

# **Microcontroladores**

Prof. Marcos Chaves

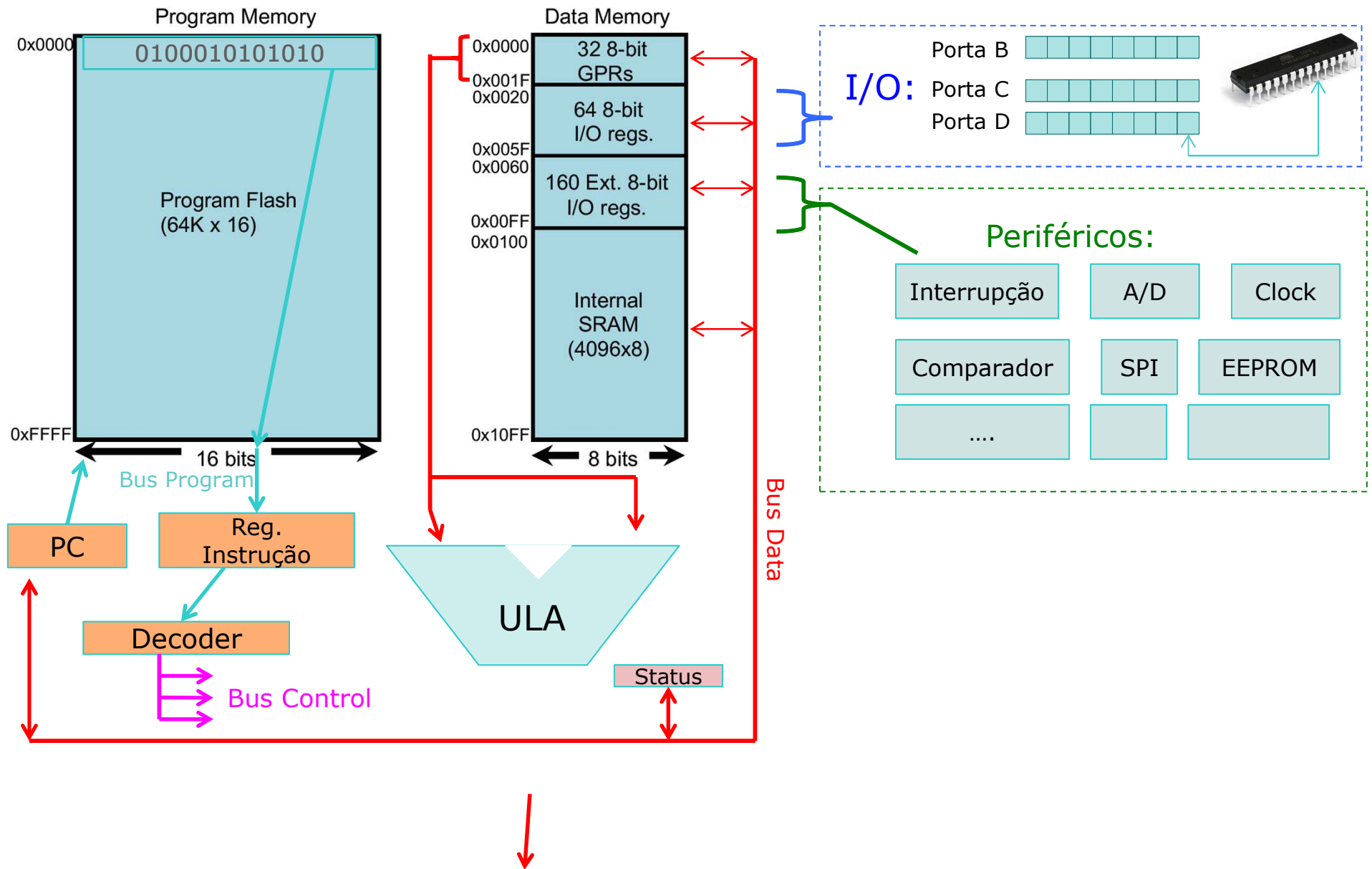
Set Instructions Assembly

# Atmega328P

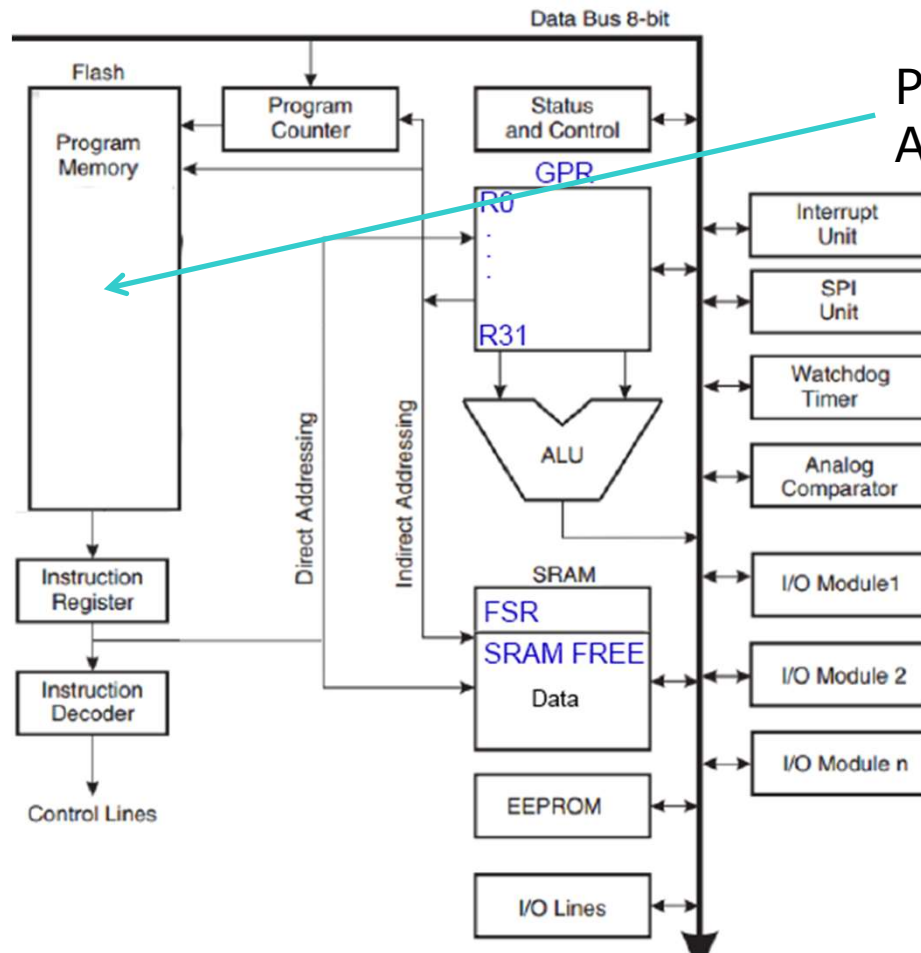


(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(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 ( $\overline{SS}$ /OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

Power
Ground
Programming/debug
Digital
Analog
Crystal/Osc



# Arquitetura Microcontrolador Atmega328



Programa Assembly



# Assembly

- A linguagem Assembly é atrelada à arquitetura de uma certa CPU, ou seja, ela depende completamente do hardware
- Por essa razão Assembly não é uma linguagem portátil, ao contrário da maioria das linguagens de alto nível

- Linguagem de programação - Assembly

As linguagens mais populares para programação de microcontroladores são:

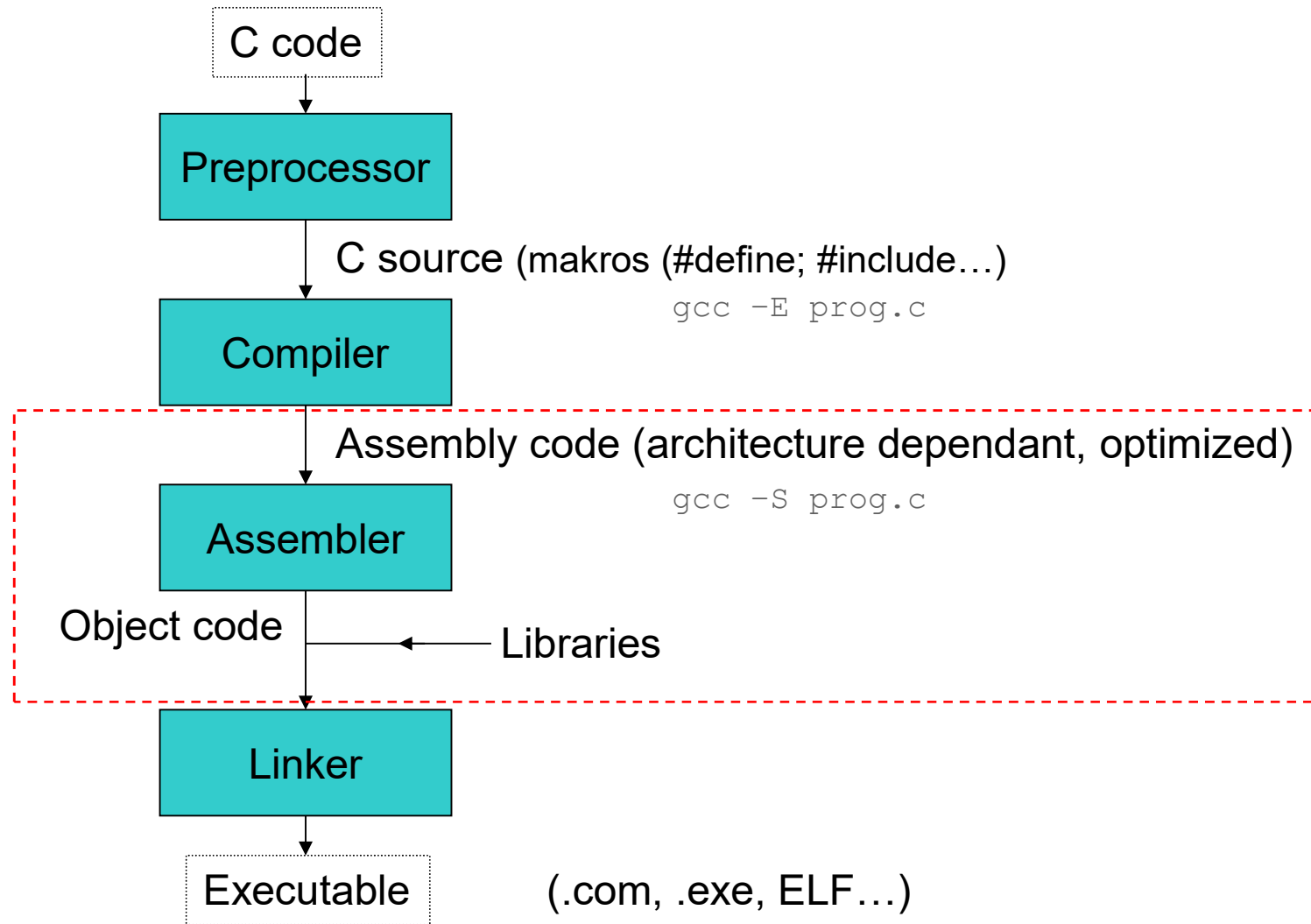
- Assembly: É a linguagem nativa que tem uma abordagem direta ao seu núcleo. Muito importante conhecer mesmo com o melhor compilador em outras linguagens, pois possibilita desenvolver códigos com melhor eficiência, ou seja, mais rápido, melhor utilização dos recursos e menor consumo de memória.

- C: Por excelência, é a linguagem da engenharia por causa de sua estrutura, portabilidade e reutilização de recursos de processamento. Os compiladores dessa linguagem estão cada vez mais otimizadas.

# Assembly - Assembler

- A linguagem Assembly é de baixo nível, porém ainda precisa ser transformada na linguagem que a máquina entende
- Quem faz isso é o Assembler. O Assembler é um utilitário que traduz o código Assembly para a máquina

# Compilação





# Efeito do Compilador:

- Exemplo:

Antes -> mov al, 061h (x86/IA-32)

Depois -> 10110000 01100001

# Formato Linguagem Assembly

- Rótulo (demarca uma posição, endereço)
  - Está na margem esquerda
  - Deve iniciar por letra ou \_
  - 2º letra p frente pode ser qquer caractere alfanumérico
  - Correto: INICIO35b, \_ABC3FG etc
  - Errado: 3fby, OPÇÃO, VX%
  - Rótulo especial \$: indica o próprio endereço
- Referência a rótulo: quando o símbolo do rótulo é um parâmetro de uma instrução ou diretiva (refere-se a um rótulo)

# Rótulo:

Um **Label** (etiqueta/rótulo) é uma designação textual (geralmente de fácil leitura) de uma linha num programa ou de uma seção de um programa para onde um microcontrolador deve saltar ou, ainda, o início de um conjunto de linhas de um programa.

O comprimento de uma label pode ir até 32 caracteres. É também importante que a Label comece na primeira coluna.

# Formato Linguagem Assembly

- Comentários
  - Iniciam por ; Tudo que vem depois até o final da linha é ignorado durante a montagem.
- Mnemônico
  - Abreviatura da instrução.
- Parâmetros
  - A maioria das instruções possui parâmetros.

## Instruções

As instruções são específicas para cada microcontrolador, assim, se quisermos utilizar a linguagem assembly temos que estudar as instruções desse microcontrolador.

O modo como se escreve uma instrução é designado por "sintaxe".

## Operandos

Operandos são os elementos da instrução necessários, para que a instrução possa ser executada. Normalmente são registros, variáveis e constantes. As constantes são designadas por

"literais". A palavra literal significa "número".



# AVR assembly - Instruções

mnemônico

Argumento (operando)

```
ldi R16, 0b10100101
```

out PORTC, R16

Mnemo.	Operands	Description	Operation	Flags	#Clk
<b>ADD</b>	Rd, Rr	Add without Carry	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
<b>ADC</b>	Rd, Rr	Add with Carry	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
<b>ADIW</b>	Rd, K	Add Immediate to Word	$Rd+1:Rd \leftarrow Rd+1:Rd + K$	Z,C,N,V	2
<b>SUB</b>	Rd, Rr	Subtract without Carry	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1

# Criando um programa Assembly AVR

- Estruturar um programa
- Comentários ;
- Arquivos de definição: INCLUDES  
#include <nomedoarquivo.inc>  
#include "C:\MICRO1\arquivo.inc"
- Constantes e definições: .equ

.equ Nome\_da\_variavel = valor

ou

.equ Nome\_da\_variavel = valor\_constante

.equ VALOR\_HEXA = 0x0003

.equ VALOR\_BINARIO = 0b00000001

.equ VALOR\_DECIMAL = 123

## Tipos de instruções

- Lógica e aritmética
- Desvios e chamadas
- Movimentação de dados
- Manipulação de bits
- Teste de bit



# AVR assembly – Instruções

## Lógica e Aritmética

a+b	ADD
a-b	SUB
a&b	AND
a b	OR
a++	INC
a--	DEC
-a	NEG
a=0	CLR
...	...

## Move

reg1=reg2	MOV
reg=17	LDI
reg=mem	LDS
reg=*mem	LD
mem=reg	STS
*mem=reg	ST
periperal	IN
peripheral	OUT
heap	PUSH
heap	POP
...	...

## Operadores de Bit

a<<1	LSL
a>>1	LSR,
Ø C (not avail. In C)	ROL, ROR
Status bits	SEI, CLI, CLZ...
No op.	NOP
...	...

# AVR assembly – Desvios (rjump)

- RJMP: Pula para linha do programa incondicionalmente

Exemplo Loop

Em C:

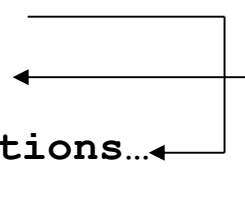
```
M_LOOP:
    ...instructions...
    rjmp M_LOOP
```

```
while (1) {
    ...instructions...
}
```

- CALL, RET: Chama Rotina e retorna

Exemplo Subrotina:

```
M_LOOP:
    ...
    CALL FV
    ...
FV: ...instructions...
    RET
```



```
void fv() { ...instructions...
    return;
}
void main () {...
    fv();
}
```

## Lógica e aritmética

AND	Rd, Rr	Lógica E entre registradores	$Rd \leftarrow Rd \bullet Rr$	Z, N, V	1
ANDI	Rd, K	Lógica E entre registrador e constante	$Rd \leftarrow Rd \bullet K$	Z, N, V	1
OR	Rd, Rr	Lógica OU entre registradores	$Rd \leftarrow Rd \vee Rr$	Z, N, V	1
ORI	Rd, K	Lógica OU entre registrador e constante	$Rd \leftarrow Rd \vee K$	Z, N, V	1
ADD	Rd, Rr	Soma dois registradores	$Rd \leftarrow Rd + Rr$	Z, C, N, V, H	1
ADC	Rd, Rr	Soma dois registradores com <i>Carry</i>	$Rd \leftarrow Rd + Rr + C$	Z, C, N, V, H	1
SUB	Rd, Rr	Subtrai dois registradores	$Rd \leftarrow Rd - Rr$	Z, C, N, V, H	1
CLR	Rd	Limpa registrador	$Rd \leftarrow Rd \otimes Rd$	Z, N, V	1
INC	Rd	Incrementa registrador	$Rd \leftarrow Rd + 1$	Z, N, V	1
DEC	Rd	Decrementa registrador	$Rd \leftarrow Rd - 1$	Z, N, V	1

## Desvios e chamadas

RJMP	k	Desvio relativo	$PC \leftarrow PC + k + 1$	Nenhum	2
RCALL	k	Chama de sub-rotina	$PC \leftarrow PC + k + 1$	Nenhum	3
RET		Retorno de sub-rotina	$PC \leftarrow STACK$	Nenhum	4
RETI		Retorno de interrupção	$PC \leftarrow STACK$	I	4

## Movimentação de dados

MOV	Rd, Rr	Movimento entre registradores	$Rd \leftarrow Rr$	Nenhum	1
LDI	Rd, K	Carrega valor imediato	$Rd \leftarrow K$	Nenhum	1
IN	Rd, P	Leitura de registrador de I/O	$Rd \leftarrow P$	Nenhum	1
OUT	P, Rr	Escrita de registrador de I/O	$P \leftarrow Rr$	Nenhum	1
LDS	Rd, k	Carrega diretamente da SRAM	$Rd \leftarrow (k)$	Nenhum	2
STS	k, Rr	Carrega diretamente para SRAM	$(k) \leftarrow Rr$	Nenhum	2

## Manipulação de bits

SBI	P, b	Ativa o bit no registrador de I/O	$I/O(P,b) \leftarrow 1$	Nenhum	2
CBI	P, b	Limpa o bit do registrador de I/O	$I/O(P,b) \leftarrow 0$	Nenhum	2

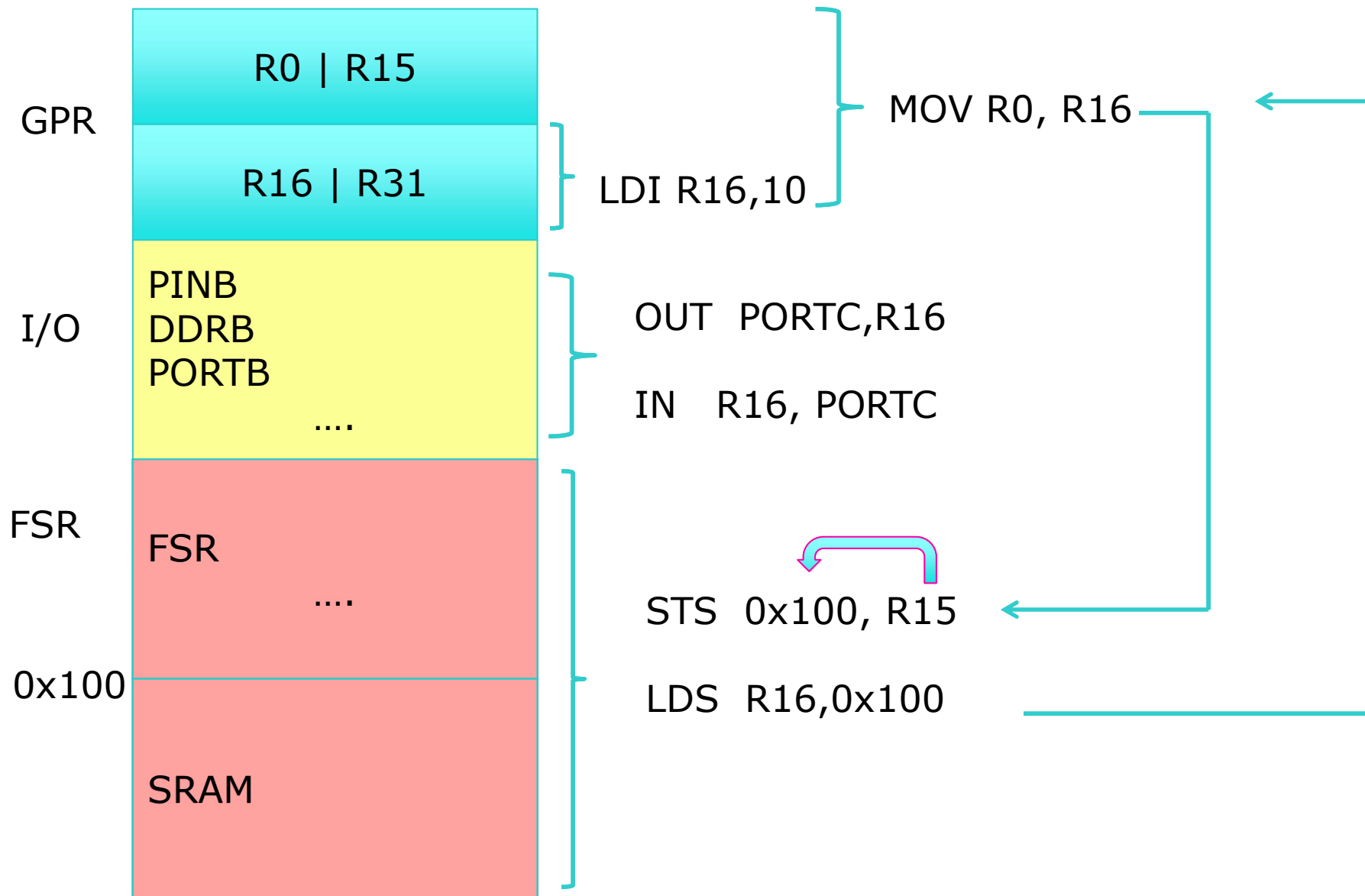
## Teste de bit

SBIC	P, b	Pula se o bit do registrador de I/O estiver limpo (0)	$\text{if}(P(b)=0) \text{ PC} \leftarrow \text{PC} + 2 \text{ ou } 3$	Nenhum	1/ 2/ 3
SBIS	P, b	Pula se o bit do registrador de I/O estiver ativo (1)	$\text{if}(P(b)=1) \text{ PC} \leftarrow \text{PC} + 2 \text{ ou } 3$	Nenhum	1/ 2/ 3
BREQ	k	Desvia se igual	$\text{if}(Z=1) \text{ PC} \leftarrow \text{PC} + k + 1$	Nenhum	1/ 2
BRNE	k	Desvia se diferente	$\text{if}(Z=0) \text{ PC} \leftarrow \text{PC} + k + 1$	Nenhum	1/ 2

## Observações:

Todas as instruções que operam com registradores de uso geral tem acesso direto em um único ciclo a todos eles. As exceções são as cinco instruções lógicas e aritméticas entre uma constante e um registrador, SBCI, SUBI, CPI, ANDI e ORI, e a instrução para carga de constante imediata, LDI. Essas instruções se aplicam somente a metade superior dos registradores de uso geral (R16..R31). As instruções SBC, SUB, CP, AND, OR e as demais operações entre um ou dois registradores se aplicam a todo o banco de registradores.

# MOVIMENTAÇÃO DE REGISTROS



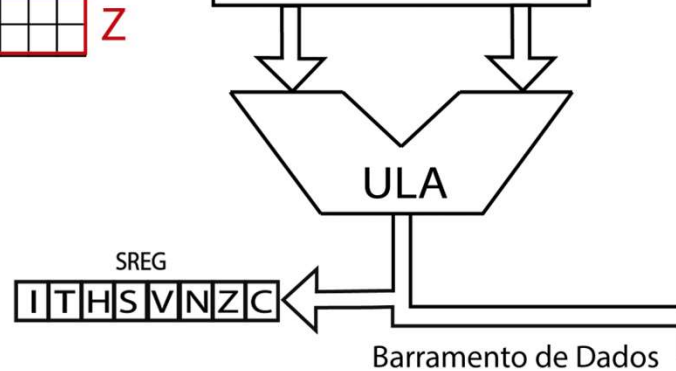
# ULA e Registradores de Trabalho

## Registradores de Propósito Geral

Diagram illustrating a memory array structure with 32 rows (R0 to R31) and 4 columns. The array is divided into two main sections: R0-R15 on the left and R16-R31 on the right. Rows R26-R27 are highlighted in yellow and labeled 'X'. Rows R28-R29 are highlighted in green and labeled 'Y'. Rows R30-R31 are highlighted in red and labeled 'Z'.

GPR

R0	R1	R2	R3
R4	R5	R6	R7
R8	R9	R10	R11
R12	R13	R14	R15
R16	R17	R18	R19
R20	R21	R22	R23
R24	R25	R26	R27
R28	R29	R30	R31



# Registrador de Status (SREG)

Bit	7	6	5	4	3	2	1	0
SREG	I	T	H	S	V	N	Z	C
Lê/Escreve	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

I – Global Interrupt Enable  
Chave Geral, habilita as interrupções

T – Bit Copy Storage  
Bit de cópia temporária de um registrador

H – Half Carry Flag  
Carry de um nibble

S – Sign Bit



# Registrador de Status (SREG)

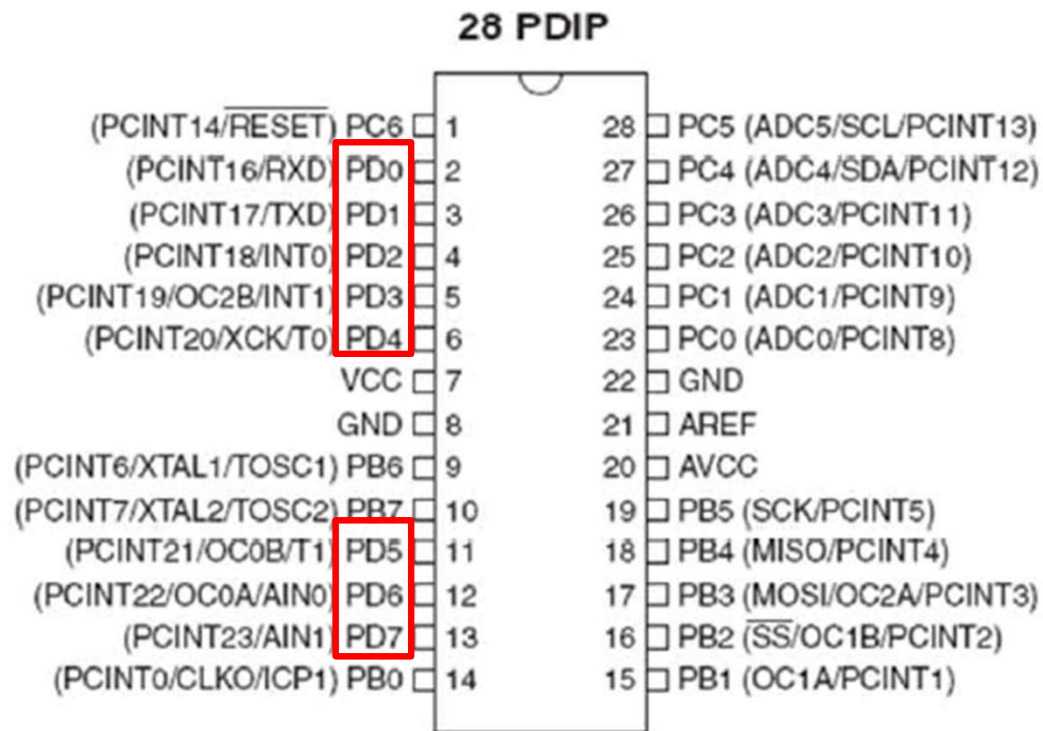
Bit	7	6	5	4	3	2	1	0
	SREG							
	I	T	H	S	V	N	Z	C
Lê/Escreve	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

V – Two's Complement Overflow Flag  
estouro do complemento de dois

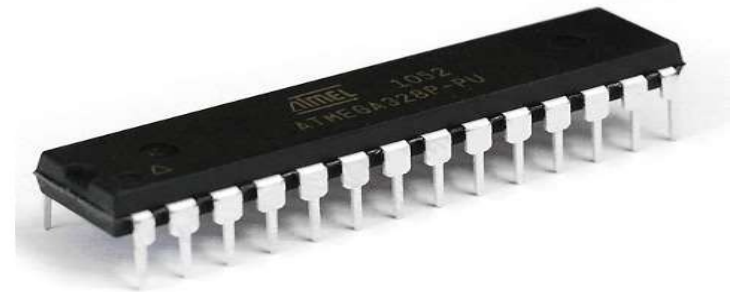
N – Negative Flag  
Indica uma operação aritmética ou lógica resulta em um valor negativo.

Z – Zero Flag  
indica quando uma operação resulta em zero.

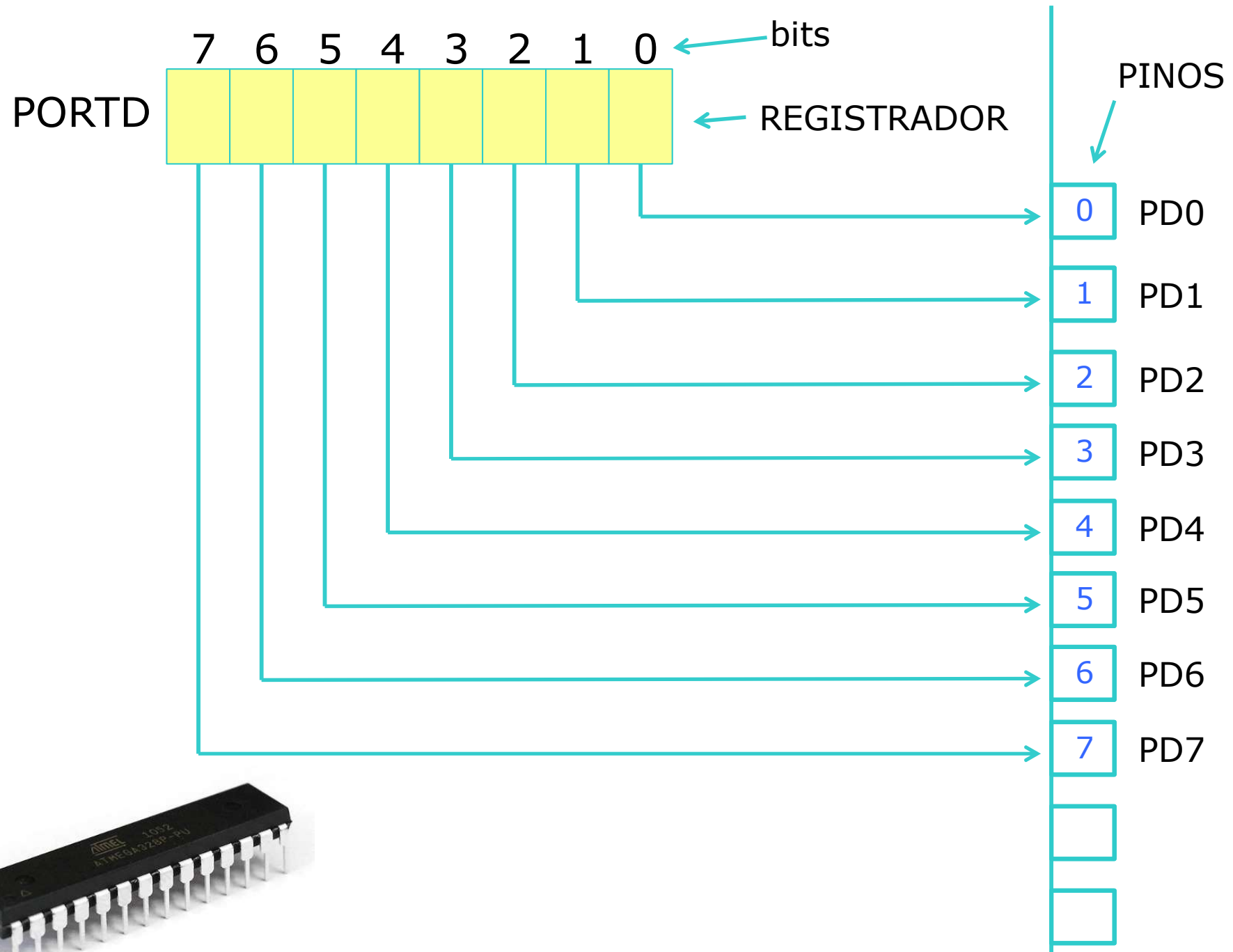
C – Carry Flag  
houve um estouro



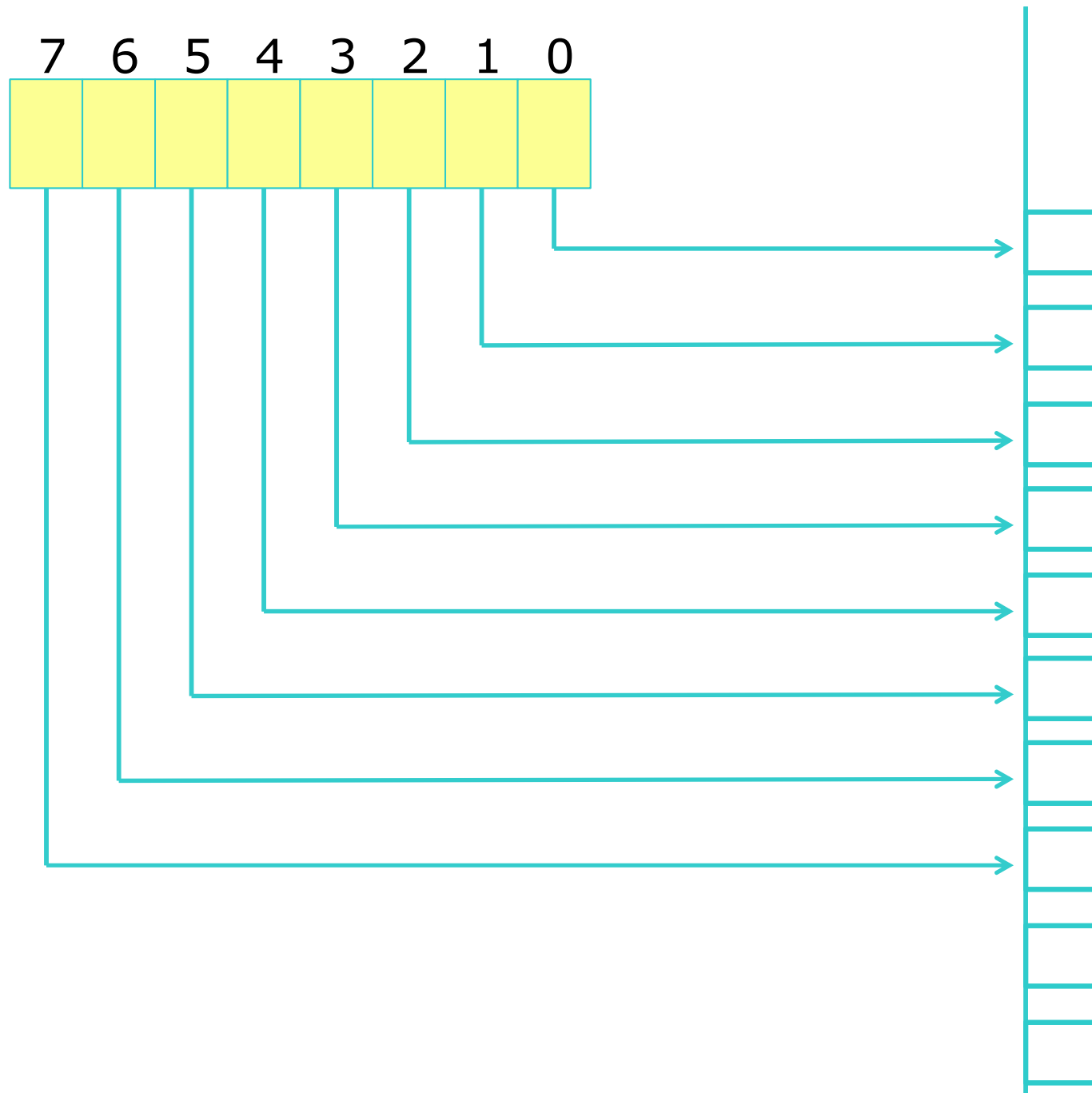
**Porta D**

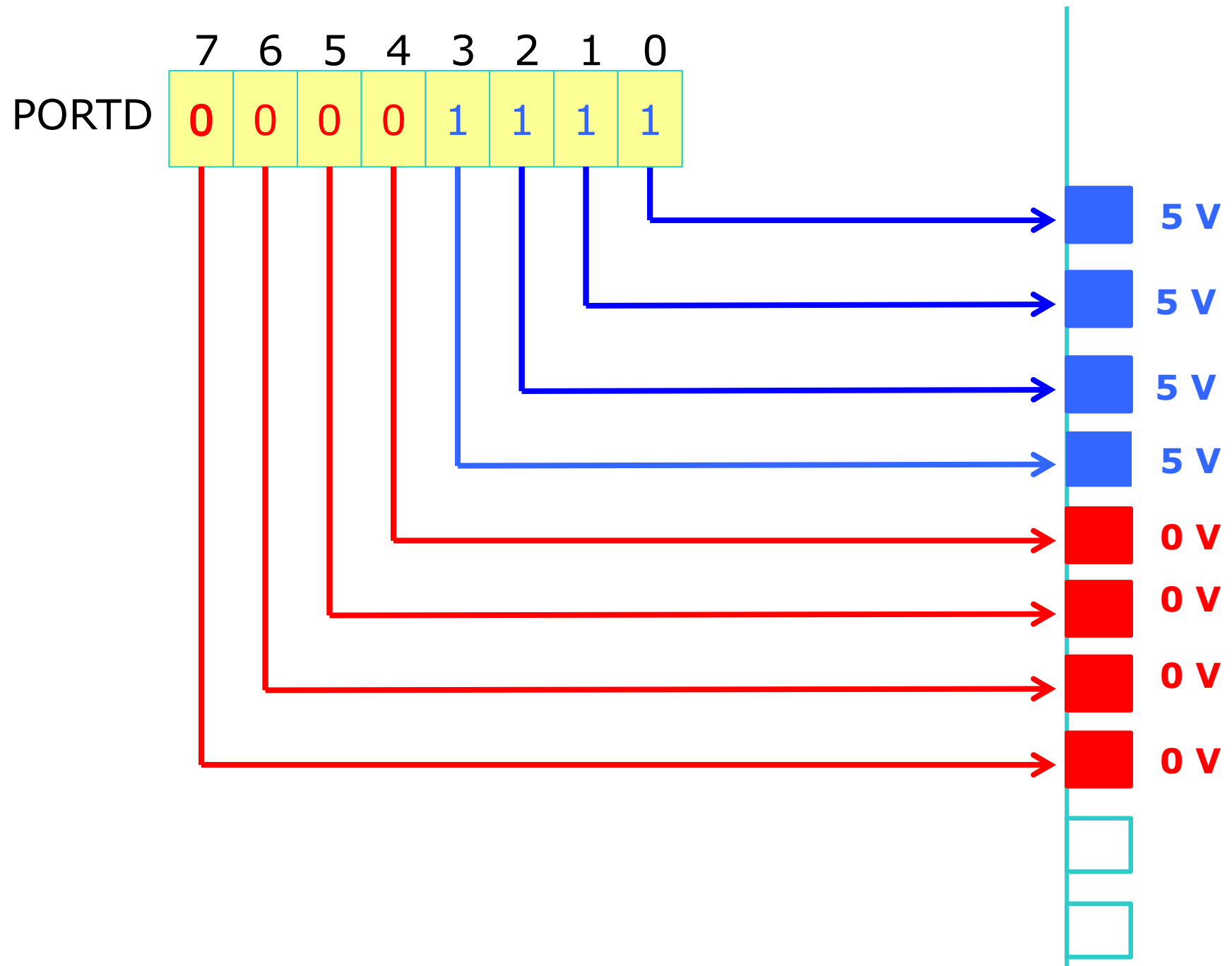


AVR Atmega 328P



PORTD



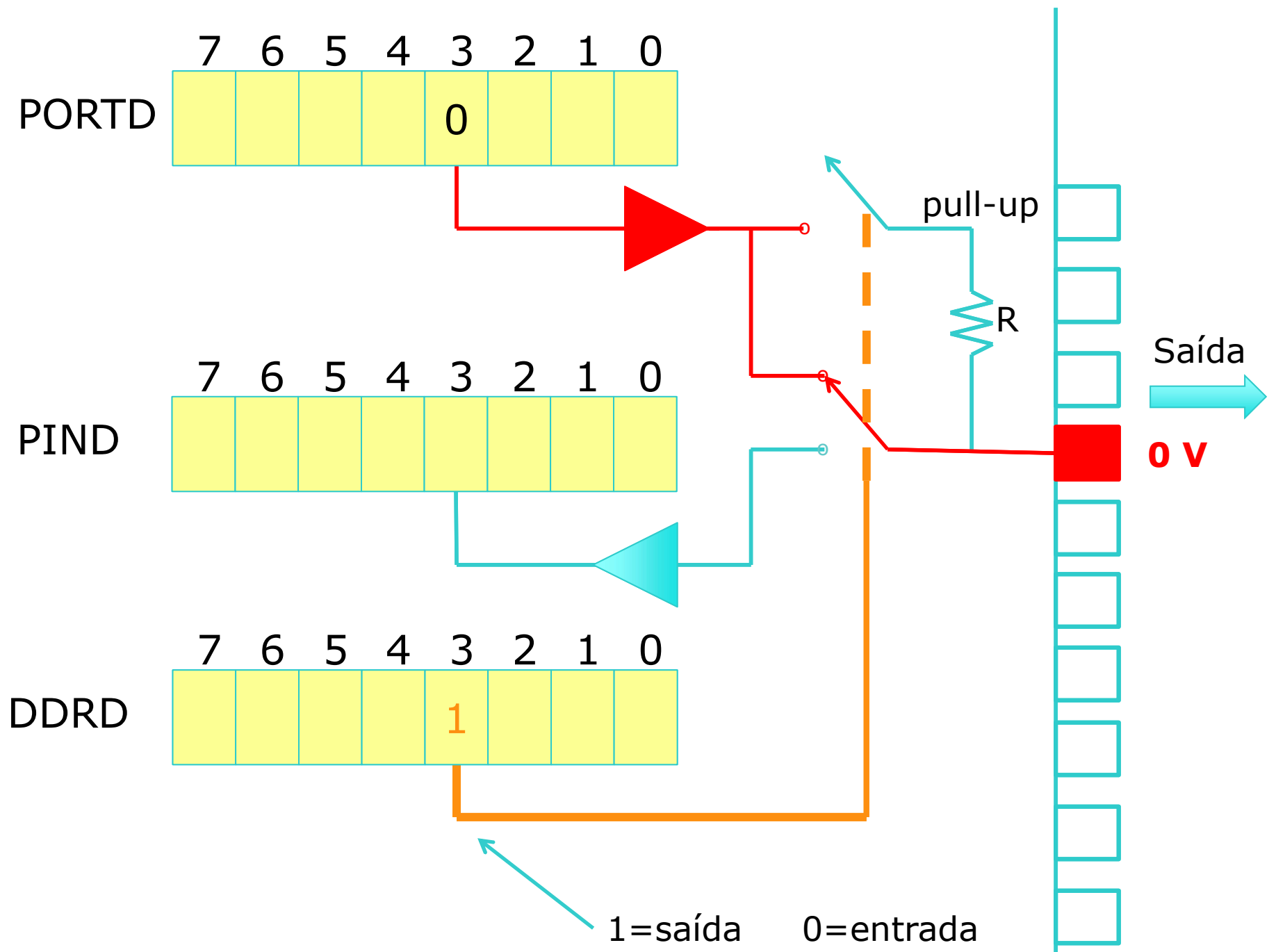


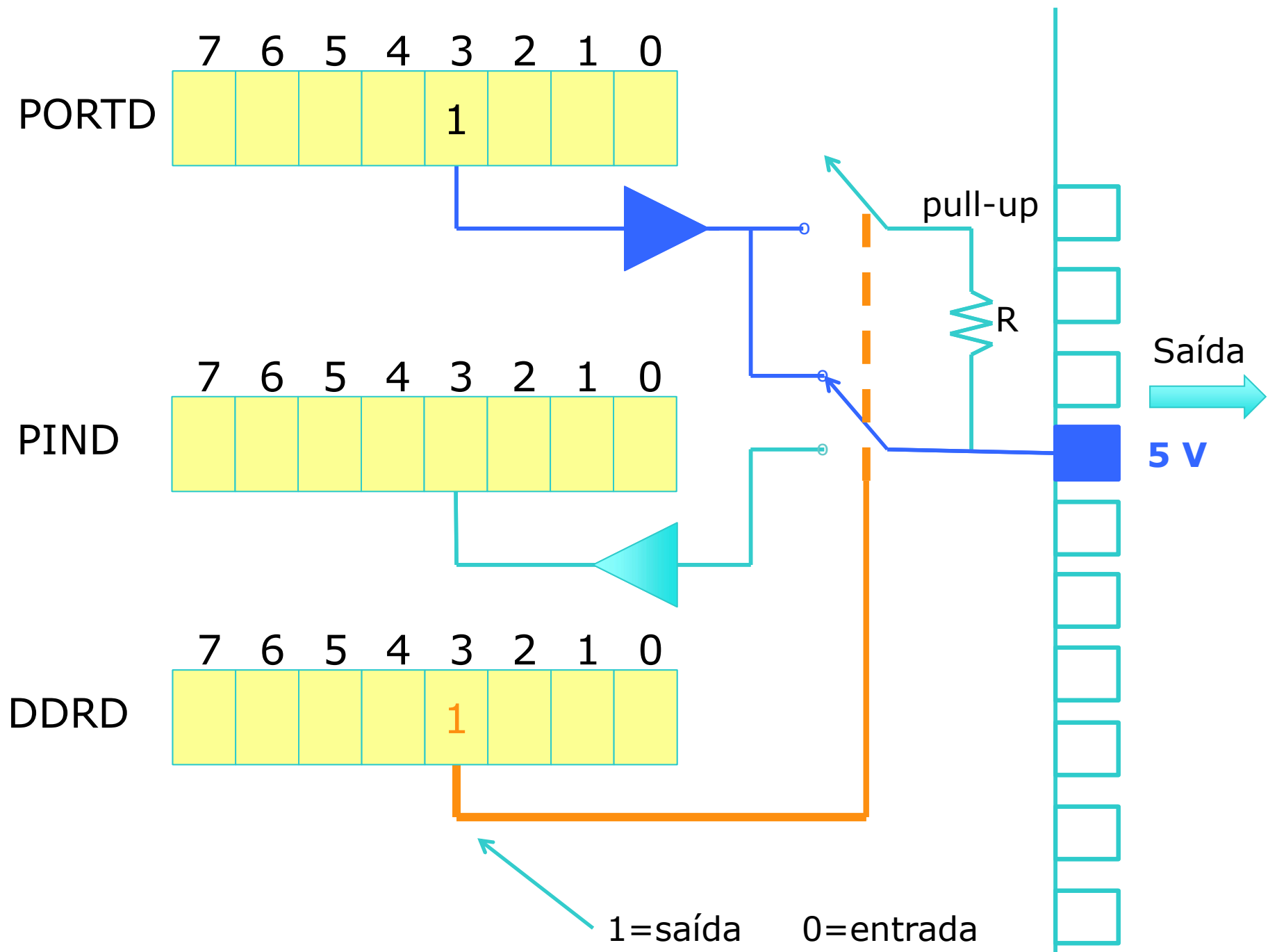
## **REGISTRADORES DE I/O**

**PORTx:** registrador de dados, usado para escrever nos pinos do PORTx em saída ou ativar pull-up entrada.

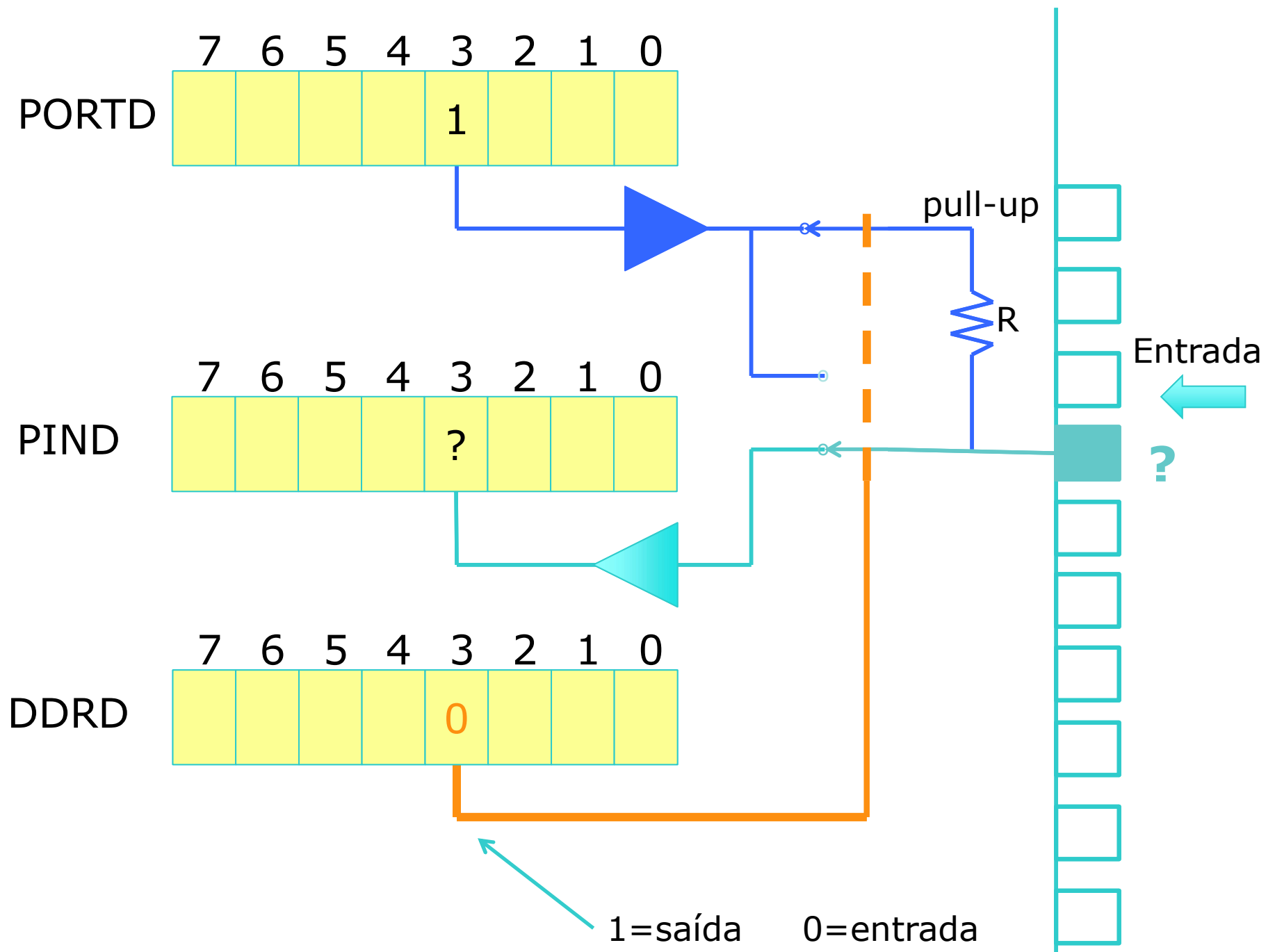
**PINx:** registrador de entrada, usado para ler o conteúdo dos pinos do PORTx.

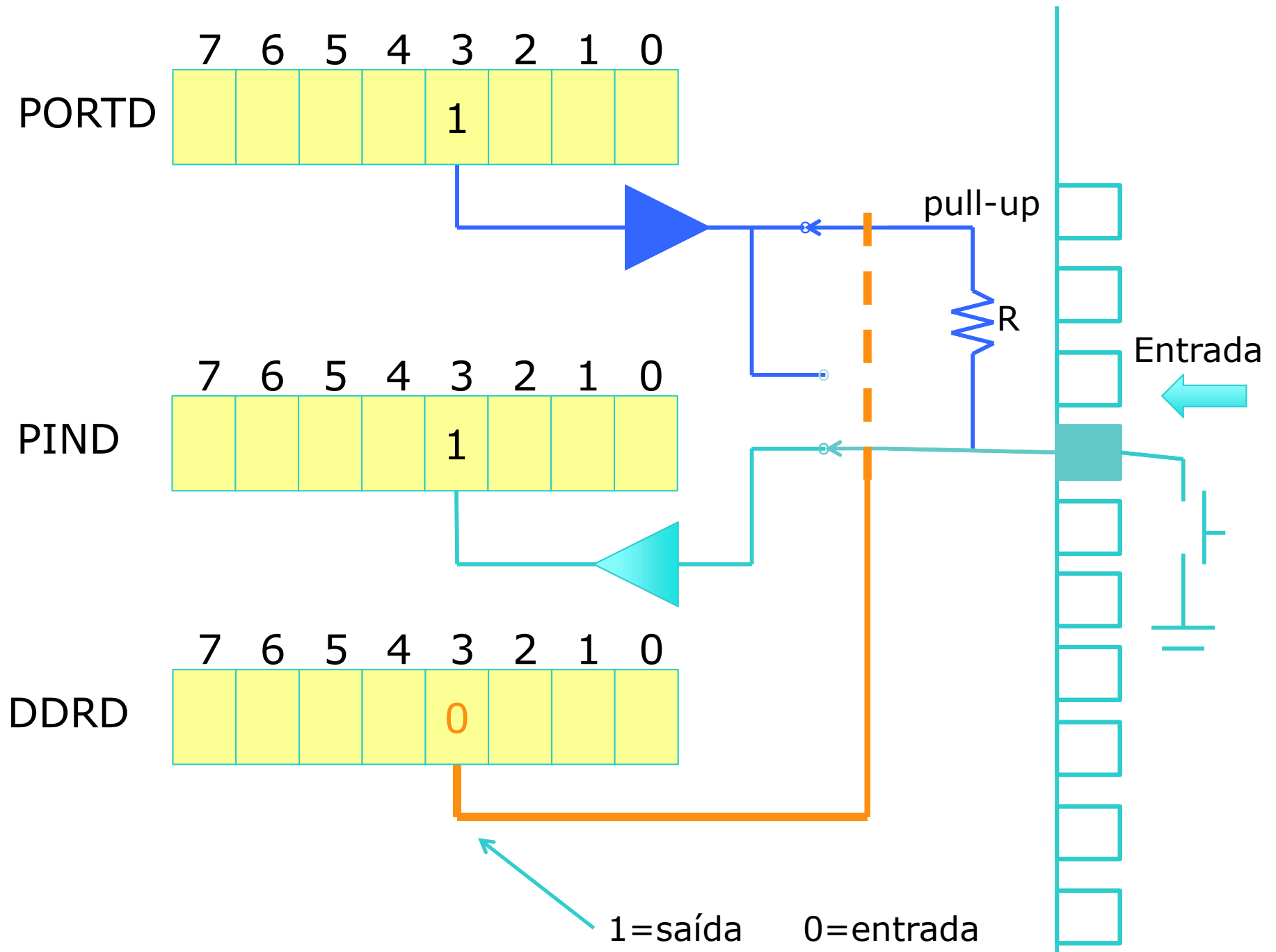
**DDRx:** registrador de direção, usado para definir se os pinos do PORTx são entrada ou saída.







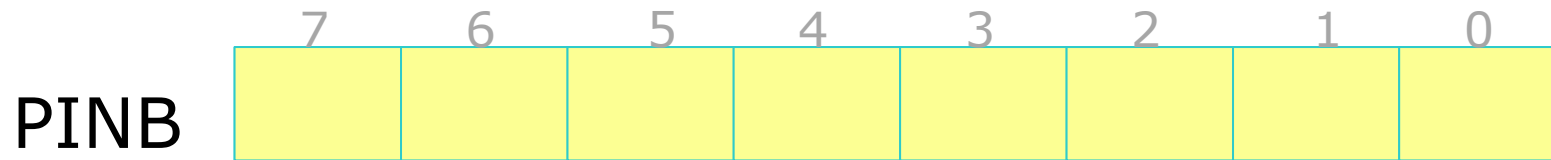




# Manipulação de bits

SBI – Set Bit in I/O

CBI – Clear Bit in I/O



SBI - Seta "1", no bit especificado

SBI PINB, 3

CBI - Limpa "0", no bit especificado

CBI PINB, 3

\*Apenas registradores I/O  
(PORTx, PINx, DDRx)

# Movimentação de Valores

- STS - Copia o valor do registrador para memória

`STS 0x100, R17` → R17 = valor armazenado no endereço \$0x100

- LDS - Copia o valor da memória para o registrador

`LDS R17, 0x100`



R17 = valor armazenado no endereço da memória SRAM \$0x100

STS – Store Direct to SRAM

LDS - Load Direct to SRAM

Testa se bit em LOW (0)



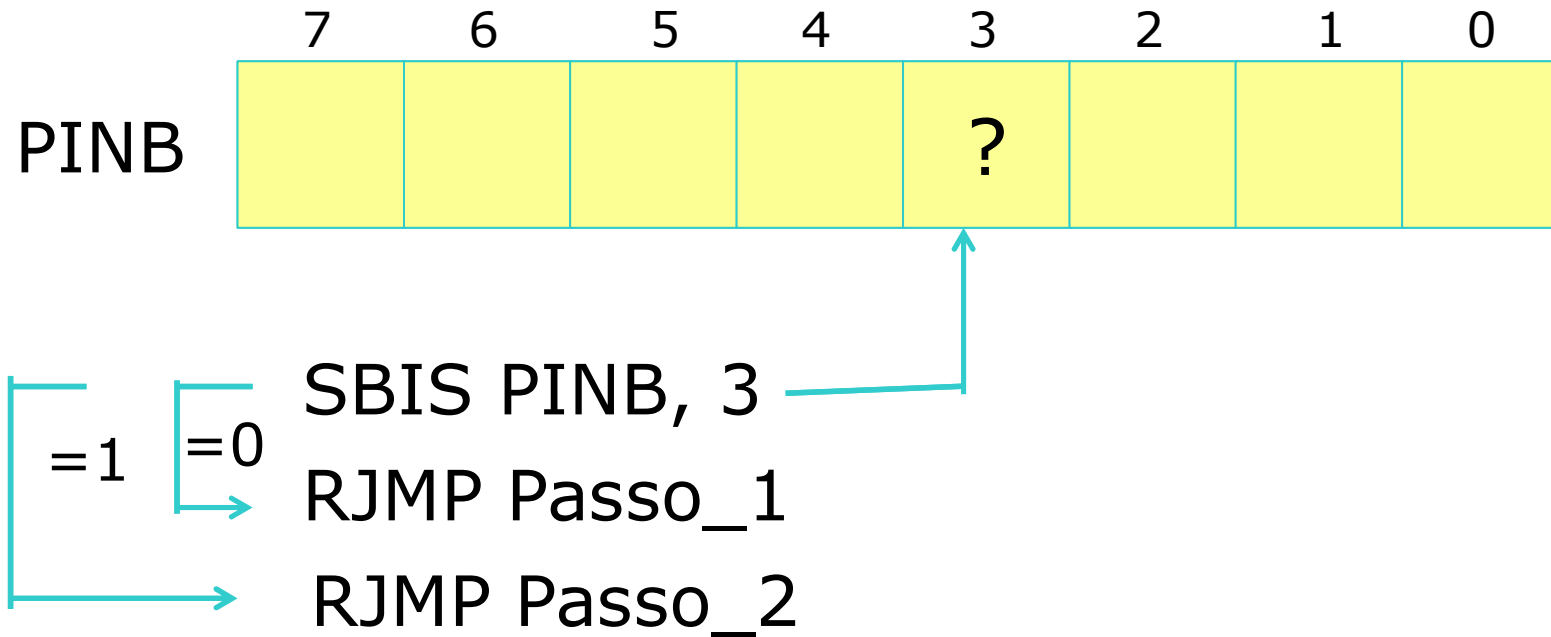
`SBIC PINB, 3`  
`RJMP Passo_1`  
`RJMP Passo_2`

`=0` → `RJMP Passo_2`  
`=1` → `RJMP Passo_1`

SBIC P, b Pula se o bit do registrador de I/O estiver limpo (0)

Consulta o estado de um bit de um registrador de I/O (DDRx, PORTx e PINx). Pula a próxima linha se 0.

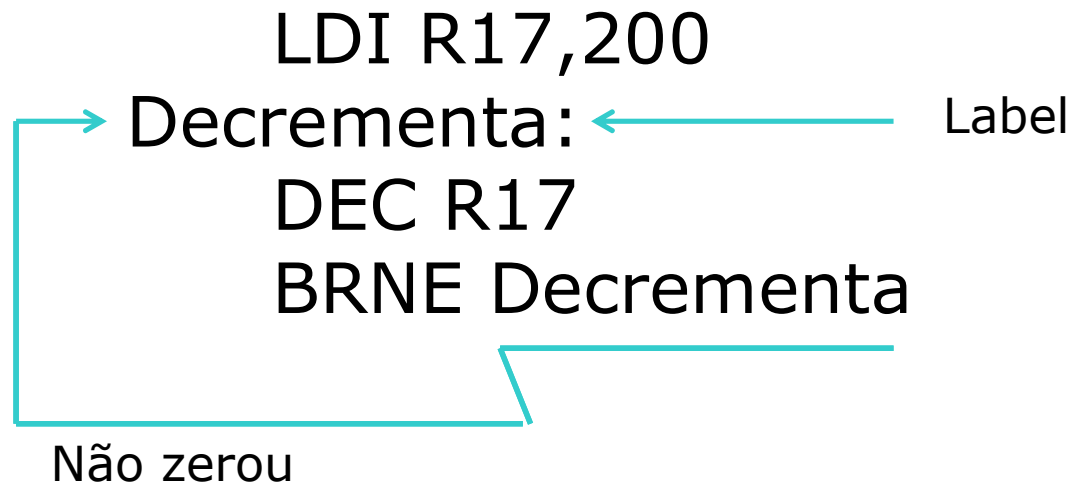
## Testa se bit em HIGH (1)



SBIS P, b Pula se o bit do registrador de I/O estiver ativo (1)

Consulta o estado de um bit de um registrador de I/O (DDRx, PORTx e PINx). Pula a próxima linha se 1.

## Comparando Valores



Verifica se a última operação resultou em valor zero no registrador. Se não ocorreu pula para o label indicado.

## Exercício:

- 1) Escreva um programa em mnemônicos que carrega 15 no Registrador R16 e 34 no Registrador R17, a seguir, soma-se e escreve no R17 da memória.

O Resultado deve aparecer em PORTD.

### Resolução:

R16 ← 15<sub>10</sub>  
R17 ← 34<sub>10</sub>  
R17 ← R17+R16  
PORTD ← R17

```
LDI R19, 0b11111111
OUT DDRD, R19
LDI R16,15
LDI R17,34
ADD R17,R16
OUT PORTD,R17
```



# Exemplo programa assembly AVR

```
//-----  
// EXEMPLO  
//-----  
;-----  
.ORG 0x000 ; endereço de inicio de gravacao memoria Flash
```

```
INICIO: ←  
    LDI R19, 0b11111111 ; carrega R19  
    OUT DDRB,R19 ; configura todos os pinos saida  
    LDI R19, 0b00000000  
    OUT PORTB, R19 ; coloca em zero todos pinos
```

```
←_PRINCIPAL: ← Rótulos  
    SBI PORTB,5  
    CALL ATRASO  
    CBI PORTB,5  
    CALL ATRASO  
    RJMP _PRINCIPAL
```

; rotina de delay.

ATRASO:

LDI R19,80

volta:

DEC R17

BRNE volta

DEC R18

BRNE volta

DEC R19

BRNE volta

RET

comentários



# Bibliotecas, variáveis e Constantes

; Comentários

; Arquivos de definição

#include <nomedoarquivo.inc>

#include "C:\MICRO1\arquivo.inc"

Variáveis: espaço guardar  
valores utilizados no  
programa

; Variaveis

.def tempo = R21 ; registrador GPR

.def temperatura = 0x100 ; memória livre sram

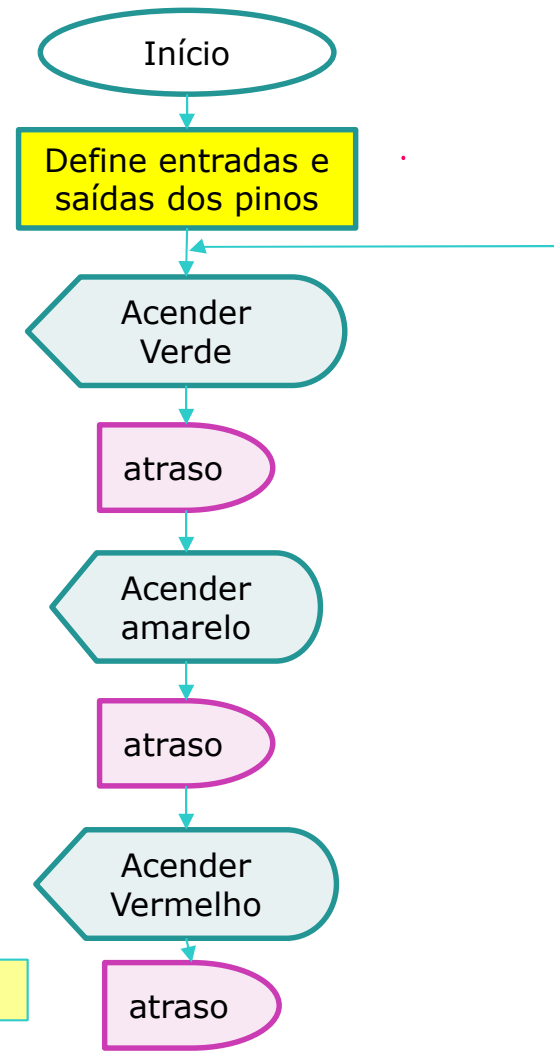
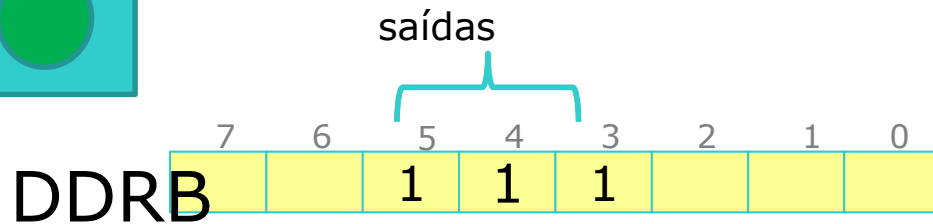
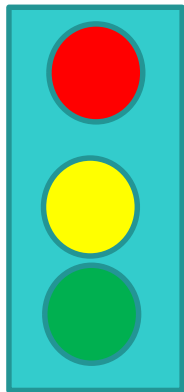
; Constantes

.equ VALOR\_HEXA = 0x0003

.equ VALOR\_BINARIO = 0b00000001

.equ VALOR\_DECIMAL = 123

Exemplo: Elabore um programa Assembly que utilize pinos do PortB como saídas para LEDs que representem um semáforo simples de 3 luzes (vermelho, amarelo e verde) com intervalo de 3 segundos para cada acionamento em loop contínuo.



# Blocos funcionais em ASSEMBLY

Define entradas  
e saídas dos  
pinos

```
; define entradas e saídas  
Início:  
ORG 0x00  
LDI R16, 0b00111000  
OUT DDRB, R16
```

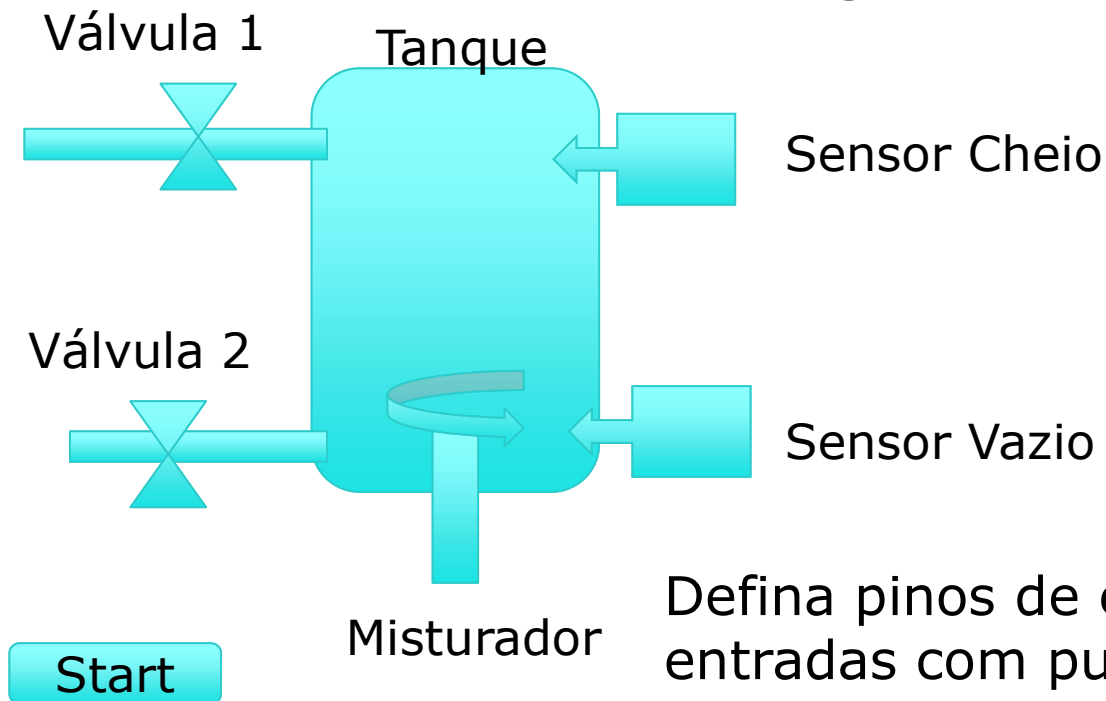
Acender  
Verde

```
Acender_verde:  
SBI PORTB,3 ; verde  
CBI PORTB,4 ; amarelo  
CBI PORTB,5 ; vermelho
```

atraso

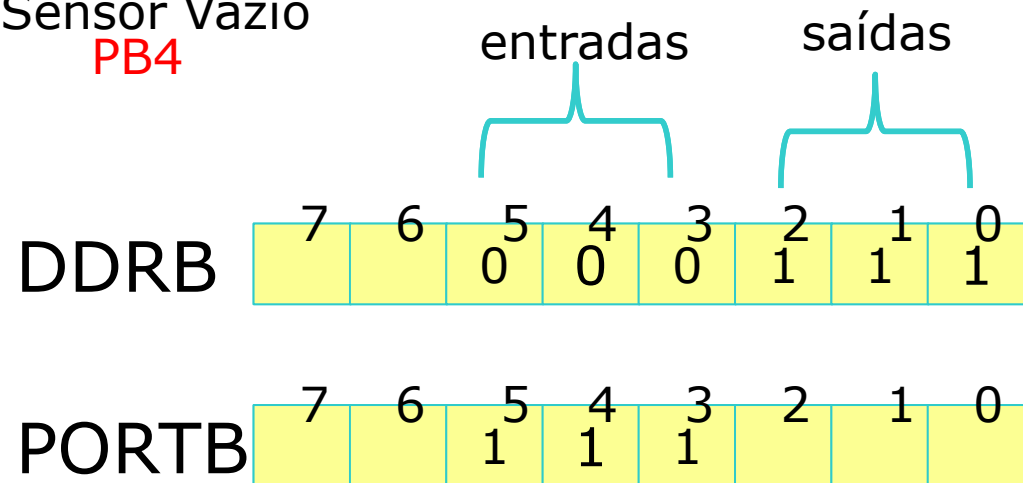
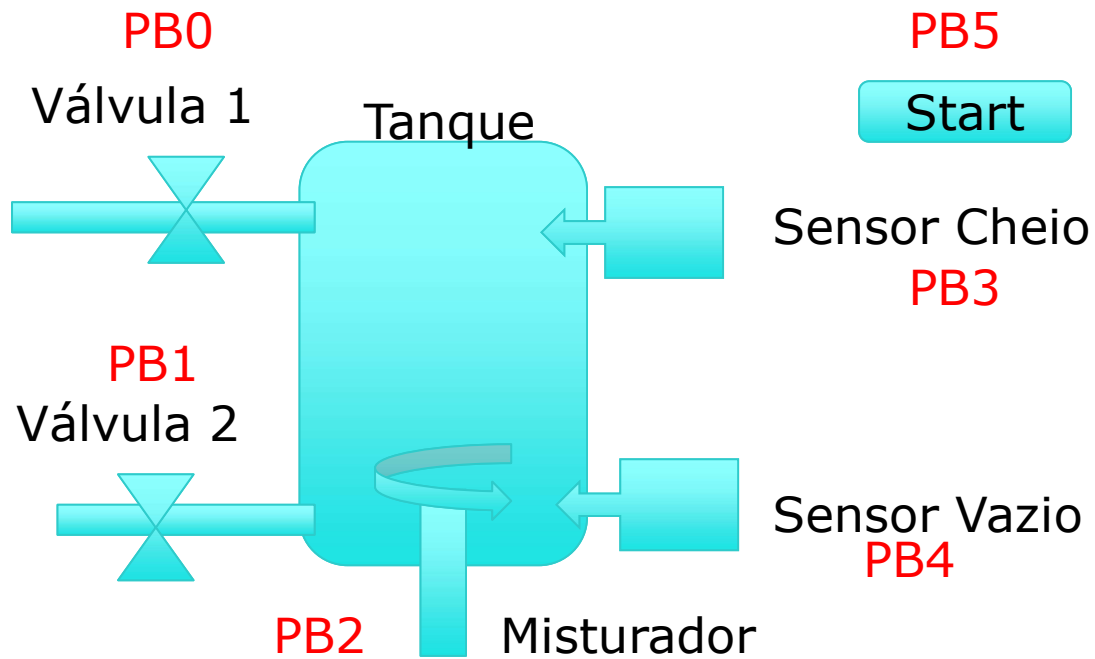
```
; rotina de atraso  
LDI delay_time, 3 ; estabelece valor  
RCALL delay_seconds ; rotina de atraso
```

Exercício: Escreva um programa em mnemônicos que controle o seguinte misturador.



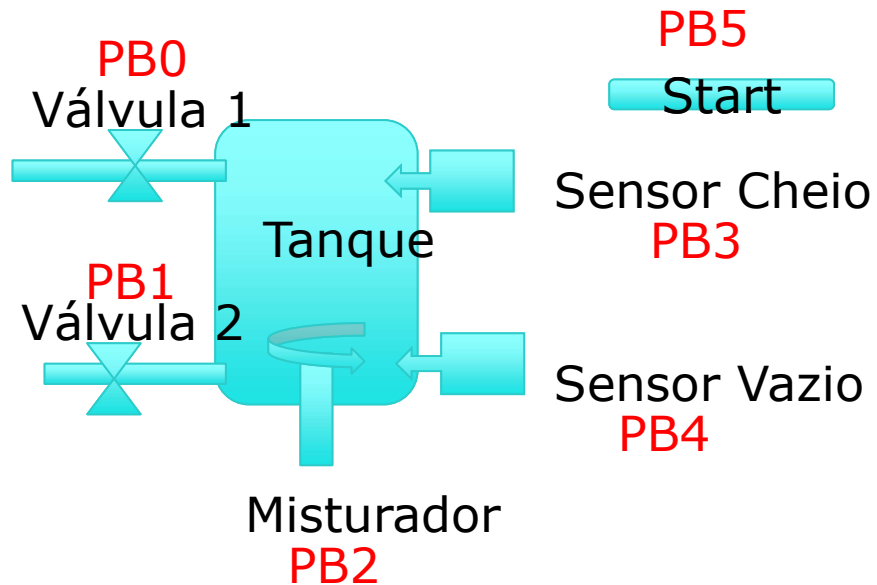
Defina pinos de entrada e saída. As entradas com push button em terra e pull up ativos. O Programa aguarda "Start" ser pressionado, que liga a Válvula 1 até que sensor cheio seja acionado. O misturador é acionado por 2 segundos. Esvazia-se o tanque até o sensor vazio ser acionado. Voltando ao estado inicial. Considere clock 16Mhz.

Resolução:



Ligando pull-up nas entradas

Resolução:



; define entradas e saídas  
Início:

```
ORG 0x00  
LDI R16, 0b00000111  
OUT DDRB, R16  
LDI R16, 0b00111000  
OUT PORTB, R16
```

; testa sensor ou botão  
PRINCIPAL:

```
SBIC PINB,5  
RJMP PRINCIPAL  
RJMP ENCHER
```

; testa sensor cheio  
ENCHER:

```
SBI PORTB,0  
SBIC PINB,3  
RJMP ENCHER  
CBI PORTB,0  
RJMP MISTURAR
```

; liga valvula por tempo  
MISTURAR:

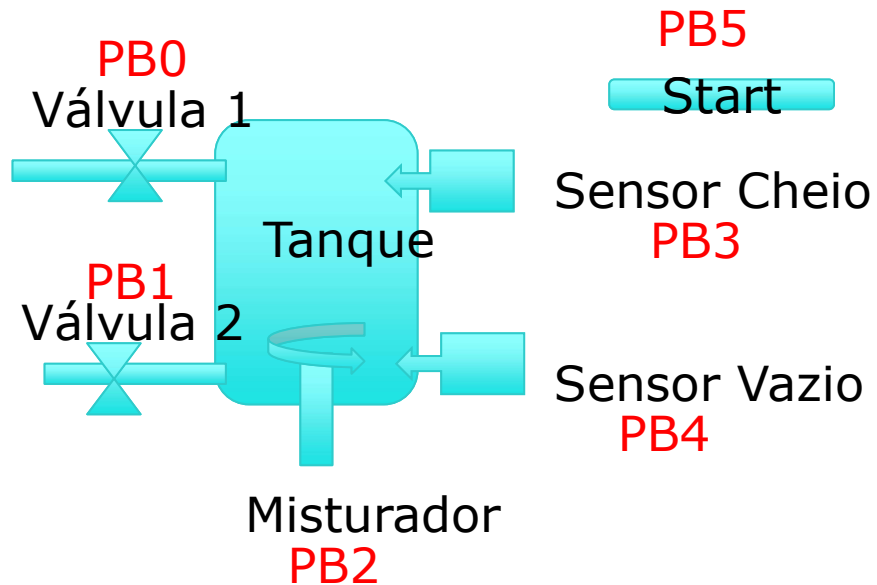
```
SBI PORTB,2  
RCALL ATRASO  
RCALL ATRASO  
CBI PORTB, 2  
RJMP ESVAZIAR
```

; aguarda sensor vazio  
ESVAZIAR:

```
SBI PORTB,1  
SBIC PINB,4  
RJMP ESVAZIAR  
CBI PORTB,1  
RJMP PRINCIPAL
```



## Resolução:



; Liga válvula 2 aguarda sensor vazio  
ESVAZIAR:

```
SBI PORTB,1  
SBIC PINB,4  
RJMP ESVAZIAR  
CBI PORTB,1  
RJMP PRINCIPAL
```

; rotina de atraso 1 segundo.

ATRASSO:

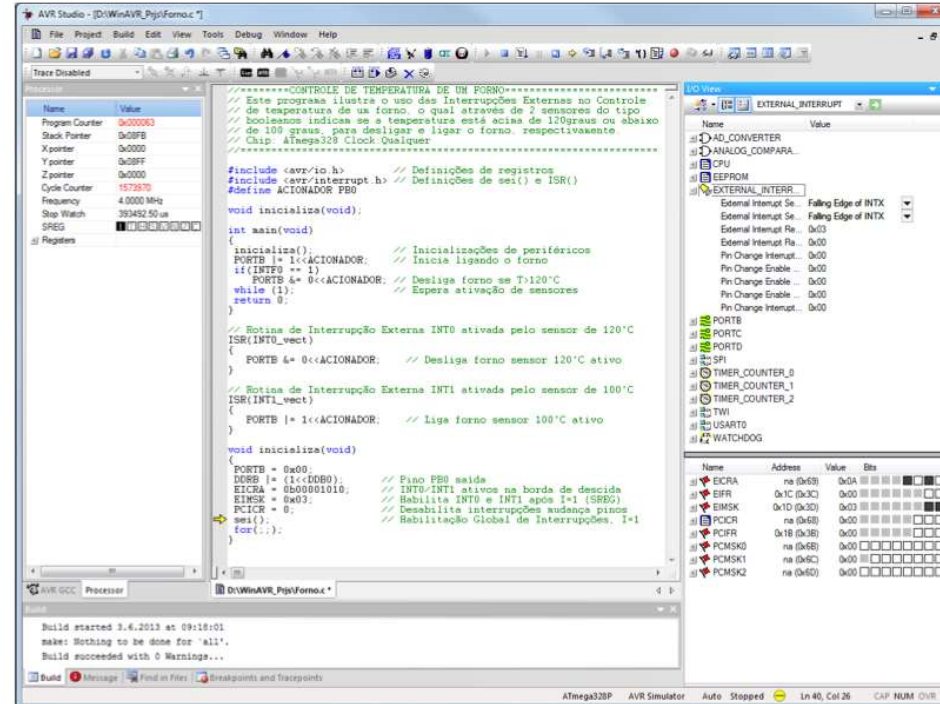
```
LDI R19,80
```

volta:

```
DEC R17  
BRNE volta  
DEC R18  
BRNE volta  
DEC R19  
BRNE volta  
RET
```

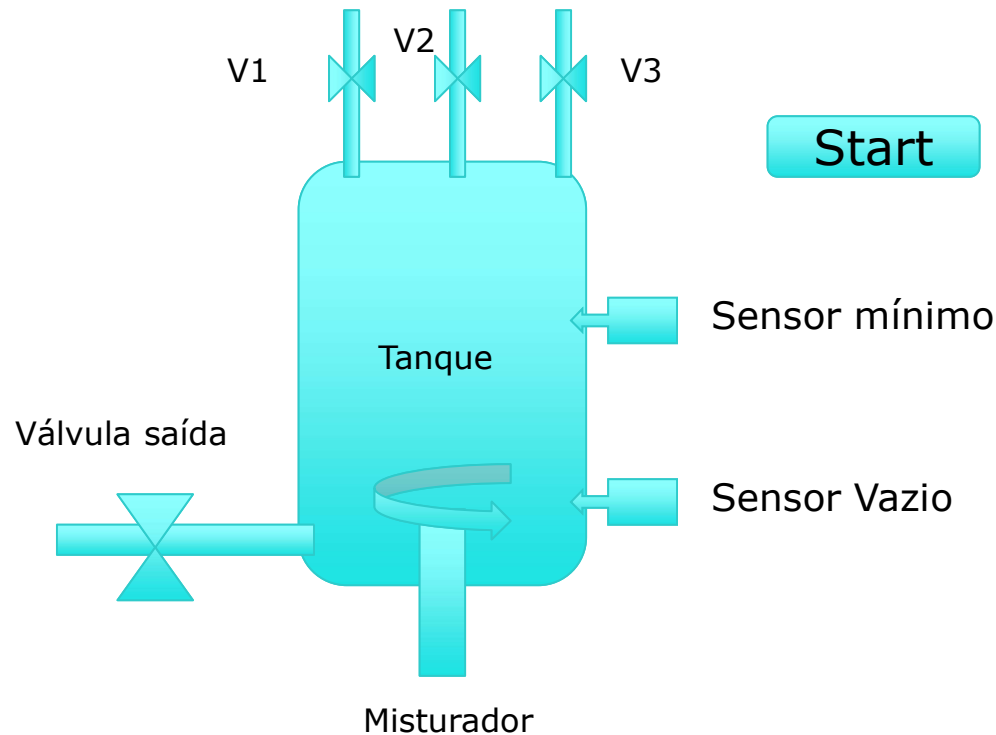
# Ambientes de Desenvolvimento p/ AVR

AVR Studio 4 e Atmel Studio 7 - desenvolvimento em assembly e C, com simulador/depurador. Freeware fornecido pela Atmel. AVR Studio. No desenvolvimento em C pode ser usado com o compilador GNU GCC WinAVR e com o AVR Toolchain da Atmel.



### Exercício:

3) Escreva um programa em mnemônicos que controle o seguinte misturador.



O Programa aguarda "Start" ser pressionado, que liga a V1 durante 3 segundos. Aciona-se V2 durante 2 segundos. O misturador é acionado por 2 segundos. Em seguida aciona-se V3 até que sensor cheio seja acionado(0). Esvazia-se o tanque até o sensor vazio ser acionado(0). Voltando ao estado inicial. Consider clock 16Mhz.

# REGISTRADORES de Entrada/Saída (I/O)

