



CEFET-MG — Centro Federal de Educação Tecnológica de Minas Gerais  
DEPARTAMENTO DE COMPUTAÇÃO DE DIVINÓPOLIS — DECOM-DV

## Microprocessadores e Microcontroladores

### Primeira Atividade Avaliativa

Aluno: HENRIQUE SOUZA FAGUNDES

Valor: 30 pontos (cada questão vale 5 pontos)

Turma: 2024/1

Prof. M. Sc. Diego Ascânio Santos

Respostas:

1      2      3      4      5      6

-----

### Questão 1

Considere o código abaixo:

```
const int ledPin = 13;
const int interruptPin = 2; // only pin 2 and 3 can be used for interrupts

volatile int state = LOW;

void blink() { // ISR function
    state = !state; // toggle the state
}

// missing setup function

void loop() {
    digitalWrite(ledPin, state);
}
```

É desejado que o LED conectado ao pino 13 comute de estado a cada vez que o botão conectado ao pino 2 for pressionado. O *pushbutton* conectado ao pino 2 do arduino também está conectado ao pino GND do microcontrolador.

Qual alternativa contém a implementação da função `setup()` que atende a esse requisito no Arduino UNO?

a)

```
void setup() {
    pinMode(2, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);
    attachInterrupt(
        digitalPinToInterrupt(interruptPin),
        blink,
        KEEPING
    );
}
```

b)

```
void setup() {
    pinMode(2, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);
    attachInterrupt(
        digitalPinToInterrupt(interruptPin),
        blink,
        CHANGE
    );
}
```

```
}
```

c)

```
void setup() {
    pinMode(2, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);
    attachInterrupt(
        digitalPinToInterrupt(interruptPin),
        blink,
        HIGH
    );
}
```

d)

```
void setup() {
    pinMode(2, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);
    attachInterrupt(
        digitalPinToInterrupt(interruptPin),
        blink,
        FALLING
    );
}
```

e)

```
void setup() {
    pinMode(2, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);
    attachInterrupt(
        digitalPinToInterrupt(interruptPin),
        blink,
        LOW
    );
}
```

---

## Questão 2

Cláudia perguntou ao professor Ascânio se sua rotina de interrupções de overflow do `Timer2` estava correta para contar intervalos de tempo de 10 em 10 segundos, pois, estava com dúvidas se seus cálculos de quantidade de *overflows* e o modo de *prescaling* que havia definido para o `Timer2` estavam corretos. O professor Ascânio verificou o código apresentado por Cláudia e disse que tanto os cálculos quanto o *prescaling* estavam corretos, mas, que a rotina de interrupção de *overflow* do `Timer2` — *Flag TOIE2* — estava desabilitada.

Qual deve ser a instrução que Cláudia deve adicionar à função `setup()` de seu código para habilitar a interrupção de *overflow* do `Timer2`?

- a) `TIMSK2 = TIMSK2 | 0b00000001;`
- b) `TIMSK2 = 0b00000000;`
- c) `TIMSK2 = TIMSK2 | 0b00000000;`
- d) `TIMSK2 = TIMSK2 | 0b00000010;`
- e) `TIMSK2 = TIMSK2 | 0b00000100;`

---

## Questão 3

A respeito de entradas e saídas digitais do Arduino, resistores pull-up e pull-down, contatos normalmente abertos e normalmente fechados, avalie as assertivas:

I. Por padrão as entradas digitais do Arduino estão preparadas para receber sinais digitais em nível lógico TTL (0V a 5V).

II. Não é necessário realizar quaisquer tipos de adaptações para conectar circuitos digitais não-TTL (por exemplo, CMOS) ao Arduino.

III. Um contato normalmente aberto é um contato que, em repouso, não permite a passagem de corrente elétrica.

IV. O nível lógico de uma entrada digital do Arduino conectada a um contato normalmente aberto em seu estado de repouso é sempre 0V.

V. Resistores pull-up são utilizados para garantir que uma entrada digital do Arduino esteja sempre em nível lógico alto em seu estado padrão.

VI. O Arduino não dispõe de resistores pull-up internos, sendo necessário adicionar resistores externos para este fim.

São verdadeiras as assertivas:

- a) I, II, III, IV, V e VI.
- b) I, III, IV, V e VI.
- c) I, III, IV e V.
- d) II e V.
- e) II e VI.

---

#### Questão 4

A respeito de entradas e saídas digitais do Arduino, resistores pull-up e pull-down, contatos normalmente abertos e normalmente fechados, avalie as assertivas:

I. Por padrão as entradas digitais do Arduino estão preparadas para receber sinais digitais em nível lógico TTL (0V a 5V).

II. Não é necessário realizar quaisquer tipos de adaptações para conectar circuitos digitais não-TTL (por exemplo, CMOS) ao Arduino.

III. Um contato normalmente aberto é um contato que, em repouso, não permite a passagem de corrente elétrica.

IV. O nível lógico de uma entrada digital do Arduino conectada a um contato normalmente aberto em seu estado de repouso é sempre 0V.

V. Resistores pull-up são utilizados para garantir que uma entrada digital do Arduino esteja sempre em nível lógico alto em seu estado padrão.

VI. O Arduino não dispõe de resistores pull-up internos, sendo necessário adicionar resistores externos para este fim.

São falsas as assertivas:

- a) I, II, III, IV, V e VI.
- b) II, III, IV, V e VI.
- c) II, IV, V e VI.
- d) II e V.
- e) II e VI.

---

#### Questão 5

Raul deseja fazer um LED comutar de estado a cada 5 segundos. Sem muito critério do entendimento viu nos códigos dos slides do prof. Ascânio o seguinte ISR que fazia o LED comutar de estado a cada 5 segundos:

##### Rotina de interrupção de Overflow associada ao `Timer2`

```
ISR(TIMER2_OVF_vect) {  
    overflows++;  
    if (overflows == 306) {  
        overflows = 0;  
        // Toggle the LED state  
        digitalWrite(13, !digitalRead(13));  
    }  
}
```

Entretanto, ao copiar os códigos, não se atentou ao modo correto de funcionamento e configurou —

sem querer — o prescaler do `Timer2` para o fator de 256, através do registrador de controle `TCCR2B` pela seguinte instrução na função `setup()`:

```
TCCR2B = 0b00000110;
```

Com isso, em vez do LED comutar de estado a cada 5 segundos, ele comutava a cada 1.25 segundos (aproximadamente). Qual dos seguintes modos de *prescaling* do `Timer2` faz com que o LED comute de estado a cada 5 segundos, como deseja Raul?

- a) `TCCR2B = 0b00000001;`
- b) `TCCR2B = 0b00000111;`
- c) `TCCR2B = 0b00000011;`
- d) `TCCR2B = 0b00000000;`
- e) `TCCR2B = 0b00000010;`

---

### Questão 6

(ENADE 2005 - 11) Apesar de todo o desenvolvimento, a construção de computadores e processadores continua, basicamente, seguindo a arquitetura clássica de von Neumann. As exceções a essa regra encontram-se em computadores de propósitos específicos e nos desenvolvidos em centros de pesquisa. Assinale a opção em que estão corretamente apresentadas características da operação básica de um processador clássico:

- a) Instruções e dados estão em uma memória física única; um programa é constituído de uma seqüência de instruções de máquina; uma instrução é lida da memória de acordo com a ordem dessa seqüência e, quando é executada, passa-se, então, para a próxima instrução na seqüência.
- b) Instruções e dados estão em memórias físicas distintas; um programa é constituído de um conjunto de instruções de máquina; uma instrução é lida da memória quando o seu operando-destino necessita ser recalculado; essa instrução é executada e o resultado é escrito no operando de destino, passando-se, então, para o próximo operando a ser recalculado.
- c) Instruções e dados estão em uma memória física única; um programa é constituído de um conjunto de instruções de máquina; uma instrução é lida da memória quando todos os seus operandos-fonte estiverem prontos e disponíveis; essa instrução é executada e o resultado é escrito no operando de destino, passando-se, então, para a instrução seguinte que tiver todos os operandos disponíveis.
- d) Instruções e dados estão em memórias físicas distintas; um programa é constituído de um conjunto de instruções de máquina; uma instrução é lida da memória quando todos os seus operandos-fonte estiverem prontos e disponíveis; essa instrução é executada e o resultado é escrito no operando de destino, passando-se, então, para a instrução seguinte que estiver com todos os seus operandos disponíveis.
- e) Instruções e dados estão em memórias físicas distintas; um programa é constituído de uma seqüência de instruções de máquina; uma instrução é lida da memória de acordo com a ordem dessa seqüência e, quando é executada, passa-se, então, para a próxima instrução na seqüência.