Universidade Federal Rural de Pernambuco - UFRPE

Arquitetura e Organização de Computadores - 2022.2
Projeto 01 - Assembly MIPS e Simulador MARS
Prof. Vítor A Coutinho

INSTRUÇÕES

1. **Esta atividade compões nota de Projetos 1 (PE1) da 1a VA;**
2. A atividade deve ser feita em grupos de no máximo 4 pessoas;
3. **Data de entrega: 08/08/2023 (horário da aula)**
4. **Data das apresentações: (08 e 10/08/2023)**
5. Todas as questões devem ser implementadas utilizando Assembly MIPS e o simulador MARS;
6. A atividade é composta de uma **lista de exercícios (2 questões)** e **um projeto**. A lista de exercícios vale 30% da nota desta atividade e serve para praticar e criar funções que devem ajudar na implementação do projeto, que vale 70% da nota desta atividade.
7. Os arquivos entregues devem ser em formatos de código assembly *.asm, sendo 1 arquivo .asm ou .zip para cada questão de exercício e um arquivo .zip contendo um diretório para o projeto. Também é permitido os arquivos serem entregues através de um repositório do Github (OBS: este ŕepositório não pode ser atualizado após a data de entrega da atividade).
8. Os códigos devem estar **TOTALMENTE comentados**, explicando cada linha de código; cada arquivo deve apresentar um cabeçalho (utilizando comentários) contendo todas as informações (nome do grupo, atividade 1VA, disciplina, semestre letivo, questão referente ao arquivo, breve descrição do conteúdo do arquivo, etc.);
9. A entrega será feita via Google classroom. Sendo em grupo, basta um dos membros anexar a entrega, escrevendo no mural quem são os demais membros; os demais membros devem dar “Turn in/Retornar” na atividade (mesmo sem anexar nada) e escrever no mural de mensagem privada da atividade quem foi o membro que enviou. Também é permitido (e recomendável) desenvolver e entregar a atividade pelo Github (OBS: mesmo assim, o link do repositório precisa ser enviado no Classroom e cada membro precisa dar turn in para colaborar na organização da correção).
10. **Não serão toleradas cópias da internet ou de outros colegas;**

Não seguir as instruções à risca pode implicar em invalidação parcial ou total do projeto.

Lista de exercícios (30%)

1. (15%) O objetivo desta questão é implementar em assembly MIPS algumas funções da biblioteca “string.h” para linguagem C. As funções implementadas devem ser de grande utilidade para a implementação do projeto descrito ao fim deste documento. Devem ser implementadas as seguintes funções, com os nomes originais, e também um código “main” para testá-las adequadamente.

    a. Função strcpy (link):
    Função específica para cópia de strings. Copia uma string — incluindo o caractere NULL (‘\0’) — apontado pela **source** diretamente para o bloco de memória apontado pelo **destination.** Implemente esta função de modo que o parâmetro **destination** (endereço de memória do destino) deve ser passado em $a0 e o **source** (endereço de memória da origem) em $a1. Após a cópia, o parâmetro **destination** deve retornar em $v0.
    Referência adicional: https://man.archlinux.org/man/strcpy.3.en

    b. Função memcpy (link):
    Copia um total de bytes dado por **num** do local apontado pela **source** diretamente para o bloco de memória apontado pelo **destination.** Essa função não precisa procurar por um caractere NULL (‘\0’), ela apenas copia a quantidade de bytes especificada. Também não importa o tipo de dado que está sendo copiado. Implemente esta função de modo que o parâmetro **destination** deve ser passado em $a0, **source** em $a1 e **num** em $a2. Após a cópia, o parâmetro **destination** deve retornar em $v0.
    Referência adicional: https://man.archlinux.org/man/memcpy.3

    c. Função strcmp (link)
    Compara a string apontada por **str1** com a string apontada por **str2**. Esta função deve comparar o primeiro caractere de cada string. Se forem iguais entre si, a função deve continuar com os pares seguintes até que os caracteres difiram entre si ou até que um caractere NULL (‘\0’) seja alcançado. O retorno da função depende do valor ASCII (valor decimal do byte) dos caracteres comparados de acordo com o especificado a seguir:

Retorno	**Situação**
Inteiro negativo	O primeiro caractere diferente tem um

	valor decimal menor em **str1** do que em **str2**
Zero	O conteúdo de ambas as strings são iguais
Inteiro positivo	O primeiro caractere diferente tem um valor decimal maior em **str1** do que em **str2**

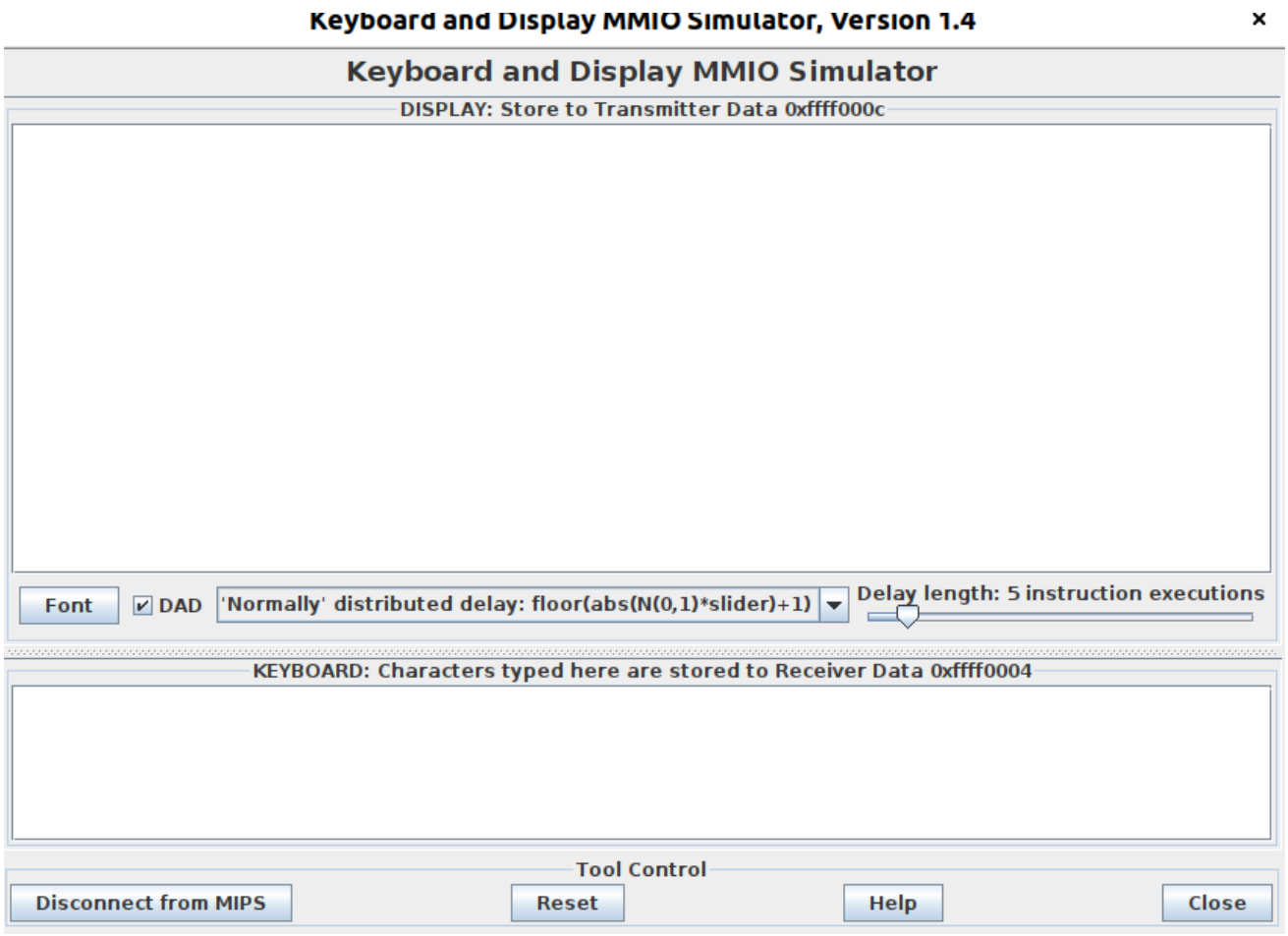
Implemente esta função de modo que o parâmetro **str1** deve ser passado em $a0 e o **srt2** em $a1. O retorno deve ser colocado em $v0.
Referência adicional: https://man.archlinux.org/man/strcmp.3

d. Função strncmp (link):
Similar à função *strcmp*, exceto por especificar um número máximo de caracteres que devem ser comparados ao invés da string completa. Compara até um número **num** de caracteres da string apontada por **str1** com a string apontada por **str2**. Esta função começa comparando o primeiro caractere de cada string. Se eles forem iguais entre si, ele continua com os pares seguintes até que os caracteres sejam diferentes, até que um caractere nulo de terminação seja alcançado ou até que um número **num** de caracteres sejam iguais em ambas as strings (o que acontecer primeiro). O retorno da função é análogo ao da função *strcmp.* Implemente esta função de modo que o parâmetro **str1** deve ser passado em $a0, **srt2** em $a1 e **num** em $a3. O retorno deve ser colocado em $v0.
Referência adicional: https://man.archlinux.org/man/strcmp.3

e. Função strcat (link):
Função de concatenação de strings. Acrescenta uma cópia da string apontada por **source** à string apontada por **destination**. O caractere NULL (‘\0’) de terminação em **destination** deve substituído pelo primeiro caractere de **source** e um caractere NULL é incluído no final da nova string formada pela concatenação de ambos em **destination,** isto é, a string concatenada só deve ter um único caracter NULL (‘\0’) ao final. Os endereços de **destination** e **source** não devem se sobrepor. O programador que utilizar essa função deve garantir que isso não ocorra. Se a cópia ocorrer entre objetos que se sobrepõem, o comportamento da função é indefinido. Implemente esta função de modo que o parâmetro **destination** deve ser passado em $a0 e **source** em $a1. Após a concatenação, o parâmetro **destination** deve retornar em $v0.
Referência adicional: https://man.archlinux.org/man/strcat.3p

2. (15%) Até o presente momento, utilizamos o terminal fornecido pelo próprio MARS e os serviços do **syscall** como forma de realizar operações de entrada e saída (ler dados, imprimir dados, etc). Nesta atividade, utilizaremos um mecanismo mais próximo do que acontece em um sistema físico real: **memory-mapped Input/Output (MMIO)**. Neste método de acessar dispositivos periféricos — como um teclado e um monitor ou display —, reservamos algum (ou alguns) endereço(s) do espaço de memória para ser exclusivo deste determinado periférico. Para acessar o periférico, fazemos leitura e escritas no(s) endereço(s) reservado(s) utilizando as instruções usuais de acesso a memória: **lw** e **sw**.

   Nesta atividade utilizaremos display e teclado MMIO, simulados pela ferramenta do MARS "*tools->Keyboard and Display MMIO Simulator*". Esta ferramenta se apresenta conforme ilustrado abaixo:



   Para que o simulador funcione, é necessário clicar em "Connect to MIPS". Além disso, é preciso garantir o correto funcionamento da leitura do teclado e da escrita no display através de instruções Assembly — diferentemente do terminal padrão do MARS, esta ferramenta não funciona "automaticamente". O correto funcionamento destes periféricos requer que o programador se comunique com o periférico através das

instruções **sw** e **lw** conforme descrito no apêndice A.8 do livro texto “Organização e Projeto de Computadores, John Hennessy”, disponível na plataforma “Minha Biblioteca” (UFRPE).

Perceba que há um campo para o DISPLAY e outro para o KEYBOARD. **Ambos funcionam independentemente**. O objetivo desta questão é realizar um código que lê caracteres do teclado e os imprime em seguida no display — esse processo contínuo é chamado de *Echo* e é similar ao que acontece quando você digita algum texto no terminal do seu PC.

Implemente um código fica constantemente tentando ler um caractere do KEYBOARD MMIO e, sempre que receber um, imprime o mesmo caracter imediatamente no DISPLAY MMIO. A leitura do Apêndice A8 é fundamental para a implementação da questão. Utilize a abordagem sem interrupção (“polling”) por simplicidade.

Projeto (70%)

O objetivo deste projeto é implementar um sistema para restaurantes de cadastro de cardápio e contas de mesas. Este sistema de cadastro deve ser operado através de um terminal (prompt/shell) que funciona como um interpretador de comandos de texto, ou seja, o sistema vai ficar constantemente checando por entradas de texto (string) e interpretando o que for recebido a partir de uma lista de comandos que o sistema deve ser capaz de executar. Os comandos que devem ser implementados serão descritos adiante na seção de requisitos de projeto.

ATENÇÃO: *é permitido utilizar o terminal padrão do MARS através das chamadas de syscall. Entretanto, o grupo que implementar o sistema utilizando o KEYBOARD MMIO e DISPLAY MMIO ganha (até)* ***1.0 ponto extra na nota deste projeto!!***

Sugestão: *evite trabalhar com acentos e cedilhas por simplicidade.*

Os requisitos do projeto devem ser seguidos rigorosamente e são elucidados a seguir.

Requisitos de Projeto:

R1. O sistema deve ser capaz de registrar itens no cardápio do restaurante. O sistema permite cardápios com até 20 itens. Cada item deve possuir um código (número de 1 a 20), um preço e uma descrição do item.

R2. O sistema deve ser capaz de gerenciar até 15 mesas simultaneamente. Cada mesa deve possuir um código (de 01 a 15), status (ocupada/desocupada), um nome de responsável, telefone de contato e um registro de pedidos.

R3. O sistema deve ser capaz de registrar pedidos associados a cada mesa. Cada mesa pode ter até 20 pedidos. Pedidos repetidos não devem introduzir uma nova entrada no registro de pedidos. Um contador para pedidos repetidos deve ser considerado.

R4. O sistema deve possibilitar o pagamento parcial das contas, abatendo o valor pago do saldo devedor atual.

R5. O sistema deve fornecer um relatório de consumo total, listando cada pedido e suas respectivas quantidades, o valor total da conta, o saldo total já pago e o saldo devedor atual por mesa;

R6. O sistema deve ser capaz de fechar a conta de uma mesa. Para isso, deve checar se o saldo total já foi quitado. Ao fechar a mesa, o sistema deve apagar todos os registros de consumo automaticamente e a mesa deve ficar com status “Desocupada”.

R7. O sistema deve salvar todos os dados em um arquivo de modo que, caso o sistema seja reiniciado, os dados salvos anteriormente devem ser resgatados. Para tal, devem ser utilizados os serviços de arquivos do **syscall**. Sempre que o programa for inicializado, os dados salvos devem ser resgatados automaticamente.

R8. O sistema deve ser operado através através de um terminal que fica constantemente lendo entradas de texto (strings) e interpretando o que for recebido para executar comandos. O terminal deve aguardar uma quebra de linha “\n” para tentar interpretar o comando do usuário.

R9. A cada nova linha do terminal, uma string padrão deve ser impressa como “banner” na parte anterior ao campo que o usuário deve escrever um comando (similar a um terminal linux ou windows). O formato desta string deve ser “<nome_restaurante>-shell>>“. Criem um nome para o restaurante. Caso o nome seja grande, considere usar abreviaturas no banner. Por exemplo, a string banner do Arvo Restaurante poderia ser “arvo-shell>>“. Segue um vídeo com este exemplo em anexo (**link**).

R10. Todos os comandos devem ser interpretados através de uma string digitada pelo usuário e finalizados com uma quebra de linha (‘\n’). Alguns comandos podem ter opções e todas as opções devem ser iniciadas com o caractere “-” (sinal de menos). Sempre que um comando não existente for executado, o terminal deve retornar uma mensagem “Comando invalido”. Todos os comandos da seguinte lista devem ser implementados:

cmd_1. `cardapio_ad-<option1>-<option2>-<option3>`

Este comando adiciona um item ao cardápio. A opção <option1> deve ser uma string numérica que informa o código do item, de 01 a 20. A <option 2> deve ser uma string numérica que informa o preço, em centavos, no formato “XXXXX”. A `<option3>` fornece uma descrição para o item. Após a realização do comando com sucesso, deve retornar uma mensagem “Item adicionado com sucesso”. Caso o item do cardápio já possua alguma entrada cadastrada, deve retornar

mensagem de “Falha: número de item já cadastrado”. Caso o número de item fornecido esteja fora do intervalo 01 a 20, retornar “Falha: código de item inválido”.

Exemplo de uso: para adicionar uma coca-cola como item de código 15 que custe R$ 4,90, deve ser usado

```
cardapio_ad-15-00490-coca cola
```

cmd_2. `cardapio_rm-<option1>`

Este comando remove um item do cardapio pela <option1> que representa uma string numérica que informa o código do item, de 01 a 20. Após a realização do comando deve retornar uma mensagem “Item removido com sucesso”. Caso o código do item informado não possua nem um cadastro, retornar uma mensagem “Código informado não possui item cadastrado no cardápio”. Caso o número de item fornecido esteja fora do intervalo 01 a 20, retornar “Falha: código de item inválido”.

Exemplo de uso: para remover a coca-cola como item de código 15

```
cardapio_rm-15
```

cmd_3. `cardapio_list`

Este comando deve listar todos os itens cadastrados, na ordem crescente de códigos cadastrados. A lista deve informar o código, valor e descrição do item.

cmd_4. `cardapio_format`

Este comando deve apagar todas as entradas do cardápio.

cmd_5. `mesa_iniciar-<option1>-<option2>-<option3>`

Este comando inicia o atendimento em uma mesa desocupada. A <option1> deve fornecer o código da mesa, através de uma string numérica XX (01 a 15), <option 2> deve fornecer o telefone de contato do responsável, no formato DDDNNNNNNNN e a <option3> deve ser uma string com o nome do responsável. Após a realização do comando, deve retornar uma mensagem “Atendimento iniciado com sucesso”. Caso a mesa escolhida já esteja em atendimento, retornar uma mensagem “Falha: mesa ocupada”. Caso o código da mesa não seja de 01 a 15, retornar “Falha: mesa inexistente”.

Exemplo de uso: iniciar o serviço na mesa 9 para o cliente Jose Silva

```
mesa_iniciar-09-08198765432-Jose Silva
```

cmd_6. `mesa_ad_item-<option1>-<option2>`

Este comando adiciona na conta da mesa especificada pela <option1> o item do cardápio especificado pela <option2>. Após o comando, retornar “Item adicionado com sucesso”. Caso a mesa especificada não exista, retornar “Falha: mesa inexistente”. Caso a mesa especificada esteja desocupada, retornar “Falha: mesa nao iniciou atendimento”. Caso o código do item não tenha sido cadastrado no cardápio, retornar “Falha: item não cadastrado no cardápio”. Caso o código do item seja invalido (fora de 01 a 20), retornar “Falha: codigo do item invalido”.

Exemplo de uso: adicionar o item 10 do cardapio na mesa 9

```
mesa_ad_item-09-10
```

cmd_7. `mesa_rm_item`

Este comando remove da conta da mesa especificada pela <option1> o item do cardápio especificado pela <option2>. Após o comando, retornar “Item removido com sucesso”. Caso a mesa especificada não exista, retornar “Falha: mesa inexistente”. Caso a mesa especificada esteja desocupada, retornar “Falha: mesa nao iniciou atendimento”. Caso o item não conste no pedido da mesa, retornar “Falha: item nao consta na conta”. Caso o código do item seja invalido (fora de 01 a 20), retornar “Falha: codigo do item invalido”.

Exemplo de uso: remover o item 10 do cardapio na mesa 9

```
mesa_rm_item-09-10
```

cmd_8. `mesa_format`

Este comando deve colocar todas as mesas com status “desocupado”, removendo todos os registros de cada mesa.

cmd_9. `mesa_parcial-<option1>`

`Este comando deve` fornecer um relatório de consumo atual, listando cada pedido e suas respectivas quantidades, o valor total da conta, o saldo total já pago e o saldo devedor atual por mesa;

cmd_10. `mesa_pagar-<option1>-<option2>`

Este comando realiza um pagamento parcial na conta da mesa especificada pela <option1> do valor especificado pela string da <option2>, no formato XXXXXX em centavos. Após o comando, retornar “Pagamento realizado com sucesso”. Caso a mesa especificada não exista, retornar “Falha: mesa inexistente”. Caso a mesa especificada esteja desocupada, retornar “Falha: mesa nao iniciou atendimento”.

cmd_11. `mesa_fechar-<option1>`

Este comando deve fechar a mesa. Apenas deve permitir que a mesa seja fechada caso o saldo devedor atual não seja maior do que zero. Após o

comando, deve retornar uma mensagem “Mesa fechada com sucesso” e a mesa deve ficar com status “desocupada” e apagar os registros da conta anterior. Caso a o código da mesa seja inválido (fora de 01 a 10), retornar “Falha: mesa inexistente”. Caso ainda haja saldo devedor, retornar “Falha: saldo devedor ainda não quitado. Valor restante: R$ XXXX,XX”, em que o valor restante deve ser apresentado no formato indicado.

cmd_12. `salvar`

Deve salvar todas as informações registradas em um arquivo externo. Cabe aos projetistas do grupo elaborar uma estrutura adequada para o formato do(s) arquivo(s).

cmd_13. `recarregar`

Recarrega as informações salvas no arquivo externo na execução atual do programa. Modificações não salvas serão perdidas e as informações salvas anteriormente recuperadas.

cmd_14. `formatar`

Apaga todas as informações da execução atual do programa, deixando todos as mesas e cardápios vazios. Este comando não deve salvar automaticamente no arquivo externo, sendo necessário usar posteriormente o comando “salvar” para registrar a formatação no arquivo externo.