ETRO
ELECTRONICS & INFORMATICS

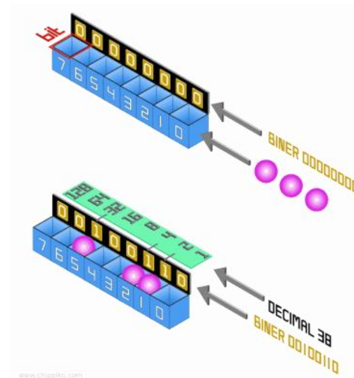
# Sensors & Microsystem Electronics: microcontrollers

BACKGROUND 1: REGISTERS, INSTRUCTIONS AND LOOPS

# What is a register?

A register is a place in a processor that can store data as a combination of bits

- Readable & writable at runtime
- Stores values from 0 to $2^n$-1  (n = number of bits)
  - 8-bit microcontroller: values from 0 to 255
- Store data as bitmap
  - Bit pattern or mapping represented as an 8-bit value
- Some are bitaddressable
  - Read or Modify 1 bit at the time with specific instructions



# Types of Registers

- General Purpose Registers
  - Fast
  - Used as operands in program logic
  - In processor core
- Special Function Registers (SFR)
  - In RAM address space
  - Some are read-only
  - Specific functions
    - Flags
    - I/O ports
    - Peripherals
  - E.g. PINB, PORTB, ADCSRA,…

**14.4.2 PORTB – The Port B Data Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x05 (0x25) | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 | PORTB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**14.4.3 DDRB – The Port B Data Direction Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x04 (0x24) | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 | DDRB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**14.4.4 PINB – The Port B Input Pins Address[1]**

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

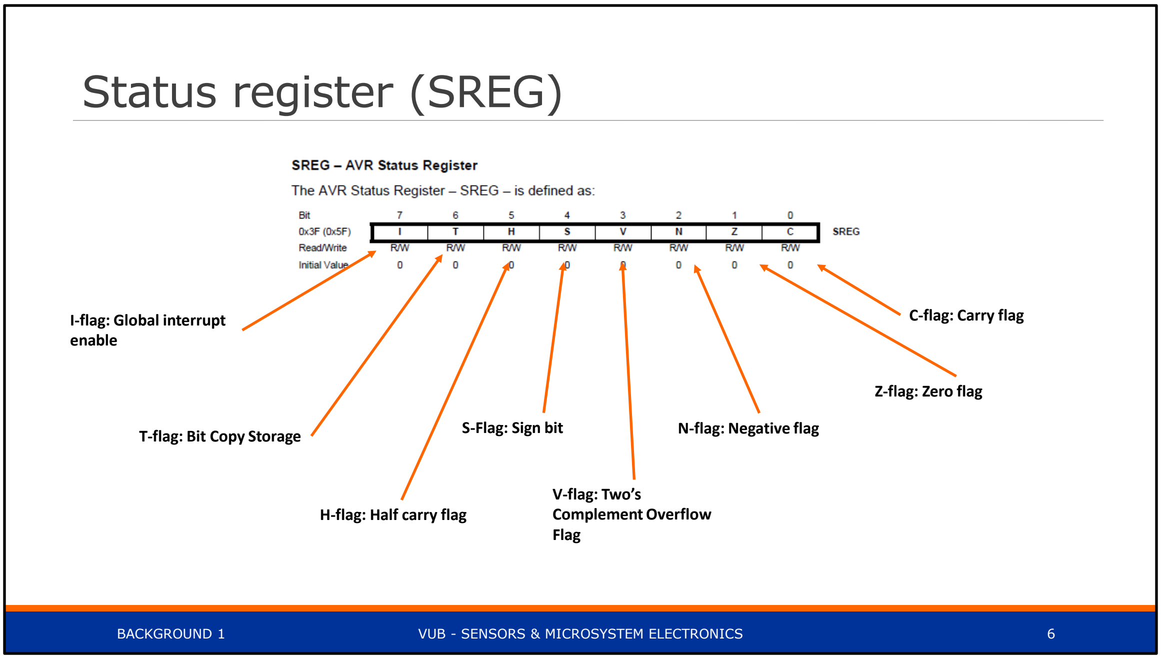
## General purpose registers

```
; Finite loop
LDI R16,0x14 ; Store constant in GPR R16
LDI R10,0x04 ; Store constant in GPR R10
ADD R16, R10 ; Operation (SUM) on R16 and R10
MyLoop: ; Loop label
   NOP
   DEC R16 ; Operation (DEC) on R16
   BRNE MyLoop ; Go back to loop label if R16 did not become 0
```

# What are Special Function Registers (SFR)?

- Control on-chip hardware: Control Bits
  - DDRA: pin direction for pinbank A pins
  - PORTD: pin configuration for pinbank D pins
  - TCCR0A: timer 0 configuration register A
- Signals from on-chip hardware: Flag Bits
  - PINB: pin state for pinbank B pins
  - ADCL: lower 8 bits of the ADC conversion result
  - SREG: status register flags from ALU operation result

- Flag bits changes their status when a certain situation occurs in the uC.




The SREG is the main status register. It has 8 flags with a different purpose. Bit 7 is the global interrupt flag to enable or disable the interrupts. All the other flags are set or cleared by the ALU based on the result of the last instruction executed.

As an example: for the instruction **DEC R16**, this instruction decrements R16, if the result of this instruction results in R16 becoming zero the Z or " Zero-Flag" will be set to **1**

Another example: if an addition **ADD** results in the need of a carry bit (result is larger than 8 bit) the C-flag or " Carry-Flag" will be set.

<u>Below is the description of the flags in the SREG from the datasheet (section 7.3.1):</u>

**• Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable
control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of
the interrupts are enabled independent of the individual interrupt enable settings.

The I-bit is cleared by
hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts.
The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the
instruction set reference.

**• Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the
operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T
can be copied into a bit in a register in the Register File by the BLD instruction.

**• Bit 5 – H: Half Carry Flag**

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Is useful in BCD
arithmetic. See the “Instruction Set Description” for detailed information.

**• Bit 4 – S: Sign Bit, S = N $\oplus$ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two’s Complement Overflow Flag V.
See the “Instruction Set Description” for detailed information.

**• Bit 3 – V: Two’s Complement Overflow Flag**

The Two’s Complement Overflow Flag V supports two’s complement arithmetic. See the “Instruction Set
Description” for detailed information.

**• Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the “Instruction Set
Description” for detailed information.

**• Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the “Instruction Set Description”
for detailed information.

**• Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the “Instruction Set Description” for
detailed information.

# Branching

- Conditional branching statements such as BRNE (Branch if not equal) are found in code usually as follows:
- **DEC R19**
  **BRNE** someLabel

  The ALU cannot know for this branching instruction that the last operation was on R19. The condition for the branching is whether a certain flag is set in the SREG as a result of the previous instruction. For BRNE, the branching happens if the Z is cleared. Different branching instructions check the value of different states for different flags. For a more detailed description check the instruction set manual.

# Infinite loops

```
main: ; Loops always begin with a label

    ; Your logic goes here


    RJMP main ; Go back to the beginning
```

## Finite loops

| Mnemonic | Operands | Description | Operation | Flag | Cycles |
|---|---|---|---|---|---|
| BRNE | k | Branch if Not Equal | if (Z== 0) then PC= PC + k + 1 | None | 1/2. |

```
; Finite loop ~comparable to FOR loop in C

LDI R16,0x14 ; choose and pre-load your iterator var
MyLoop:      ; Loops always begin with a label

    ; Your logic goes here

    DEC R16 ; decrement your iterator
    BRNE MyLoop   ; check Z to see if iterator became
                  zero. If not 0, go back to the
                  beginning else go to next line
```

**L**oa**D** **I**mmediate (**LDI**) in register

**DEC**rement (**DEC**) register by 1

**BR**anch if **N**ot **E**qual (**BRNE**) branches if the zero flag (Z flag) = 0

When R16=0, Z= 1.



In this case, when R16 goes to zero, the zero flag is cleared, and the branching occures.

# Instructions and their operands

- Instructions are what your uC executes
- Can take 1 or more clock cycles depending on the instruction
- Operands are the information that the instructions need to do their thing
  - Possible options
    - No operands needed e.g. NOP; CLI; SEI
    - Single register e.g. TST r0; INC r21
    - Constants and Registers e.g. LDI r16, 12
    - Registers and other registers e.g. IN r2, PINB; CP R1,R6
    - Registers and memory address e.g. STS 0x006E, r0; LDS r6, TIMSK0
    - ...

# Directives

- Are not operations
- Are executed/interpreted by the compiler when building the project
- Examples
  - .def
  - .equ
  - High([arg])
  - Low([arg])
  - .macro

## Constants and Expressions

**EQU - Set a symbol equal to an expression**

**Syntax:**
.EQU label = expression
**Example:**
.EQU io_offset = 0x23
.EQU porta = io_offset + 2

- **DEF - Set a symbolic name on a register**

**Syntax:**
.DEF Symbol=Register
**Example:**
.DEF temp=R16
.DEF ior=R0

Use: ldi temp,0xf0 ; Load 0xf0 into temp register



These are the two most used compiler directives (other than MACRO) These can be used to define a specific expression or name a register. This is very similar to a #define form the C lanuage.

# Constants and prefixes

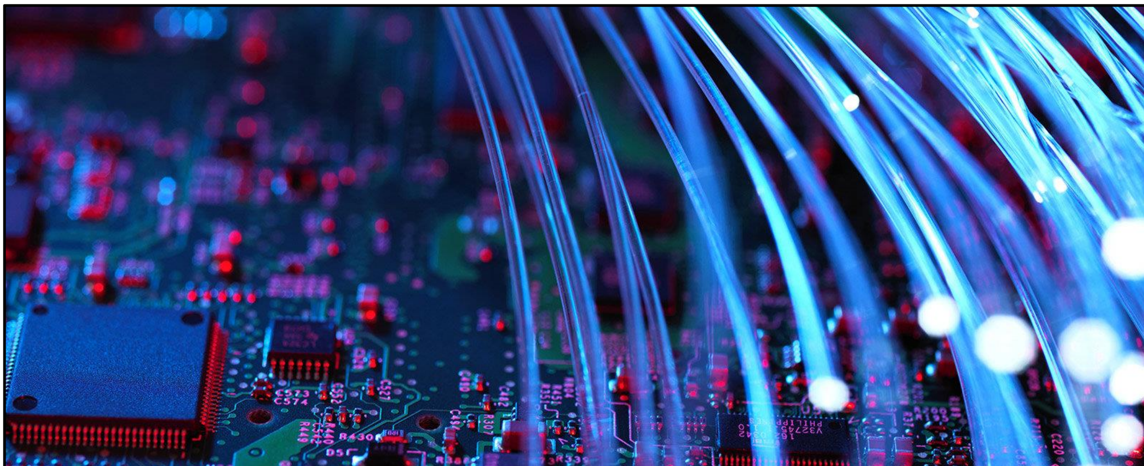
- Decimal notation: no prefix
  - Example: 70
  - Code example: LDI R16, 70

**Leading zeros can be omitted:**

**0b00010010 = 0b10010**
**0x05 = 0x5**

- Hexadecimal notation: prefix 0x_____
  - Example: 0x46 (70 in decimal)
  - Code example : LDI R16, 0x46

- Binary notation: prefix 0b_____
  - Example: 0b1000110 (70 in decimal)
  - Code example : LDI R16, 0b1000110



End of background 1: registers, instructions and loops