarrow$	-

# AVR Instruction: SBI

## SBI – Set Bit in I/O Register

### Description

Sets a specified bit in an I/O Register. This instruction operates on the lower 32 I/O Registers – addresses 0-31.

Operation:

$$(i) \quad I/O(A,b) \leftarrow 1$$

Syntax:

Operands:

Program Counter:

$$(i) \quad SBI A,b$$

$0 \leq A \leq 31, 0 \leq b \leq 7$

$$PC \leftarrow PC + 1$$

16-bit Opcode:

1001	1010	AAAA	Abbb
------	------	------	------

### Status Register (SREG) and Boolean Formula

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

# AVR Instruction: SBIC

## SBIC – Skip if Bit in I/O Register is Cleared

### Description

This instruction tests a single bit in an I/O Register and skips the next instruction if the bit is cleared. This instruction operates on the lower 32 I/O Registers – addresses 0-31.

Operation:

- (i) If  $I/O(A,b) == 0$  then  $PC \leftarrow PC + 2$  (or 3) else  $PC \leftarrow PC + 1$

Syntax:

Operands:

Program Counter:

- (i) SBIC A,b

$0 \leq A \leq 31, 0 \leq b \leq 7$

$PC \leftarrow PC + 1$ , Condition false - no skip

$PC \leftarrow PC + 2$ , Skip a one word instruction

$PC \leftarrow PC + 3$ , Skip a two word instruction

16-bit Opcode:

1001	1001	AAAA	Abbb
------	------	------	------

### Status Register (SREG) and Boolean Formula

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