Laboratório 1 - Organização e Arquitetura de Computadores - Agenda Telefônica Digital - Turma B - 23 de Abril de 2017

Thúlio Noslen Silva Santos
14/0164090
thulionoslen_@hotmail.com

Lucas de Moura Quadros
14/0150668
lucasmq92@gmail.com

1. Objetivos

Desenvolvimento de uma aplicação que implemente as ações básicas de uma agenda telefônica digital e análise crítica do desempenho da aplicação. A atividade também tem o intuito de desenvolver a compreensão das metodologias de organização e arquitetura de computadores.

2. Introdução

2.1. Arquitetura MIPS

Na disciplina Organização e Arquitetura de Computadores, a linguagem Assembly usada corresponde à usada pela Arquitetura MIPS, que é o nome de uma arquitetura de processadores baseada no uso de registradores[1][2]. Suas instruções apresentam um conjunto de 32 registradores para realizar diferentes operações. Alguns destes registradores não podem ser usados por programadores, pois são usados pela própria máquina para guardar ou manipular informações úteis, como os registradores $at, $k0 e $k1.

Processadores MIPS são do tipo RISC (Reduced Instruction Set Computer - ou seja, Computadores com Conjunto de Instruções Reduzidas), o que significa que o conjunto de instruções que o processador faz é considerado pequeno. Ainda que este número seja pequeno, é possível criar diversas operações a partir das intruções disponíveis.

2.2. Registradores

Sobre os registradores usados pelos programadores: os 4 registradores $ax (x variando de 0 a 3) são geralmente usados para argumentos de funções e são obrigatoriamente usados em diversas situações que envolvem o comando syscall, comando esse que também exige o uso dos 2 registradores $vx (x sendo 0 ou 1). Os $vx são também geralmente usados para retorno de valores de funções. Os 10 registradores $tx (x variando de 0 a 9) são caller-saved: usados para dados temporários que não necessitam ser preservados durante as chamadas de funções/sub-rotinas. Os 8 registradores $sx (x variando de 0 a 7) são calle-saved (necessitam ser preservados durante as chamadas). O registrador $sp (Stack Pointer) armazena o endereço na pilha de onde novos dados podem ser inseridos: acessando seu endereço e alterando-o, é possível guardar valores na memória. O registrador $ra armazena o endereço de retorno para o qual a última função chamada deve retornar.

2.3. Instruções

Na linguagem Assembly (MIPS), há três tipos de instruções: tipo R, tipo I e tipo J. Na instrução tipo R, há sempre dois registradores operandos e um outro que armazena o resultado, como no exemplo: add $rd, $rs, $rt , em que $rd é o registrador destino e $rs e $rt são os operandos.

Na instrução tipo I, há sempre um registrador destino, sendo que pode haver um registrador operando e um label, ou apenas um dos dois últimos, como nos exemplos: addi $rt, $rs, 5, lw $rt, label($rs) ou sw $rt, label. O label geralmente é uma variável em .data.

A instrução tipo J consiste em um desvio no código, em que o programa passa a executar instruções de um trecho de código. O local do código de onde o programa passa a executar as instruções depende do comando, como nos exemplos: jal label, jr $ra ou j label, em que o jal label faz com que o $ra armazene informação necessária para o programa terminar a execução do código em label e, no final da função label o comando jr $ra faz com que o programa passe a executar comandos depois da linha em que foi inserido o jal label. O j label simplesmente faz com que o programa passe a executar comandos contidos em label.Para implementar um programa, deve-se separar uma parte para variáveis estáticas em .data e uma parte para as instruções em .text.

No caso do simulador MARS, todas as informações mencionadas aqui e mais detalhes das instruções podem ser conferida no menu Help[3][4] do programa.

3. Materiais e Métodos

Os materais e equipamentos utilizados foram computadores comuns, em que o programa foi testado tanto em sistemas Linux quanto Windows. Para desenvolver a aplica-

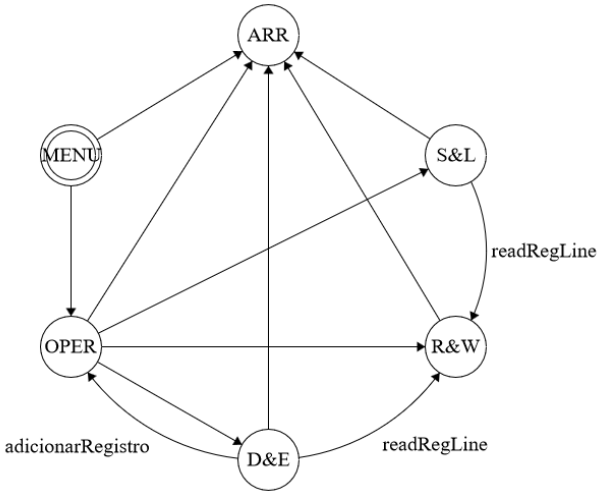

Figura 1. Diagrama mostrando a relação entre os arquivos do projeto.

ção, foi usado o simulador MARS versão 4.5, tanto para editar o código fonte quanto para executar o programa.

3.1. Organizaçao dos arquivos do projeto

O programa foi elaborado inicialmente num só arquivo *.asm*, e ao final acabou se tornando 6 arquivos diferentes, na tentativa de deixar o projeto mais organizado e tornar a localização de funções específicas mais fácil. O diagrama da Figura 1 tenta elucidar como os arquivos se conectam.

As setas, que saem de um arquivo de origem para um arquivo de destino, indicam que o arquivo de origem chama várias funções presentes no arquivo de destino. Note que todos os arquivo chamam funções em *arrayManip.asm*, que contém várias funções miscelâneas, como o cálculo do tamanho de uma string e a transformação de um número na memória em uma representação de n dígitos em ASCII, entre outras.

A interface do menu, presente em *menu.asm*, foi isolada do resto do código para manter o menu separado das operações de memória feitas nos arquivos. Desconsiderando-se as chamadas às funções em *arrayManip.asm* daqui em diante, *menu.asm* só chama funções presentes no arquivo *operations.asm*, que define o que cada opção do menu faz e implementa a função de coleta de dados do usuário. Como ele define todas as operações do menu, o arquivo chama funções de todos os três arquivos restantes, *searchAndList.asm*, *readAndWrite.asm* e *deleteAndEdit.asm*.

O arquivo *readAndWrite.asm* é responsável pelas funções de leitura e escrita nos arquivos texto usados pelo programa, além da criação e destruição destes. As funções deste arquivo são independente dos outros arquivos. O arquivo *deleteAndEdit.asm* implementa as funções de apagar e editar registros na agenda, mas as funções não são completamente independentes. A função *adicionarRegistro* de *operations.asm* é chamada quando um registro deve ser editado e *readRegLine* de *readAndWrite.asm* é chamada para ler um registro do arquivo *db.txt*. Por fim, o arquivo *searchAndList.asm* implementa as funções de listar todos os contatos e buscar e selecionar um contato específico. Este arquivo também tem uma chamada à função *readRegLine*, usada em ambas as funções mencionadas.

3.2. Serviços de syscall utilizados

No MARS, existe um menu de ajuda, acessado pela tecla F1, em que podem ser conferidos, por exemplo, as instruções básicas e pseudo-instruções disponíveis e os serviços de syscall que podem ser usados com o MARS.

Para a interface do programa, foram usados os serviços de *syscall* 50, 51, 54, 55 e 59. O serviço 50 permitiu a implementação de caixas de diálogo para confirmação e ações, como a confirmação de apagar um registro, de inserir um registro e sair do programa. Os serviços 51 e 54 permitiram a leitura de um número ou de uma string do usuário, respectivamente, e foram usados nas funções de busca, edição e inserção de um registro. O serviço 55 permitiu a implementação de caixas de diálogo com avisos variados, desde avisos informativos até avisos de erro, como no caso de um usuário digitar uma entrada inválida em alguma caixa de entrada. Por fim, o serviço 59 permitiu mostrar duas string como mensagem na tela, que facilitou a exibição dos contatos na tela.

Para a leitura de arquivos texto, foram usados os serviços 13, 14, 15 e 16. O serviço 13 permitiu abrir um arquivo para a leitura ou escrita. Quanto à escrita, o serviço permite que o arquivo seja inicializado vazio ou que o arquivo seja aberto com o ponteiro de escrita ao final do arquivo. Estas funções são similares à abrir um arquivo em C++ nos modos *read*, *write* e *append*, respectivamente. O serviço 14 permite a leitura do arquivo sabendo-se quantos caracteres devem ser lidos com antecedência, o serviço 15 permite a escrita no arquivo também sabendo-se quantos caracteres devem ser escrito com antecedência e o serviço 16 fecha um arquivo aberto.

3.3. Organizaçao interna do arquivo de database

O arquivo texto principal, *db.txt*, segue uma lógica em sua organização interna. Cada registro ocupa uma linha, e uma linha só pode conter um registro. A primeira informação de um registro é o tamanho do registro seguido dos tamanhos dos 4 campos do registro. São 3 dígitos para o tamanho total do registro e 3, 2, 3 e 2 dígitos para o tamanho do nome completo, do apelido, do e-mail e do telefone, respectivamente. Veja que todos os números estão concatenados sem divisão. A inclusão destas informações facilita a leitura das informações do registro, já que evita laços que leem caractere por caractere, substituindo-os por instruções

que leem tamanhos determinados matematicamente.

A próxima informação é o ID de um registro na agenda, que é um número de 3 dígitos. Veja que isto implica um limite máximo de 999 contatos na agenda. Por fim, depois do ID, seguem os campos de informação do contato, que em ordem são o nome completo, apelido (nome curto), endereço de e-mail e número do telefone, todos separados por ponto e vírgula. Note que sempre, ao final de um registro, deve existir um caractere \n.

3.4. Inicializaçao do programa

Quando o programa é iniciado, várias funções preparatórias são chamadas. O arquivo texto é criado se ele não existir, e quando ele existe os IDs são recalculados e os registros são re-enumerados na agenda. O programa faz isto copiando todo o arquivo para um arquivo auxiliar *auxdb.txt*, re-enumerando todos os registros, e depois escreve o arquivo auxiliar no arquivo principal. Isto é feito como proteção contra edições manuais nos registros no arquivo texto. Veja que os registros são re-enumerados conforme são lidos do arquivo, desconsiderando qualquer numeração prévia do IDs. Por fim, o número de contatos na agenda é obtido desta mesma função que copia o arquivo auxiliar no principal, e o resultado é guardado na memória para ser sempre exibido no menu principal.

3.5. Operaçao "adicionar novo registro"

Para implementar a operação 1 do menu, de adição de registro, o programa abre o arquivo *db.txt* para escrita ao final do arquivo, limpa todos os *buffers* de entrada e requisita os dados do usuário, fazendo verificação de dados a cada etapa. Se o usuário confirmar a operação de inserção, o programa calcula os tamanhos de cada entrada e manda esta informação e os endereços dos *buffers* para outras duas funções que conseguem escrever o campo de informações do registro e as entradas do usuário no arquivo.

Para a verificação de dados, foram tomados alguns limites máximos para cada campo: 200 caracteres para o número de contatos (201 se incluído o \0), 30 para o apelido e 200 para o e-mail. Para o telefone, o usuário deve inserir o um número puro de 10 dígitos, sendo os dois primeiros o DDD e os oito últimos o número do telefone em si, algo no formato XXYYYYYYYY. O programa verifica se somente números foram inseridos e se exatamente 10 foram inseridos, e caso os dados sejam válidos, passa o número para o formato (XX)YYYY-YYYY para ser armazenado na agenda.

3.6. Operaçao "mostrar lista de registros"

Para implementar a operação 2 do menu, de mostrar a lista de contatos, o programa percorre o arquivo texto, aberto no modo de leitura, de 5 em 5 contatos, ou até que se atinja o fim do arquivo. Como foram usadas caixas de diálogo e estas não suportam a rolagem de tela, optou-se por dividir os contatos em páginas, e mostrar 5 contatos por página. Veja que o programa não consegue voltar páginas, só avançar, mas esta discussão será retomada mais à frente, na seção 5.

Para deixar a exibição mais profissional, existem funções auxiliares que calculam o total de páginas e a página atual e escrevem os números em strings específicas, além de uma função que formata todos os campos para uma exibição mais natural.

3.7. Operaçao "buscar registro"

Para implementar a operação 3 do menu, de buscar e selecionar um registro na lista de contatos, primeiro o programa pede a letra inicial do nome completo, e faz uma breve verificação de dados. Com isto, procura-se em todo o arquivo texto registros que comecem com a letra inserida, e os registros encontrados são copiados para o arquivo auxiliar. Depois disto, todos os registros encontrados são mostrados ao usuário, permitindo que ele escolha um deles ou que avance a página. Aqui também não é possível voltar a página, somente avançar, e só é possível selecionar contatos que estão sendo mostrados na página atual, então não podem ser selecionados contatos de páginas passadas ou páginas seguintes. Os contatos devem ser escolhidos escolhendo um dos números na tela, ou pode-se avançar a página inserindo o número zero na entrada.

Por fim, um vez que um contato é selecionado, seu ID é recuperado no arquivo auxiliar, e é retornado pela função para que seja armazenado como o ID selecionado. No menu, depois que um contato for selecionado, seu nome completo será mostrado para que o usuário saiba que selecionou o contato. O nome do contato pode ser alterado caso o usuário edite o contato ou o campo de registro selecionado pode ser limpo caso o usuário apague o contato selecionado ou apague toda a agenda.

3.8. Operaçao "editar registro selecionado"

Para editar um registro selecionado, que é a operação 4 no menu, a função de busca deve ter sido previamente executada e um contato selecionado. Depois de confirmar que o usuário quer editar o contato, o programa copia todos os registros do arquivo principal num arquivo auxiliar, até que se chegue no ID do contato alvo. Aqui, o programa novamente pede todos os dados do contato para o usuário, assim como acontece na inserção de um novo contato, e insere o contato editado no arquivo auxiliar. Por fim, o programa termina de copiar os contatos restantes no arquivo auxiliar e em seguida escreve o arquivo auxiliar por cima do arquivo principal. Veja que esta operação poderia ser facilitada se o arquivo fosse carregado na memória, mas esta discussão será feita mais a frente na seção 5.

3.9. Operaçao "apagar registro selecionado"

Para apagar um registro selecionado da agenda, que é a operação 5, o programa segue de maneira similar à edição. Uma vez selecionado um contato e com a confirmação do usuário, o programa copia os registros no arquivo principal no arquivo auxiliar, mas pula o registro com o ID igual ao ID do contato selecionado. Em seguida, o programa termina de copiar os registros no arquivo auxiliar e novamente os passa do arquivo auxiliar ao principal. Ao final da operação algumas informações na memória são atualizadas, e o contanto selecionado volta a ser "nenhum"no menu.

3.10. Operaçao "apagar toda a lista de contatos"

A operação 6 do menu corresponde a apagar todos os registros da agenda. Isto se traduz no programa abrindo o arquivo texto no modo de escrita no início do arquivo, que resulta na criação forçada um arquivo vazio no lugar do arquivo antigo. Depois disto, a função atualiza algumas variáveis na memória para mostrar no menu que não há contatos na agenda.

3.11. Operaçao "sair do programa"

Por fim, a operação 7 corresponde a sair do programa. Aqui o programa simplesmente confirma se o usuário quer sair do programa e usa o serviço 10 do syscall, que corresponde a parar a execução.

3.12. Compilaçao do projeto no ambiente Linux

Para compilar e executar o projeto, todos os 6 arquivos deve estar junto ao executável do MARS para que seja possível usar o seguinte comando, executado na pasta em que se encontra o executável:

java -jar Mars4_5.jar p sm menu.asm

Com ele, o MARS compilará todos os 6 códigos-fonte e executará o programa a partir do *label* global *main*, que se encontra em *menu.asm*. É sugerido que os códigos-fonte sejam abertos num editor com codificação UTF-8 para que comentários sejam legíves. No MARS executado no Windows esta é a configuração padrão do editor nos computadores utilizados no projeto, mas abrindo-se o MARS no Linux o editor pode não oferecer suporte para UTF-8. As strings, no entanto, foram escritas em codificação ASCII.

3.13. Detalhes adicionais

Mais detalhes podem ser conferidos nos arquivos-fonte do projeto, em que há vários detalhamentos de mais baixo nível explicando o funcionamento de cada função. Além disto, as entradas, saídas e variáveis usadas em cada função também estão descritas nos comentários acima de cada função para facilitar o entendimento.

4. Resultados

Para obter resultados do programa, as funções do menu foram executadas sequencialmente e o contador de instruções não foi reiniciado, somente quando o programa foi encerrado, obtendo os resultados da tabela 1. Na tabela 2, estão os resultados para um segundo teste, com a agenda contendo 999 contatos. Em anexo ao projeto, junto aos arquivos *.asm*, foram incluídos os arquivos *dbTest.txt*, que contém a agenda usada para obter os resultados, e *dbFull.txt*, que contém uma agenda com 999 contatos, mas com os IDs foram de ordem. Para que qualquer um dos arquivos seja usado como o arquivo principal, o arquivo deve ser renomeado para *db.txt*.

5. Discussão e Conclusões

5.1. Comentários sobre os resultados

Conforme as tabelas 1 e tabelas 2, veja que a operação mais dispendiosa é a de mostrar a lista de contatos, seguida do conjunto de funções de preparação do programa. Isto acontece porque uma das operações mais computacionalmente complexas é a de copiar o arquivo principal no auxiliar e depois sobrescrever o arquivo principal com os dados do arquivo auxiliar. Se o arquivo fosse carregado na memória, isto possivelmente usaria menos instruções, já que as edições seriam diretas, sem copiar para outro arquivo.

Conforme a tabela 3, veja que somente a operação de preparo custa mais de 500 mil instruções. Como esta operação é similar às operações 4 e 5, poderíamos supor que uma execução sequencial das opções do menu facilmente ultrapassaria 1 milhões de instruções. Veja que isto poderia ser aliviado se fosse retirada a função de preparação que corrige todos os IDs. Aqui, do mesmo jeito, se o arquivo fosse carregado na memória, possivelmente teríamos maior desempenho.

5.2. Arquivo texto

Se o usuário editar manualmente o arquivo *db.txt*, o programa pode não funcionar corretamente. Se somente os IDs forem alterados, mas continuarem números de 3 dígitos, o programa, ao ser iniciado, vai re-enumerar os IDs conforma os registros aparecem no arquivo, ignorando qualquer ordem de IDs prévia. Se os IDs já estiverem em ordem, o programa ainda assim re-enumera os registros. Veja que isto foi implementada como uma medida de precaução, mas se mais desempenho fosse necessário, esta preparação pode ser retirada se houver certeza de que o arquivo *db.txt* só será modificado pela aplicação da agenda. Por outro lado, qualquer modificação ao arquivo *auxdb.txt* não fará diferença ao programa, porque ele sempre é sobrescrito.

Ainda sobre a edição manual do arquivo principal, o usuário pode alterar o texto que está em cada campo, mas somente se mantiver o mesmo número de caracteres, já que

Parâmetro	Preparo	Inserir	Mostrar lista	Buscar	Editar	Apagar	Apagar toda a lista	Sair
ALU	57%	56%	51%	51%	52%	53%	53%	53%
Jump	6%	6%	8%	8%	8%	8%	8%	8%
Branch	11%	13%	15%	15%	14%	13%	13%	13%
Memory	16%	16%	18%	18%	18%	17%	17%	17%
Other	10%	9%	7%	7%	8%	8%	8%	8%

Tabela 1. Estatísticas em porcentagem de instruções no primeiro teste, com o arquivo *dbTest.txt*.

Parâmetro	Preparo	Inserir	Mostrar lista	Buscar	Editar	Apagar	Apagar toda a lista	Sair
ALU	13091	15250	33700	40681	51849	62233	62755	62770
Jump	1413	1576	5248	6271	7520	8602	8626	8626
Branch	2733	3692	10361	11890	13788	15116	15358	15369
Memory	3725	4619	12339	14518	17352	19681	19919	19919
Other	2134	2263	4484	5701	7898	10050	10068	10070

Tabela 2. Estatísticas em número de instruções no primeiro teste, com o arquivo *dbTest.txt*.

o programa não checa se os valores estão todos corretos. O programa assume que ele próprio já organizou o arquivo corretamente. Se o usuário souber o que está fazendo, é possível alterar também os campos de informação do registro para editar manualmente um contato. Ainda assim, pode ser que um caractere inválido seja inserido no arquivo texto, e o programa pode não se comportar conforme o esperado.

O programa não permite ao usuário que entre com campos vazios ao adicionar ou editar um registro, mas note que se algum campo for alterado no arquivo texto e todo o campo de informações for ajustado corretamente, o programa não conseguirá verificar que existe um campo vazio.

5.3. Uso de memória

Inicialmente, o programa tinha um limite de contatos de 9999. A ideia inicial do projeto era ler todo o arquivo texto principal para a memória, realizar as operações necessária e ao sair do programa escrever os registros da memória no arquivo principal. No pior dos casos, para que fosse possível armazenar 9999 registros de tamanho máximo na memória, seria preciso um espaço de $(13 + 3 + 200 + 30 + 200 + 13 + 6 * 2) * 9999 = 471 * 9999 = 4709529$ bytes, ou $4709529/2^{20} = 4.49$ megabytes, contabilizando o tamanho do campo de informações, o tamanho do ID, o tamanho máximo de todos os campos e o tamanho de todos os separadores juntos. Quando um *array* com este tamanho foi usado no MARS, houveram problemas de compilação constantemente.

Com isto, o número máximo de contatos foi reduzido para 999, resultado num tamanho necessário de $471*999 = 471529$ bytes, ou $471529/2^{20} = 0.45$ megabytes. Em um dos computadores usados para desenvolver o projeto não houve problemas de compilação, mas em alguns computadores o código não pôde ser compilado. Neste caso, optou-se por manter o limite de contatos de 999 mas operar com os arquivos sem carregá-los na memória. Neste caso, contabilizando a memória restante alocada sem carregar o arquivo na memória, estimou-se que o programa usa em torno de 8700 bytes, ou $8700/2^{10} = 8.46$ kilobytes. Neste caso, esta redução no uso de memória acaba custando uso maior de memória do disco rígido.

5.4. Interface do menu

Como foi usada a caixa de diálogo do MARS ao invés de usar o terminal de entrada e saída, surgiram algumas vantagens e desvantagens. Uma das grandes vantagens é que a verificações dos dados se torna mais fácil, já que as caixas de diálogo configuram algumas flags quando existem problemas, como apertar em cancelar, apertar OK mas não inserir nada e passar do limite máximo de caracteres.

Como desvantagem, as caixas de diálogo não suportam rolagem de texto, então quando uma lista de contatos é mostrada, são mostrados 5 contatos por vez em várias páginas, tanto na operação 2 quanto na operação 3. Se fossem mostrados todos os contatos, a altura da janela ultrapassaria a tela. Optou-se por 5 contatos para que o programa fosse compatível com uma grande variedade de telas *widescreen*. Com isto, somente foi implementado avanço de páginas. No caso da operação 2, basta apertar em OK para avançar a página. Para a operação 3, a entrada 0 na caixa de diálogo avança a página. Como o arquivo não pode ter seu ponteiro de leitura alterado manualmente, implementar uma função que voltasse uma página acabou se tornando muito complicado, e esta função não foi incluída no trabalho. Talvez isto seja um inconveniente para o usuário caso ele queira selecionar um contato de uma página passada na operação de busca.

Parâmetro	Preparo		Apagar toda a lista		Sair	
ALU	348086 instr.	61%	348608 instr.	61%	348623 instr.	61%
Jump	37340 instr.	7%	37364 instr.	7%	37364 instr.	7%
Branch	39631 instr.	7%	39873 instr.	7%	39884 instr.	7%
Memory	73637 instr.	12%	73875 instr.	12%	73875 instr.	12%
Other	73988 instr.	13%	74006 instr.	13%	74008 instr.	13%

Tabela 3. Estatísticas das instruções no segundo teste, em número de instruções e porcentagem de instruções, com o arquivo *dbFull.txt*.

Referências

[1] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers, 5th edition, 2014.

[2] F. Vidal. Curso: Organização e arquitetura de computadores – turma b – 1/2017. `https://aprender.ead.unb.br/course/view.php?id=2431`, 2017. [Online; accessed 23-April-2017].

[3] K. Vollmar and P. Sanderson. Mars 4.5 help, 2017. Acessed by pressing F1 while running MARS 4.5.

[4] K. Vollmar and P. Sanderson. Mars 4.5 help contents. `http://courses.missouristate.edu/kenvollmar/mars/Help/MarsHelpIntro.html`, 2017. [Online; accessed 23-April-2017].