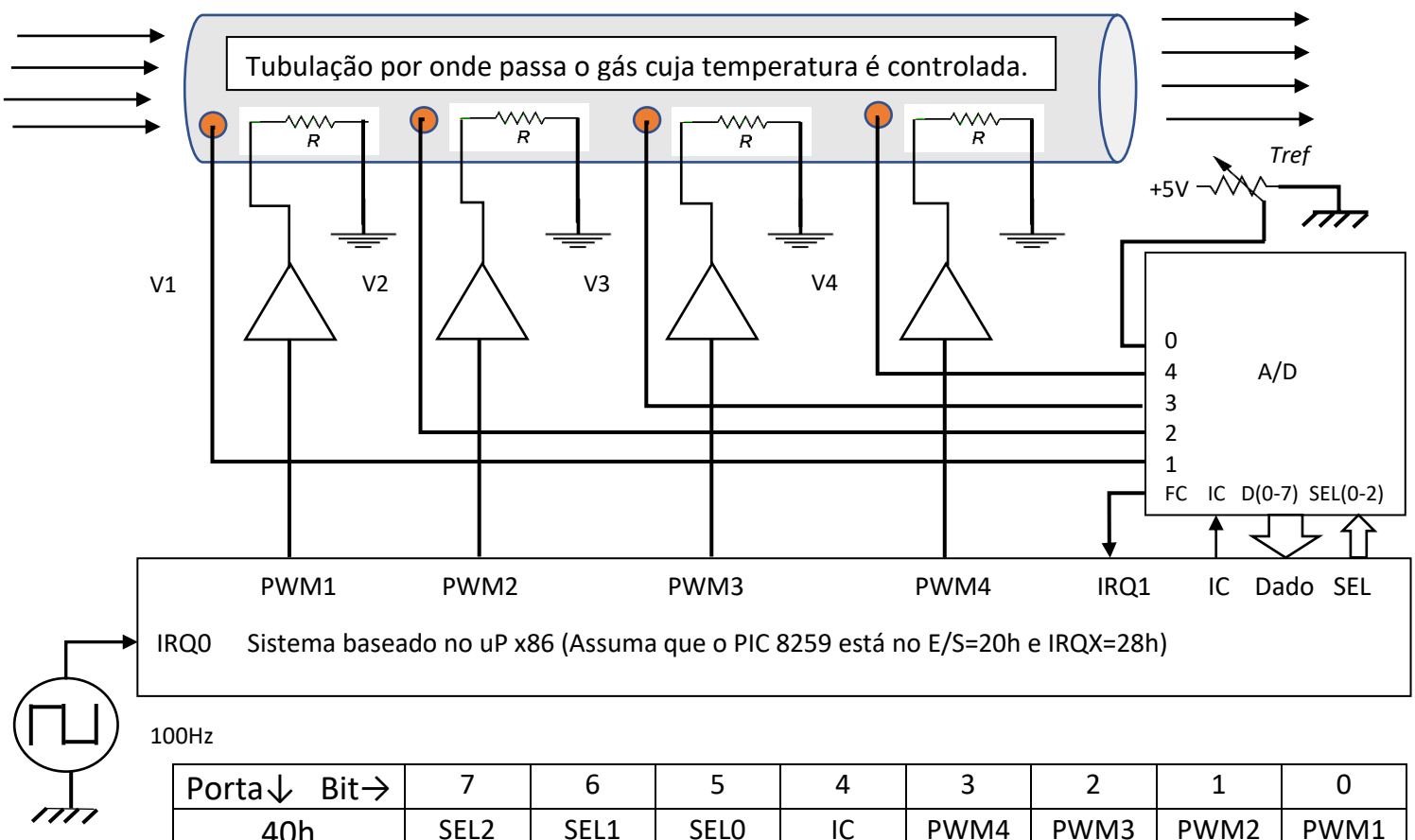
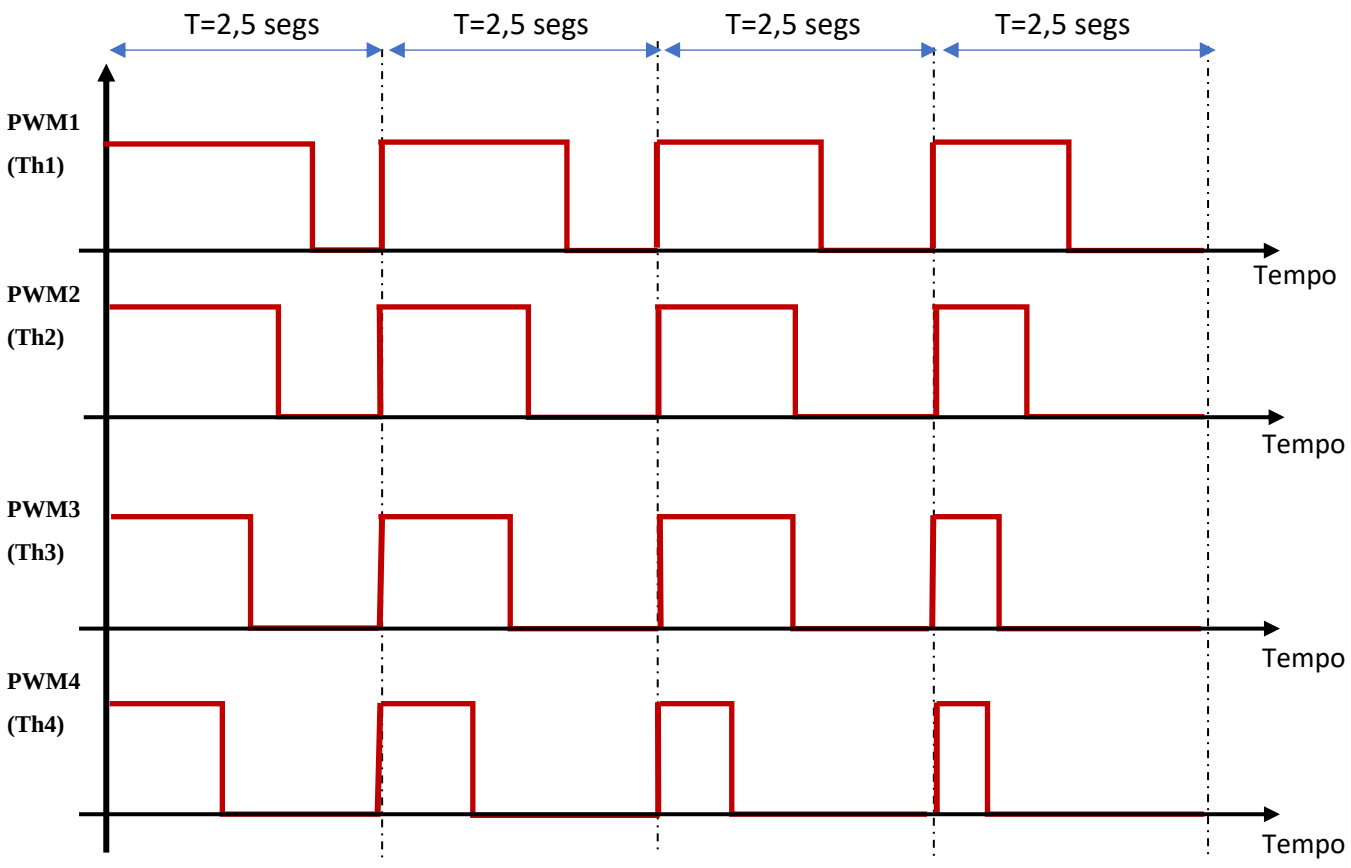
Exercício: Controle PI de um forno industrial de 4 zonas, com saídas em PWM.

Deseja-se construir um controlador de temperatura para uma chapa, usando um microprocessador da família MSC86, para um ensaio de poluição, como mostrado no desenho em anexo. O fluxo de um gás deve passar ao longo da chapa, que precisa ser mantida a uma temperatura constante $\mathbf{T_{ref}}$. Para melhorar o controle, a chapa foi dividida em 4 zonas, tendo cada uma delas um controle independente. Um amplificador de potência, controlado por um sinal **PWM**, alimenta uma resistência para aquecer a zona correspondente. Um sensor de temperatura dá uma saída analógica em volts **vi** ($1 \leq i \leq 4$) de 0 a 5V, correspondente a uma temperatura de 0 a 255 ºC. Cada tensão **vi** é conectada na *i*-ésima entrada de um conversor A/D. Este conversor A/D é de 8 bits e tem um sinal de início de conversão **IC** (transição positiva) e um sinal de fim de conversão FC (transição positiva). O sinal **FC** deve ser a entrada de interrupção **IRQy**. A entrada 0 do conversor A/D é alimentada por um potenciômetro (0 a 5V) para definir a temperatura de referência $\mathbf{T_{ref}}$. Uma entrada digital do conversor A/D, $\mathbf{SEL_{0\text{-}2}}$, seleciona qual canal do conversor ($0=000_2$ e $4=100_2$ sendo $\mathbf{SEL_2}$ a mais significativo) será convertido. As saídas $\mathbf{PWM_{1\text{-}4}}$ têm período de 2,5 segundos e um tempo de nível alto ($1 \leq$ **Thi** ≤ 250, em centésimos de segundo) definido na folha em anexo. Uma onda quadrada de 100Hz deve ser a entrada de interrupção **IRQx**. A cada início do período de 2,5 segundos, o controlador deverá calcular o **Thi** para cada zona i e gerar a onda **PWM** correspondente. $erro_i(n) = Tref(n) - Tmed_i(n)$, onde $Tmed_i(n)$ é a temperatura medida pelo i-ésimo sensor (representado abaixo por bolas laranjas).

$$Thi(n) = 10.erro_i(n) + {}^{1}/_{100}\sum erro_i(n).\ Assuma\ \left|\sum erro_i(n)\right| \leq 10.000\,.$$



Porta↓ Bit→	7	6	5	4	3	2	1	0
40h	SEL2	SEL1	SEL0	IC	PWM4	PWM3	PWM2	PWM1
80h	D7	D6	D5	D4	D3	D2	D1	D0



```
segment code
..start:
; Exercício Controle, por PWM, de um forno de 4 zonas.
; Portas:
;*****************************************************************
; 20h-> PIC8259, IRQx>IRQy e IRQx=28h
; 40h-> saída, [sel2 sel1 sel0 IC PWM4 PWM3 PWM2 PWM1].
; 80h-> entrada, [D7 D6 D5 D4 D3 D2 D1 D0]
;*****************************************************************
; Variáveis declaradas
; Temp    dw  0
; Tref    dw  0
; espelho db   0
; th      db   0, 0, 0, 0
; erro    dw  0, 0, 0, 0
; s_erro  dw  0, 0, 0, 0
; tique   db   0
; adc_conv          db   0
;*****************************************************************

cli
mov     ax,data
mov     ds,ax
mov     ax,stack
mov     ss,ax
mov     sp,stacktop

;       Programando o controlador de interrupções: PIC8259
mov     al, 13h; programando registrador ICW1
out     20h, al; (A0=0, por isso o endereço é igual a 20h)
mov     al, 28h; programando registrador ICW2
out     21h, al; (A0=1, por isso o endereço é igual a 21h)
mov     al, 1; programando registrador ICW4
out     21h, al; (A0=1, por isso o endereço é igual a 21h)
mov     al, 11111100b; programando registrador OCW1. Aceita IRQ0 e IRQ1
out     21h, al; (A0=1, por isso o endereço é igual a 21h)

;       Preenchendo a Tabela de Interrupções com as
;       localizações em memória RAM das rotinas de interrupções (ISR's) "pwm" e "le_adc"
xor     ax, ax
mov     ES, ax
mov     word[ES:28h*4], pwm
mov     word[ES:28h*4+2], cs
mov     word[ES:29h*4], le_adc
mov     word[ES:29h*4+2], cs
sti
inicio:
mov     byte[tique], 0
mov     al, 00001111b; observe que 1<=thi<=250; começa com IC=0 (bit 4 da porta 40h) e SEL0-2=000b (bits 5 a 7 porta 40h)
out     40h, al
mov     byte[espelho], al
xor     si, si ; SI é usado para indexar os vetores "erro" e "s_erro", do tipo word.
xor     di, di ; DI é usado para indexar o vetor "th", do tipo byte.
call    dispara_adc
mov     ax, word[Temp]
mov     word[Tref], ax
mov     cx, 4
volta:
call    dispara_adc
call    calcula_pid
add     si, 2
inc     di
loop    volta
volta_tique:
cmp     byte[tique], 250
jb      volta_tique
jmp     inicio
```

```
;********************** Rotinas e ISR´s*********************************
;***************************************************************************
le_adc:; Essa é a rotina ISR de tratamento da interrupção de hardware IRQ1
    push  ax
    in    al, 80h
    xor   ah, ah
    mov   word[Temp], ax
    mov   byte[adc_conv], 1
    mov  al, 20h ; al = 20h  libera o controlador de interrupções, colocado na porta 20h, para aceitar novas interrupções.
    out   20h, al ; Ao colocar al=20 na porta de E/S no endereço 20h, programa-se OCW2 = 20h no controlador de interrupção.
    pop   ax
iret;
*****************************************************************************************************
dispara_adc:
    push  ax
    mov   byte[adc_conv], 0
    mov   al,byte[espelho]
    ; Gerando a borda de subida para disparar o ADC.
    and   al, 11101111b; Faz IC=0.
    out   40h, al
    or    al, 00010000b; Faz IC=1 => essa transição de IC dispara o A/D.
    out   40h, al
    espera:
    cmp   byte[adc_conv], 0
    jz    espera;
    add   al, 00100000b; Incrementa SEL0-2 para a próxima leitura do ADC.
    mov   byte[espelho], al
    pop   ax;
ret
    ;*****************************************************************************************************
calcula_pid:
    push  ax
    mov   ax, word[Tref]
    sub   ax, word[Temp] ; gera erro:  erroi(n)= Tref (n) – Temperaturai(n), em que 1<=i<=4 (4 zonas) e n indexa a o instante n.
    mov   word[erro+si], ax
    add   word[s_erro+si], ax
    mov   bx, 10
    imul  bx; Faz DX:AX = AX*BX
    push  ax ; empilha o termo 10*erro(i)
    cmp   word[s_erro+si], 10000
    jng   testa_neg10000
    mov   word[s_erro+si], 10000
    jmp   segue_calcula_pid
    testa_neg10000:
    cmp   word[s_erro+si], -10000
    jnl   segue_calcula_pid
    mov   word[s_erro+si], -10000
    segue_calcula_pid:
    mov   bx, 100
    mov   ax, word[s_erro+si]
    cwd   ; Converte word (em AX) para double word (DX:AX) levando-se em conta o sinal de AX;
    idiv  bx; (DX:AX)/BX → Quociente em AX e resto da divisão em BX
    pop   bx ; desempilha o AX empilhado na linha 106
    add   ax, bx
    cmp   ax, 250
    jng   testa_limite_inferior_thi
    mov   ax, 250
    jmp   fim_pid
    testa_limite_inferior_thi:
    cmp   ax, 1
    jnl   fim_pid
    mov   ax, 1
    fim_pid:
    mov   byte[th+di], al
    pop   ax
ret
;*****************************************************************************************************
```

```
;*****************************************************************************************************
pwm ; Essa é a rotina de tratamento da interrupção de hardware IRQ0
      push  ax;
      push  bx;
      inc     byte[tique];
      mov   al, byte[espelho]
      mov   bl, byte[th];
      cmp   byte[tique], bl
      jne     segue_pwm2;
      and     al, 11111110b;
      segue_pwm2:
        mov  bl, byte[th+1];
        cmp  byte[tique], bl
        jne  segue_pwm3;
        and al, 11111101b;
      segue_pwm3:
        mov  bl, byte[th+2];
        cmp  byte[tique], bl
        jne  segue_pwm4;
        and al, 11111011b;
      segue_pwm4:
           mov  bl, byte[th+3];
           cmp  byte[tique], bl
           jne  segue_pwm;
           and  al, 11110111b;
      segue_pwm:
        mov  byte[espelho], al;
        out  40H, al;
        mov  al, 20h ; al = 20h  libera o controlador de interrupções, colocado na porta 20h, para aceitar novas interrupções.
        out  20h, al ; Ao colocar al=20 na porta de E/S no endereço 20h, programa-se OCW2 = 20h no controlador de interrupção.
        pop bx;
        pop ax;
iret
;*****************************************************************************************************
segment data
Temp      dw        0
Tref      dw        0
espelho   db        0
th        db        0, 0, 0, 0
erro      dw        0, 0, 0, 0
s_erro    dw        0, 0, 0, 0
tique     db        0
adc_conv            db          0

;*************************************************************************
segment stack stack
resb      512
stacktop:
```