"Aula" exercícios/dúvidas/revisão

Programação C e 8051

Universidade Federal de Santa Catarina

23/10/2020

Ideia geral dessa reunião:

- Aula totalmente não oficial. Pensem como uma ajudinha a mais de alguém que já fez a matéria, e não de alguém que é expert no campo (como os professores).

Ideia geral dessa reunião:

- Aula totalmente não oficial. Pensem como uma ajudinha a mais de alguém que já fez a matéria, e não de alguém que é expert no campo (como os professores).
- Colaborativo! Eu pergunto coisas pra vocês, vocês perguntam coisas pra mim. Se eu não souber a resposta, buscamos juntos.

Ideia geral dessa reunião:

- Aula totalmente não oficial. Pensem como uma ajudinha a mais de alguém que já fez a matéria, e não de alguém que é expert no campo (como os professores).
- Colaborativo! Eu pergunto coisas pra vocês, vocês perguntam coisas pra mim. Se eu não souber a resposta, buscamos juntos.
- **Todo** o material, incluindo códigos, slides, a gravação da aula e qualquer outra coisa ficará disponível pra vocês a partir fim da aula.

Ideia geral dessa reunião:

- Aula totalmente não oficial. Pensem como uma ajudinha a mais de alguém que já fez a matéria, e não de alguém que é expert no campo (como os professores).
- Colaborativo! Eu pergunto coisas pra vocês, vocês perguntam coisas pra mim. Se eu não souber a resposta, buscamos juntos.
- **Todo** o material, incluindo códigos, slides, a gravação da aula e qualquer outra coisa ficará disponível pra vocês a partir fim da aula.
- Adendo: pra que ninguém fique desconfortável em fazer perguntas e interagir, se assume que a gravação da aula vai ficar apenas entre os colegas, okay?

Programação de hoje:

Programação de hoje:

- Princípios de verificação de código.

Programação de hoje:

- Princípios de verificação de código.
- Discussão sobre tratadores de interrupção e resolução de um problema pequeno.

Programação de hoje:

- Princípios de verificação de código.
- Discussão sobre tratadores de interrupção e resolução de um problema pequeno.
- Revisão sobre SFRs.

Programação de hoje:

- Princípios de verificação de código.
- Discussão sobre tratadores de interrupção e resolução de um problema pequeno.
- Revisão sobre `SFR`s.
- Tática de resolução de problemas e como resolução de um problema que vocês já fizeram.

Programação de hoje:

- Princípios de verificação de código.
- Discussão sobre tratadores de interrupção e resolução de um problema pequeno.
- Revisão sobre `SFR`s.
- Tática de resolução de problemas e como resolução de um problema que vocês já fizeram.
- Resolução de um problema de prova, começo ao fim.

- Composição de procedimentos: Uma coisa por vez.

- Composição de procedimentos: Uma coisa por vez.
- Escreve o código $\rightarrow$ Verifica o funcionamento $\rightarrow$ Usa esse código num sistema.

- Composição de procedimentos: Uma coisa por vez.
- Escreve o código → Verifica o funcionamento → Usa esse código num sistema.
- Exemplo de um dos exercícios (faremos depois):

6) Faça um programa que teste a chave conectada a P2.7. Se a mesma estiver fechada (P2.7='0'), rotacione um led aceso para a esquerda (se P2.6='1') ou para a direita (se P2.6='0') pelo número de vezes especificado pelo complemento do valor das 4 chaves menos significativas (P2.3 a P2.0). Se a chave P2.7 estiver aberta, fique aguardando ser alterada. Depois que a chave conectada a P2.7 for fechada, faça o solicitado e aguarde a mesma retornar para nível lógico baixo.

- Composição de procedimentos: Uma coisa por vez.
- Escreve o código → Verifica o funcionamento → Usa esse código num sistema.
- Exemplo de um dos exercícios (faremos depois):

6) Faça um programa que teste a chave conectada a P2.7. Se a mesma estiver fechada (P2.7='0'), rotacione um led aceso para a esquerda (se P2.6='1') ou para a direita (se P2.6='0') pelo número de vezes especificado pelo complemento do valor das 4 chaves menos significativas (P2.3 a P2.0). Se a chave P2.7 estiver aberta, fique aguardando ser alterada. Depois que a chave conectada a P2.7 for fechada, faça o solicitado e aguarde a mesma retornar para nível lógico baixo.

- Parte fundamental: rotacionar LED. Resto: Firula.

- *temperatura* deveria receber P2, mas recebe P1:

```
void main(void){
        if(verificar_temperatura()){
                acender_aquecedor(); //não funciona!
        }
}
void verificar_temperatura(){
        temperatura = P1; //mas devia ser P2...
}
```

- Se você não verificou o funcionamento de `verificar_temperatura()` (e tinha um erro) e escreveu `acender_aquecedor()`, se dobra o número de funções a procurar o possível erro! Pode ficar **MUITO** difícil de debuggar.

Façam seus tratadores de interrupção curtos e grossos.

```
void c51_int0 (void)  interrupt 0 {
        rotesq();
}

void rotesq (){
        while(1){
                if(leds == 0xFE) leds = 0x7F;
                else leds = (leds >> 1) | 0x80;

                P1 = leds;
                delay();
        }
}
```

Façam seus tratadores de interrupção curtos e grossos. Estudo de caso de um código que vocês escreveram (o código nunca vai pra frente):

```
void c51_int0 (void)  interrupt 0 {
        rotesq();
}

void rotesq (){
        while(1){
                if(leds == 0xFE) leds = 0x7F;
                else leds = (leds >> 1) | 0x80;

                P1 = leds;
                delay();
        }
}
```

- Qual o principal problema no nesse código e como possivelmente eu ia saber que esse era o problema?

- Qual o principal problema no nesse código e como possivelmente eu ia saber que esse era o problema?
- Chamadas de função fazem uso de `pop` e `push`, ou seja: mexem com o stack toda vez que se chama a função.

- Qual o principal problema no nesse código e como possivelmente eu ia saber que esse era o problema?
- Chamadas de função fazem uso de `pop` e `push`, ou seja: mexem com o stack toda vez que se chama a função.
- Se você nunca retornar da função e fazer contínuas chamadas $\rightarrow$ stack overflow (chamadas recursivas teriam o mesmo efeito).

- Qual o principal problema no nesse código e como possivelmente eu ia saber que esse era o problema?
- Chamadas de função fazem uso de `pop` e `push`, ou seja: mexem com o stack toda vez que se chama a função.
- Se você nunca retornar da função e fazer contínuas chamadas $\rightarrow$ stack overflow (chamadas recursivas teriam o mesmo efeito).
- Como eu **possivelmente** poderia saber disso?

Do manual do 8051:

As soon as any priority 1 interrupt is acknowledged, the IE (Interrupt Enable) register is re-defined so as to disable all but "priority 2" interrupts. Then, a CALL to LABEL executes the RETI instruction, which clears the priority 1 interrupt-in-progress flip-flop. At this point any priority 1 interrupt that is enabled can be serviced, but only "priority 2" interrupts are enabled.

- Alternativa de código (extremamente parecida com os exemplos da aula!):

```
bit state = 0;
void main(void) {
        while(state)
        {
                rotesq();
        }
}

void c51_int0 (void) interrupt 0 {
state = 1;
}
```

SFR - Special Function Register

```
sfr name = address;
//em reg51.h
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
```

`sbit` e bit-addressable SFRs

```
sbit name = sfr-name ^ bit-position; //caso 1
sbit name = sfr-address ^ bit-position; //caso 2
sbit name = sbit-address; //caso 3
```

Caso 1:

```
sbit name = sfr-name ^ bit-position;

/*Onde sfr-name deve ser divisível por 8
(Qual a razão?)*/
sfr P0   = 0x80;
sfr P1   = 0x90;

sbit P0_0 = P0^0;
sbit P0_1 = P0^1;
sbit batata = P1^4;
P0_0 = 1;
```

- Pois a memória é dividida em bytes! Ou 8 bits.

Caso 2:

```
sbit name = sfr-address ^ bit-position;

/*Onde sfr-address deve ser divisível por 8
(Qual a razão?)*/
sfr P0   = 0x80;
sfr P1   = 0x90;

sbit P0_0 = 0x80^0;
sbit P0_1 = 0x90^1;
sbit batata = 0x90^4;
```

- Pois a memória é dividida em bytes! Ou 8 bits.

- Nota: Nem todo SFR é bit-addressable. Só aqueles que terminam em 0 ou 8.
- Curiosidade: No entanto, o C51 tem um hack pra isso. Procurem a documentação.

- Nota: Nem todo SFR é bit-addressable. Só aqueles que terminam em 0 ou 8.
- Curiosidade: No entanto, o C51 tem um hack pra isso. Procurem a documentação.
- Quem vocês xingam sobre isso: Projetistas de silício da Intel.

Caso 3:

```
sbit name = sbit-address;

 /* Onde sbit-address deve estar entre 0x80-0xFF.
 (Qual a razão?) */
sfr P0   = 0x80;
sfr P1   = 0x90;

sbit P0_0 = 0x80;
sbit P0_1 = 0x81;
sbit batata = 0x93; //P1^4
```

- Exemplo no Keil, do arquivo `reg51.h`.

Exercícios!!!

- Uma prova antiga.
- Exercício 6 do lab 2. (rotacionar LEDs)

6) Faça um programa que teste a chave conectada a P2.7. Se a mesma estiver fechada (P2.7=’0’), rotacione um led aceso para a esquerda (se P2.6=’1’) ou para a direita (se P2.6=’0’) pelo número de vezes especificado pelo complemento do valor das 4 chaves menos significativas (P2.3 a P2.0). Se a chave P2.7 estiver aberta, fique aguardando ser alterada. Depois que a chave conectada a P2.7 for fechada, faça o solicitado e aguarde a mesma retornar para nível lógico baixo.

6) Faça um programa que teste a chave conectada a P2.7. Se a mesma estiver fechada (P2.7='0'), rotacione um led aceso para a esquerda (se P2.6='1') ou para a direita (se P2.6='0') pelo número de vezes especificado pelo complemento do valor das 4 chaves menos significativas (P2.3 a P2.0). Se a chave P2.7 estiver aberta, fique aguardando ser alterada. Depois que a chave conectada a P2.7 for fechada, faça o solicitado e aguarde a mesma retornar para nível lógico baixo.

TCON - Timer Control Register – Bit Addressable

Interrupt (n)	Descrição	Endereço ROM (Hex)
0	Externa 0 (P3.2)	0003
1	Timer/Counter 0	000B
2	Externa 1 (P3.3)	0013
3	Timer/Counter 1	001B
4	Serial	0023

IE - Interrupt Enable Register - Bit Addressable