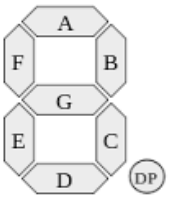

Fig. 1: Denominações dos LEDs do display de 7 segmentos.

*ITENS ERRADOS **NÃO ANULAM** ITENS CORRETOS. O cartão de respostas e o funcionamento dos principais registradores se encontram no final da prova. Em todas as questões, considere que o MCLK e SMCLK foram configurados para funcionarem a 1 MHz e que o Watchdog Timer está desligado. Boa prova.*

1. (1.25 pontos)

```
1  #define BTN BIT3
2
3  void Configura_Dado(void)
4  {
5    P1DIR &= ~BTN;
6    P1OUT |= BTN;
7    P1REN |= BTN;
8    TACCR0 = 5;
9    TACTL = TASSEL_2 + ID_0 + MC_1;
10 }
11
12 int Retorna_Dado(void)
13 {
14   while((P1IN&BTN)~=0);
15   while((P1IN&BTN)==0);
16   return(TAR+1);
17 }
```

O código acima implementa a escolha do valor de um dado digital, baseado no Timer A e em um botão conectado ao pino P1.3. Para CADA item a seguir, indique se ele é verdadeiro ou falso.

A A função `Configura_Dado()` só precisa ser chamada uma vez antes da função `Retorna_Dado()` para o dado digital funcionar corretamente.

B A função `Retorna_Dado()` realiza o *debounce* do botão.

C O usuário deveria ser capaz de contar passos de 10^{-6} segundos para tirar sempre o valor que desejasse.

D O valor do dado é determinado quando o usuário pressiona o botão.

2. (1.25 pontos) **Considere uma carga de 12 V que necessita de** $(d+501)/1000$ **A para funcionar, aonde** d **corresponde aos três últimos dígitos da sua matrícula. Por exemplo, para o aluno com matrícula 12/3456789, então** $d = 789$ **e a carga é de** $(789 + 501)/1000 = 1,29$ **A. Calcule a resistência necessária e a topologia (normal ou par Darlington) para o MSP430 controlar esta carga com transistores bipolares de junção com** $V_{BE} = 0,7$ **V e** $\beta = 150$**. O MSP430 opera com** $V_{CC} = 3$ **V e corrente máxima de** 6 **mA por pino digital.**

3. (1.25 pontos)

```
1  void Config_TimerA(void)
2  {
3    P1DIR |= BIT6;
4    P1SEL |= BIT6;
5    P1SEL2 &= ~BIT6;
6    TACCR0 = 1000-1;
7    TACCR1 = 0;
8    TACCTL1 = OUTMOD_7;
9    TACTL  = TASSEL_2 + ID_0 + MC_1 + TAIE;
10   _BIS_SR(GIE);
11 }
12
13 interrupt(TIMER0_A0_VECTOR) TA_ISR(void)
14 {
15   TACCR1 = (TACCR1<(TACCR0-10)) ? TACCR1+10 : 0;
16   TACTL &= ~TAIFG;
17 }
```

O código acima implementa um LED de brilho alterável através do Timer A. Para CADA item a seguir, indique se ele é verdadeiro ou falso.

A O ciclo completo de mudança do brilho do LED dura 1 s.

B Após a chamada da função `Config_TimerA()`, a frequência com que o LED pisca não alterada.

C O LED pisca a uma frequência de 100 Hz.

D O brilho do LED aumenta e diminui gradativamente.

4. (1.25 pontos)

```
1  #define BTN BIT3
2
3  volatile char e = 0;
4
5  void Config_Botao(void)
6  {
7    P1REN |= BTN;
8    P1OUT |= BTN;
9    P1DIR &= ~BTN;
10   P1IES |= BTN;
11   P1IE  |= BTN;
12   P1IFG = 0;
13   _BIS_SR(GIE);
14 }
15
16 interrupt(PORT1_VECTOR) P1_ISR(void)
17 {
18   volatile unsigned int i=60000;
19   e ^= 1;
20   if(e==0)
21     TACTL = MC_0;
22   else
23   {
24     TACCR0 = 2500-1;
25     TACTL  = TASSEL_2 + ID_0 + MC_1;
26   }
27   while((P1IN & BTN)==0);
28   while(i--);
29   P1IFG = 0;
30 }
```

O código acima implementa o controle do Timer A via um botão ligado ao pino P1.3. Para CADA item a seguir, indique se ele é verdadeiro ou falso.

A O processador não pode ser interrompido enquanto ele está preso nos laços `while()` das linhas 27 e 28.

B O Timer A só é acionado quando o usuário solta o botão.

C A função `P1_ISR()` realiza o *debounce* do botão.

D Uma variável global define se o Timer A deve ser desligado ou ligado para contar períodos de $2,5$ ms.

5. (1.25 pontos)

```
1  void Hello1(void)
2  {
3    volatile unsigned int i;
4    P1DIR |= BIT6;
5    for(;;)
6    {
7      i=300;
8      while(i--);
9      P1OUT ^= BIT6;
10   }
11 }
12
13 void Hello2(void)
14 {
15   P1DIR |= BIT6;
16   TACCR0 = 62500-1;
17   TACTL = TASSEL_2 + ID_3 + MC_1;
18   for(;;)
19   {
20     while((TACTL&TAIFG)==0);
21     P1OUT ^= BIT6;
22     TACTL &= ~TAIFG;
23   }
24 }
25
26 void Hello3(void)
27 {
28   P1DIR |= BIT6;
29   P1SEL |= BIT6;
30   P1SEL2 &= ~BIT6;
31
32   TACCR0 = 62500-1;
33   TACCR1 = 62500/2;
34   TACCTL1 = OUTMOD_7;
35   TACTL = TASSEL_2 + ID_3 + MC_1;
36 }
```

O código acima apresenta 3 possíveis funções Olá Mundo com o MSP4302553. Considere que um LED foi conectado ao pino P1.6, em série com uma resistência adequada. Para CADA item a seguir, indique se ele é verdadeiro ou falso.

A A função `Hello3()` utiliza o Timer A em modo de comparação.

B Ao se chamar a função `Hello1()`, é possível ver o LED piscando.

C As 3 funções funcionam em loop infinito.

D As 3 funções só fazem o LED piscar se o pino P1.6 for conectado ao anodo do LED.

E Ao se chamar a função `Hello2()`, o LED pisca com frequência de 2 Hz.

F Ao se chamar a função `Hello3()`, o LED pisca com frequência de 2 Hz.

6. (1.25 pontos)

```
1  #define PIN_1 BIT0
2  #define PIN_2 BIT1
```

```
3  #define PIN_3 BIT2
4  #define PIN_4 BIT4
5  #define ALL_PINS (PIN_1+\
6        PIN_2+PIN_3+PIN_4)
7
8  void Charlie_LED_On(
9      volatile char pin_pos,
10     volatile char pin_neg)
11 {
12   P1DIR &= ~(PIN_1+PIN_2+PIN_3+PIN_4);
13   P1OUT |=  pin_pos;
14   P1OUT &= ~pin_neg;
15   P1DIR |=  (pin_pos+pin_neg);
16 }
17
18 void Charlieplex(volatile char LEDs)
19 {
20   volatile int j;
21   volatile pin_pos[ ] = {
22     PIN_1,PIN_1,PIN_1,
23     PIN_2,PIN_2,PIN_2,
24     PIN_3,PIN_3,PIN_3,
25     PIN_4,PIN_4,PIN_4};
26   volatile pin_neg[ ] = {
27     PIN_2,PIN_3,PIN_4,
28     PIN_1,PIN_3,PIN_4,
29     PIN_1,PIN_2,PIN_4,
30     PIN_1,PIN_2,PIN_3};
31   for(j=0; j<12; j++)
32     if((LEDs & (1<<j))~=0)
33       Charlie_LED_On(
34         pin_pos[j],
35         pin_neg[j]);
36 }
```

O código acima controla LEDs conectados por charlieplexing. Para CADA item a seguir, indique se ele é verdadeiro ou falso.

A Na chamada à função `Charlieplex()`, cada bit na variável de entrada `LEDs` indica se o LED correspondente será aceso ou não.

B A linha 12 do código acima impede que os LEDs errados sejam acesos nas linhas 13 e 14.

C A função `Charlieplex()` acende mais de um LED por vez.

D Para os LEDs piscarem a uma frequência f, a função `Charlieplex()` deve ser chamada f vezes por segundo.

7. (1.25 pontos)

```
1  #include <msp430g2553.h>
2  #define BTN BIT3
3  #define DSP_1 BIT0
4  #define DSP_2 BIT1
5  #define DSP_3 BIT2
6  #define DSP_4 BIT4
7  #define DSPS  (DSP_1+DSP_2+\
8          DSP_3+DSP_4)
9  #define SEG_A BIT0
10 #define SEG_B BIT1
11 #define SEG_C BIT2
12 #define SEG_D BIT3
13 #define SEG_E BIT4
14 #define SEG_F BIT5
15 #define SEG_G BIT6
16 #define SEG_DP BIT7
17 #define OITO   (SEG_A+SEG_B+\
18         SEG_C+SEG_D+\
19         SEG_E+SEG_F+SEG_G)
20 #define ZERO   (OITO-SEG_G)
21 #define UM     (SEG_B+SEG_C)
22 #define DOIS   (OITO-SEG_C-SEG_F)
23 #define TRES   (OITO-SEG_E+SEG_F)
24 #define QUATRO (UM+SEG_F+SEG_G)
25 #define SEIS   (OITO-SEG_B)
26 #define CINCO  (SEIS-SEG_E)
27 #define SETE   (UM+SEG_A)
28 #define NOVE   (OITO-SEG_E)
29 #define TUDO   OITO+SEG_DP
30
31 int main(void)
32 {
33   volatile char i, j;
34   volatile char disps[ ] =
35     {DSP_1,DSP_2,DSP_3,DSP_4};
36   volatile char vals[ ] =
37     {NOVE,SEIS,UM,CINCO,DOIS,
38     QUATRO,TRES,ZERO,OITO,SETE};
39   WDTCTL = WDTPW | WDTHOLD;
40   BCSCTL1 = CALBC1_1MHZ;
41   DCOCTL = CALDCO_1MHZ;
42   P1REN |=  BTN;
43   P1OUT |=  BTN;
44   P1DIR &= ~BTN;
45   P1DIR |= DSPS;
46   P2DIR = TUDO;
47   for(j=0; j<4; j=(j+3)%4)
48     for(i=0; i<9; i++)
49       if((P1IN & BTN)==0)
50       {
51         P1OUT |=DSPS;
52         P2OUT  = vals[i];
53         P1OUT &=~disps[j];
54       }
55   return 0;
56 }
```

O código acima controla 4 displays de 7 segmentos multiplexados. Os LEDs dos displays são denominados de acordo com a Fig. 1. Para CADA item a seguir, indique se ele é verdadeiro ou falso.

A O código acima só acende LEDs em displays de 7 segmentos de catodo comum

B O botão conectado só tem funcionalidade se ligado ao pino P1.3 e ao terra.

C A função `main()` não possui um loop infinito.

D Para os caracteres entre 0 e 9, existe a mesma probabilidade de qualquer um deles aparecer nos displays.

E No instante em que alguém pressiona o botão ligado ao pino P1.3, o código escreve um caracter entre 0 a 9 em somente um dos quatros displays conectados.

8. (1.25 pontos) **Considere que d_1 corresponde aos dois primeiros dígitos da sua matrícula, e que d_2 corresponde aos dois últimos números da sua matrícula. Por exemplo, para o aluno com matrícula 12/3456789, então $d_1 = 12$ e $d_2 = 89$. Calcule os valores de TACCR0 e TACCR1 para o Timer A funcionar em modo de comparação via canal 1 com frequência $f = 10 * d_2 + 1000$ Hz e ciclo de trabalho $C = (d_1/6 - 1) * 50\%$. Considere que o Timer A funciona em modo UP a $1/8$ da frequência do SMCLK, e que o modo Reset/Set foi escolhido. Arredonde os resultados para o valor inteiro mais próximo.**

Table 1: Cartão de respostas.

Nome _____

Matrícula _____

Turma _____

Questão	**Resposta** (V/F ou valor numérico)
1	(a)____(b)____(c)____(d)____
2	_____
3	(a)____(b)____(c)____(d)____
4	(a)____(b)____(c)____(d)____
5	(a)____(b)____(c)____(d)____(e)____(f)____
6	(a)____(b)____(c)____(d)____
7	(a)____(b)____(c)____(d)____(e)____
8	_____

Registradores das portas P1 e P2 do MSP430

Reg.	Uso	Exemplo
PxDIR	Define se o pino correspondente ao bit será de entrada (bit = 0) ou de saída (bit = 1).	Se PxDIR = 0xF, os pinos Px.0-Px.3 são definidos como saída, e Px.4-Px.7 são definidos como entrada.
PxREN	Habilita o resistor de pull-up/down correspondente ao bit.	Se PxREN = 0xF, somente os resistores de pull-up/ldown nos pinos Px.0-Px.3 são habilitados.
PxOUT	Escreve o valor no pino de saída correspondente ao bit.	Se PxOUT = 0xF, os pinos Px.0-Px.3 são levados para nível alto, e Px.4-Px.7 são levados para nível baixo.
PxOUT	Define se o resistor correspondente ao bit será de pull-up (bit = 1) ou de pull-down (bit = 0).	Se PxREN = 0xFF e PxOUT = 0xF, os pinos Px.0-Px.3 terão resistores de pull-up, e Px.4-Px.7 terão resistores de pull-down.
PxIN	Reflete o valor do pino de entrada correspondente ao bit.	Se PxIN = 0xF, os pinos Px.0-Px.3 estão em nível alto, e Px.4-Px.7 estão em nível baixo.
PxIE	Habilita interrupções no pino correspondente ao bit.	Se PxIE = 0xF, os pinos Px.0-Px.3 podem causar interrupções mascaráveis.
PxIES	Define se a interrupção habilitada será por borda de subida (bit=0) ou de descida (bit=1).	Se PxIE = 0xFF e PxIES = 0xF, os pinos Px.0-Px.3 podem causar interrupções mascaráveis por borda de descida, e os pinos Px.4-Px.7 podem causar interrupções mascaráveis por borda de subida.

Instruções para configurar o MCLK @ 1MHz:

```
BCSCTL1 = CALBC1_1MHZ;
DCOCTL = CALDCO_1MHZ;
```

Instruções para configurar o MCLK @ 8MHz:

```
BCSCTL1 = CALBC1_8MHZ;
DCOCTL = CALDCO_8MHZ;
```

Instruções para configurar o MCLK @ 12MHz:

```
BCSCTL1 = CALBC1_12MHZ;
DCOCTL = CALDCO_12MHZ;
```

Instruções para configurar o MCLK @ 16MHz:

```
BCSCTL1 = CALBC1_16MHZ;
DCOCTL = CALDCO_16MHZ;
```

Instrução para parar o Watchdog Timer: `WDTCTL = WDTPW + WDTHOLD;`

Instrução para habilitar interrupções mascaráveis: `_BIS_SR(GIE);`

Interrupção mascarável da Porta P1: `interrupt(PORT1_VECTOR) P1_ISR(void){ }`

Interrupção mascarável da Porta P2: `interrupt(PORT2_VECTOR) P2_ISR(void){ }`

Interrupção mascarável do Timer A: `interrupt(TIMER0_A1_VECTOR) TA0_ISR(void){ }`

Headers úteis: `msp430g2553.h, legacymsp430.h`

12.3.1 TACTL, Timer_A Control Register

15	14	13	12	11	10	9	8
Unused						TASSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Unused Bits 15-10 Unused

TASSELx Bits 9-8 Timer_A clock source select
- 00 TACLK
- 01 ACLK
- 10 SMCLK
- 11 INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)

IDx Bits 7-6 Input divider. These bits select the divider for the input clock.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

MCx Bits 5-4 Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.
- 00 Stop mode: the timer is halted.
- 01 Up mode: the timer counts up to TACCR0.
- 10 Continuous mode: the timer counts up to 0FFFFh.
- 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h.

Unused Bit 3 Unused

TACLR Bit 2 Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.

TAIE Bit 1 Timer_A interrupt enable. This bit enables the TAIFG interrupt request.
- 0 Interrupt disabled
- 1 Interrupt enabled

TAIFG Bit 0 Timer_A interrupt flag
- 0 No interrupt pending
- 1 Interrupt pending

12.3.2 TAR, Timer_A Register

15	14	13	12	11	10	9	8
TARx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TARx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TARx Bits 15-0 Timer_A register. The TAR register is the count of Timer_A.

12.3.3 TACCRx, Timer_A Capture/Compare Register x

15	14	13	12	11	10	9	8
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TACCRx Bits 15-0 Timer_A capture/compare register.

Compare mode: TACCRx holds the data for the comparison to the timer value in the Timer_A Register, TAR.

Capture mode: The Timer_A Register, TAR, is copied into the TACCRx register when a capture is performed.

12.3.4 TACCTLx, Capture/Compare Control Register

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

CMx Bit 15-14 Capture mode
- 00 No capture
- 01 Capture on rising edge
- 10 Capture on falling edge
- 11 Capture on both rising and falling edges

CCISx Bit 13-12 Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections.
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS Bit 11 Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
- 0 Asynchronous capture
- 1 Synchronous capture

SCCI Bit 10 Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit

Unused Bit 9 Unused. Read only. Always read as 0.

CAP Bit 8 Capture mode
- 0 Compare mode
- 1 Capture mode

OUTMODx Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0, because EQUx = EQU0.
- 000 OUT bit value
- 001 Set
- 010 Toggle/reset
- 011 Set/reset
- 100 Toggle
- 101 Reset
- 110 Toggle/set
- 111 Reset/set

CCIE Bit 4 Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.
- 0 Interrupt disabled
- 1 Interrupt enabled

CCI Bit 3 Capture/compare input. The selected input signal can be read by this bit.

OUT Bit 2 Output. For output mode 0, this bit directly controls the state of the output.
- 0 Output low
- 1 Output high

COV Bit 1 Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.
- 0 No capture overflow occurred
- 1 Capture overflow occurred

CCIFG Bit 0 Capture/compare interrupt flag
- 0 No interrupt pending
- 1 Interrupt pending