Projeto 2 - Introdução aos recursos e funcionalidades de Microcontroladores 8 bits com arquitetura Harvard/ RISC

Erik Melges - 12547399
Gabriel Sotto Rodrigues - 11800952
Pedro A. B. Grando - 12547166

1 Código Fonte Comentado

Listing 1: Código Fonte Comentado

```
int count_array;
const int display_dictionary[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
0x7D, 0x07, 0x7F, 0x67};

void main() {
    count_array = 0;

    TRISB.RB0 = 1; // Configura RB0 como um bot o de entrada
    TRISB.RB1 = 1; // Configura RB1 como um bot o de entrada
    TRISD = 0x00; // Configura RD como sa da (display de 7 segmentos)
    PORTD = 0x00; // Configura a porta D como sa da

    INTCON2.RBPU=0;

    RCON.IPEN = 1; // Habilida n veis de interrup   o
    INTCON.GIEH = 1; // Chave geral de alta prioridade se IPEN = 1

    // Configura   o da interrup   o externa INTO
    INTCON.INT0IE = 1; // Habilita interrup   o externa INT0
    INTCON.INT0IF = 0; // Flag para adicionar interrup   o INT0
    INTCON2.INTEDG0 = 1; // Borda, 1    a borda de subida ao soltar a tecla

    // Configura   o do temporizador TWR0
    INTCON.TMR0IE = 1; // Habilita a interrup   o associada ao temporizador TMR0
    INTCON.TMR0IF = 0; // Zera a flag.
```

```
    T0CON = 0b00000111; // Registro de controle do temporizador

    // Configura  o do temporizador TMR1
    PIE1.TMR1IE = 1; // Habilita interrup  o do temporizador TMR1
    PIR1.TMR1IF = 0; // Zera a flag de interrup  o do temporizador TMR1
    IPR1.TMR1IP = 1; // Prioridade alta para TMR1
    T1CON = 0b00110000; // Configura o registrador T1CON, referente ao temporizado
                        // Prescaler 1:16, modo 16 bits

    // Overflow
    TMR1H = 0x0B; // Primeiro byte do TMR1
    TMR1L = 0xDC; //  ltimo  byte do TMR1

    TMR0H=0xC2;
    TMR0L=0xF7;

    T0CON.TMR0ON = 1; // Inicia o temporizador TMR0
    T1CON.TMR1ON = 1; // Inicia o temporizador TMR1

    while(1);
}

void INTERRUPTION_HIGH() iv 0x0008 ics ICS_AUTO {
if (INTCON.INT0IF) { // se a interrup  o amostrada foi a INT0
        // l gica para o primeiro bot o (RB0)
        T0CON.TMR0ON = 1; // Inicia o timer TMR0
        T1CON.TMR1ON = 0; // Para o temporizador TMR1
        //PORTD = 0x00;
        INTCON.INT0IF = 0; // Zera a flag de interrup  o externa INTO
    }

    if (INTCON3.INT1IF) {
        // l gica para o segundo bot o (RB1)
        T1CON.TMR1ON = 1; // Inicia o temporizador TMR1
        T0CON.TMR0ON = 0; // Para o temporizador TMR0
        //PORTD = 0x00;
        INTCON3.INT1IF = 0; // Zera a flag de interrup  o externa INT1
    }

    if (INTCON.TMR0IF) {
        // logica para quando ocorre a interrup  o do temporizador TMR0
     //   T0CON.TMR0ON = 0; // Para o timer TMR0
        TMR0H = 0xC2; // Primeiro byte do TMR0
        TMR0L = 0xF7; //  ltimo  byte do TMR0
        INTCON.TMR0IF = 0; // Zera a flag de interrup  o do temporizador TMR0
    }
```

```
    if (PIR1.TMR1IF) {
        // l gica para a interrup   o do temporizador TMR1
        //T1CON.TMR1ON = 0; // Para o temporizador TMR1
        TMR1H = 0x0B; // Primeiro byte do TMR1
        TMR1L = 0xDC; //   ltimo   byte do TMR1
        PIR1.TMR1IF = 0; // Zera a flag de interrup   o do temporizador TMR1
    }
        PORTD = display_dictionary[count_array];
        count_array = (count_array >= 9) ? 0 : count_array + 1;
}
```

2 Discussão Textual

Neste projeto, implementamos um sistema de controle utilizando um microcontrolador programável em linguagem C, centrando-nos em conceitos chave como timers, interrupções e acionamento de display de 7 segmentos. O objetivo era criar um sistema de contagem, onde dois botões (RB0 e RB1) acionam diferentes contadores. O resultado final foi um programa funcional que responde adequadamente aos botões, exibindo a contagem em um display de 7 segmentos.

A configuração dos timers foi crucial para a temporização e acionamento dos contadores. Utilizamos dois timers: TMR0 para controlar a exibição dos dígitos no display de 7 segmentos e TMR1 para a contagem propriamente dita. Os timers foram configurados com prescalers e valores iniciais específicos para atender aos requisitos do projeto.

Interrupções foram essenciais para lidar com eventos assíncronos, como o acionamento dos botões. Configuramos a interrupção externa INT0 para RB0 e INT1 para RB1. Quando um botão é pressionado, a interrupção associada é acionada, permitindo uma resposta imediata e evitando polling constante.

O acionamento do display de 7 segmentos envolveu a atualização dos registros da porta D com os valores correspondentes aos dígitos do dicionário display_dictionary. Isso foi feito dentro da rotina de interrupção, garantindo uma exibição contínua e suave dos números.

A estrutura do programa organiza-se em torno de uma função principal (main) e uma rotina de interrupção de alta prioridade (INTERRUPTION_HIGH). A função principal lida com a inicialização e configuração dos periféricos, enquanto a rotina de interrupção gerencia as ações associadas aos eventos de interrupção.

Em resumo, este projeto proporcionou uma compreensão aprofundada da configuração de timers, interrupções e manipulação de portas em microcontroladores. A implementação bem-sucedida do sistema de contagem valida a eficácia da abordagem utilizada na prática.

Projeto Atual (Linguagem C, PIC18F):

Vantagens:

1. **Facilidade de Leitura e Manutenção:** O código em linguagem C é geralmente mais legível e fácil de entender do que o Assembly, facilitando a manutenção e o entendimento do código.

2. **Portabilidade:** O código em C é mais portável entre diferentes arquiteturas de microcontroladores. Mudar para um microcontrolador diferente muitas vezes envolve apenas a recompilação do código, enquanto o Assembly pode exigir uma reescrita substancial.

3. **Desenvolvimento Rápido:** A programação em C é geralmente mais rápida e eficiente em termos de tempo de desenvolvimento, especialmente para projetos mais complexos.

4. **Abstração de Hardware:** O C fornece uma camada de abstração de hardware que facilita a adaptação do código a diferentes configurações de hardware sem alterações significativas no código fonte.

Desvantagens:

1. **Overhead de Memória e Velocidade:** O código em C pode ter algum overhead em termos de uso de memória e velocidade em comparação com Assembly, já que o compilador precisa gerar instruções intermediárias para a arquitetura específica do microcontrolador.

2. **Menos Controle Direto sobre o Hardware:** Em Assembly, você tem um controle mais direto sobre o hardware, o que pode ser crucial para otimizações específicas.

Projeto Anterior (Assembly, 8051):

Vantagens:

1. **Controle Direto sobre o Hardware:** Assembly oferece um controle preciso sobre o hardware, permitindo otimizações específicas para a arquitetura do microcontrolador.

2. **Eficiência de Código:** O código Assembly pode ser mais eficiente em termos de uso de memória e velocidade, pois as instruções são diretamente adaptadas à arquitetura do microcontrolador.

Desvantagens:

1. **Complexidade e Dificuldade de Manutenção:** O código Assembly é geralmente mais complexo e difícil de manter, especialmente para projetos maiores. Cada detalhe do hardware precisa ser gerenciado manualmente.

2. **Portabilidade Limitada:** O código Assembly é altamente dependente da arquitetura específica do microcontrolador, o que pode tornar a portabilidade entre diferentes plataformas mais desafiadora.

Conclusão:

O projeto atual em linguagem C usando PIC18F oferece vantagens significativas em termos de legibilidade, manutenção e portabilidade, enquanto o projeto anterior em Assembly usando 8051 pode ter uma eficiência de código superior e controle mais direto sobre o hardware. A escolha entre os dois depende dos requisitos específicos do projeto, considerando fatores como tempo de desenvolvimento, facilidade de manutenção e otimizações de desempenho.