Projetos em Assembly com MARS para Digital Lab Sim
Organização de Computadores I

Juliana Miranda Bosio, Vinícios Rosa Buzzi

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Caixa Postal 5094 – 88035-972 – Florianópolis – SC – Brazil

1. Projeto 1: Programa em Assembly MARS para Exibição Sequencial de Números no Display de Sete Segmentos

1.1. Objetivo

- **Sequenciamento de Números:** Desenvolver um programa que escreva os números de 0 a 9 sequencialmente em um display de sete segmentos.
- **Controle do Display:** Implementar a lógica necessária para controlar o display de sete segmentos da ferramenta Digital Lab Sim.
- **Repetição Automática:** Garantir que o programa repita a sequência de números continuamente.
- **Simulação em MARS:** Facilitar a execução do programa no simulador MARS, permitindo aos usuários observar o funcionamento correto da sequência numérica.
- **Didática e Prática:** Proporcionar uma experiência prática e didática para o aprendizado de manipulação de displays e programação em Assembly.

1.2. Implementação

1.2.1. Armazenamento dos Dados

O código define uma estrutura principal para armazenar os dados:

- `segment_map`: Um vetor que mapeia os números de 0 a 9 para os seus valores correspondentes que ativam os segmentos no display de sete segmentos.

1.2.2. Área do Programa

A seção de texto do código (`.text`) é responsável pela execução do programa e inclui as seguintes etapas:

1. Inicializações de Endereços:
   - `li $t0, 0`: Inicializa o contador para percorrer os números.
   - `li $t1, 10`: Define o limite do contador para 10 (números de 0 a 9).
   - `la $t2, segment_map`: Carrega o endereço base do mapeamento dos segmentos.
2. Loop Principal:
   - `lw $t3, 0($t2)`: Carrega o valor correspondente ao número atual no display.
   - `li $t5, 0xFFFF0010`: Define o endereço do display de sete segmentos.

- `sb $t3, 0($t5)`: Escreve o valor carregado no display de sete segmentos.
- `addi $t0, $t0, 1`: Incrementa o contador para avançar para o próximo número.
- `addi $t2, $t2, 4`: Avança o endereço para o próximo valor no mapeamento dos segmentos.
- `jal delay`: Chama a função de *delay* para manter o valor exibido por um tempo antes de avançar para o próximo número.
- `beq $t0, $t1, reset`: Se o contador atingir o limite de 10, o programa reseta o contador.

3. Resetar o Contador:
   - `li $t0, 0`: Reseta o contador para 0.
   - `la $t2, segment_map`: Reseta o endereço base do mapeamento dos segmentos.
   - `j loop`: Volta para o `loop` principal para continuar o processo.
4. Rotina de *Delay*:
   - `li $t4, 500000`: Define o número de iterações para criar um atraso (*delay*).
   - `addi $t4, $t4, -1`: Decrementa o contador de *delay*.
   - `bgtz $t4, delay_loop`: Continua o `loop` de *delay* até o contador ser zerado.
   - `jr $ra`: Retorna da função de *delay* para o `loop` principal.
5. Finalização do Programa:
   - `li $v0, 10`: Realiza a chamada de S.O. *syscall* para encerrar o programa.
   - `syscall`: Termina a execução.

1.3. Execução do Programa

A execução do programa é conforme esperado, exibindo os números de 0 a 9 em sequência no display de sete segmentos. O código realiza as operações de leitura dos números, tradução para valores binários para o display e usa a função de delay para garantir que os números sejam exibidos por um período de tempo visível antes de avançar para o próximo número. O ciclo de números reinicia ao atingir 9, voltando para 0, mantendo a exibição contínua.

Registrador	Nome	Valor (Hexadecimal)
$t0	Contador (0 a 9)	0x00000000
$t1	Limite do contador	0x0000000A
$t2	Endereço do mapeamento dos segmentos	0xFFFF0010
$t3	Valor do segmento atual	Variável (depende do contador)
$t4	Contador de delay	0x007A1200
$t5	Endereço do display de 7 segmentos	0xFFFF0010

Table 1. Valores dos registradores durante a execução do programa de contagem de 0 a 9.

O programa executa corretamente a contagem e exibição dos números de 0 a 9 no display de sete segmentos. Todos os registradores desempenham suas funções conforme a lógica implementada:

- `$t0` é inicializado com o valor 0 e incrementado a cada iteração até alcançar o valor limite `0x0000000A`.
- `$t2` aponta para o endereço base do mapeamento dos segmentos e é ajustado para o próximo valor em cada iteração.
- `$t3` armazena o valor correspondente ao segmento atual a ser exibido, carregado a partir do mapeamento dos segmentos.
- `$t5` armazena o endereço do display de sete segmentos e escreve o valor `$t3` no display.
- A função de delay é chamada para garantir que cada número permaneça visível por um curto período antes de ser substituído pelo próximo.
- Após alcançar o valor limite (`$t1`), o contador `$t0` é resetado para 0 e o endereço base `$t2` é ajustado para o início do mapeamento dos segmentos.

Desta maneira, a execução do código para o contador de 0 a 9 no display de sete segmentos demonstrou um funcionamento correto e eficiente, seguindo a lógica implementada pelo programa. Os valores observados nos registradores confirmam a exibição dos números de 0 a 9 no display.

2. Projeto 2: Programa de Leitura e Exibição de Teclas em Display de Sete Segmentos no MARS Assembly

2.1. Objetivo

- **Implementação de Leitura de Teclado:** Criar um programa que leia a entrada de um teclado alfanumérico.
- **Exibição de Valores:** Mostrar o valor da tecla pressionada em um display de sete segmentos.
- **Loop Contínuo:** Garantir que o programa leia continuamente as entradas do teclado e atualize o display em tempo real.
- **Execução Passo a Passo:** Facilitar a execução passo a passo no simulador MARS, permitindo aos usuários verificar cada etapa do processo.
- **Interatividade do Usuário:** Permitir que o usuário interaja constantemente com o teclado e veja os valores atualizados no display.

2.2. Implementação

2.2.1. Armazenamento dos Dados

O código define duas estruturas principais para armazenar os dados:

- `teclado`: Armazena os valores que o teclado alfanumérico pode retornar, variando de `0x11` a `0x88` para representar as teclas de `0` a `F`.
- `numero_7seg`: Define os valores de controle para exibir dígitos e letras (de `0` a `F`) no display de sete segmentos.

2.2.2. Área do Programa

A seção de texto do código (‘.text‘) é responsável pela execução do programa e inclui as seguintes etapas:

1. Inicializações de Endereços:
    - `li $t0, 0`: Inicializa o contador para percorrer os números.
    - `li $t1, 10`: Define o limite do contador para 10 (números de 0 a 9).
    - `la $t2, segment_map`: Carrega o endereço base do mapeamento dos segmentos.
2. Loop Principal:
    - `lw $t3, 0($t2)`: Carrega o valor correspondente ao número atual no display.
    - `li $t5, 0xFFFF0010`: Define o endereço do display de sete segmentos.
    - `sb $t3, 0($t5)`: Escreve o valor carregado no display de sete segmentos.
    - `addi $t0, $t0, 1`: Incrementa o contador para avançar para o próximo número.
    - `addi $t2, $t2, 4`: Avança o endereço para o próximo valor no mapeamento dos segmentos.
    - `jal delay`: Chama a função de *delay* para manter o valor exibido por um tempo antes de avançar para o próximo número.
    - `beq $t0, $t1, reset`: Se o contador atingir o limite de 10, o programa reseta o contador.
3. Resetar o Contador:
    - `li $t0, 0`: Reseta o contador para 0.
    - `la $t2, segment_map`: Reseta o endereço base do mapeamento dos segmentos.
    - `j loop`: Volta para o `loop` principal para continuar o processo.
4. Rotina de *Delay*:
    - `li $t4, 500000`: Define o número de iterações para criar um atraso (*delay*).
    - `addi $t4, $t4, -1`: Decrementa o contador de *delay*.
    - `bgtz $t4, delay_loop`: Continua o `loop` de *delay* até o contador ser zerado.
    - `jr $ra`: Retorna da função de *delay* para o `loop` principal.
5. Finalização do Programa:
    - `li $v0, 10`: Realiza a chamada de S.O. *syscall* para encerrar o programa.
    - `syscall`: Termina a execução.

2.3. Execução do Programa

A execução do programa foi de acordo com o esperado, sem encontrar problemas significativos. Durante o processo, foram observadas várias caracteristicas e dados nos registradores, demonstrando o funcionamento correto do código. A seguir, analisaremos tecnicamente como cada registrador é utilizado para exibir o número 7 no display de sete segmentos.

Registrador	Nome	Valor (Hexadecimal)
$s0	Endereço do display	0xffff0010
$s1	Endereço do selecionador de linhas	0xffff0012
$s2	Endereço do receptor de leitura do teclado	0xffff0014
$t1	Linha do teclado sendo lida	0x00000001
$t2	Valor lido do teclado	0x82 (tecla 7)
$t3	Índice da tecla no vetor `teclado`	0x0000001C
$t4	Valor da tecla a ser comparado	0x82 (tecla 7)
$t5	Valor para o display 7 segmentos	0x07 (7 no display)
$t6	Temporizador	0x00000000
$t7	Valor limite do temporizador	0x000186A0

Table 2. Valores dos registradores para exibir o número 7 no display.

Desta forma, de acordo com a Tabela 2, podemos resumir o processo de carregamento e escrita dos registrados da seguinte forma:

- `$t1` é carregado com a linha do teclado (valor inicial `0x00000001`).
- O valor lido do teclado é armazenado em `$t2`. Quando a tecla 7 é pressionada, `$t2` contém o valor `0x82`.
- O loop de comparação (`$t4`) verifica qual tecla foi pressionada. Quando `$t4` contém o valor `0x82`, ele corresponde à tecla 7.
- `$t3`, que é o índice da tecla pressionada, é ajustado para apontar para o valor correspondente no vetor `numero_7seg`. Nesse caso, `$t5` recebe o valor `0x07`, que representa o número 7 no display de sete segmentos.
- O valor `0x07` é enviado para o display (`$s0`), e o número 7 é exibido.
- O temporizador (`$t6`) é usado para garantir que o número permaneça no display por um curto período antes que o processo de verificação de teclas seja reiniciado.

Desta maneira, a execução do código para a leitura e exibição das teclas no display de sete segmentos demosntrou um funcionamento correto e eficiente, seguindo a lógica implementada pelo programa. Cada registrador desempenharam o seu papel, desde a leitura até a atualização do display e controle do temporizador. Os valores observados nos registradores confirmam a exibição clara e estável do número 7 no display.

3. Considerações Finais

A execução do primeiro programa, que exibe números de 0 a 9 no display de sete segmentos, apresentou a maior dificuldade na implementação do temporizador. Inicialmente, o display alternava os números numa frequência muito alta, tornando a visualização difícil. Para resolver isso, foi necessário aprender como inserir um temporizador adequado que garantisse que cada número fosse exibido por um período de tempo visível antes de avançar para o próximo.

O segundo programa teve como principais desafios o mapeamento correto para o display de sete segmentos e a inserção do loop contínuo de verificação de teclas pressionadas. Inicialmente, os valores exibidos no display não correspondiam às teclas pressionadas, devido a um mapeamento inadequado. Corrigimos isso mapeando corretamente os valores para que o display mostrasse o número ou caractere correto quando uma tecla

fosse pressionada. Além disso, implementar um loop contínuo de verificação de teclas pressionadas foi essencial para garantir que o programa respondesse em tempo real às entradas do teclado.

A criação e desenvolvimento destes programas ressaltaram a importância de uma abordagem iterativa e cuidadosa no desenvolvimento de software. Cada desafio encontrado, desde o temporizador até o mapeamento dos segmentos e a implementação do loop de verificação, serviu para aprimorar a eficiência dos programas. Os valores observados nos registradores durante a execução confirmam que ambos os programas funcionam conforme esperado. Por fim, a possibilidade de simular o comportamento do teclado e do display foi crucial para validar a lógica dos programas e assegurar sua funcionalidade.

4. Repositório da Atividade

Confira os códigos implementados acessando o $QRCode$ abaixo ou pelo link:

`https://github.com/buzziologia/UFSC/tree/main/OrganizacaoDeComputadores/Laboratorio02`

Figure 1. Repositório Github