

DCC/ICEX/UFMG

Nome: Gustavo Guedes de Azevedo Barbosa

RA: 2016097471

Curso: Bacharelado em Engenharia Elétrica

Disciplina: Software Básico

2º Semestre de 2017

Trabalho Prático 1

Introdução: O primeiro trabalho prático da matéria de Software Básico apresentou tanto uma contradição quanto uma confirmação. O trabalho necessário para conseguir quase completar a atividade prática foi realmente intenso e extenso, confirmando o nome “Trabalho Prático”. Já a contradição encontra-se no nome da disciplina, que não tem absolutamente nada de básico. Observações pessoais a parte, desenvolver uma máquina virtual simples trouxe muito conhecimento tanto sobre programação em linguagem C, quanto sobre o funcionamento de uma máquina real, como aquelas que tenho estudado nas aulas da matéria.

A minha compreensão sobre modos de endereçamento, utilização de registradores, linguagem de máquina e de montagem, técnicas de montagem e ligação, após o termino da atividade, está muito melhor do que era quando eu estudava pelo livro, lendo apenas sobre a teoria por trás dessas implementações. Apesar de não ter conseguido terminar o trabalho, fiquei extremamente satisfeito com a rica carga de conhecimento que extraí durante o desenvolvimento da máquina virtual do Marcos.

Implementação e decisões de projeto: As estruturas de dados utilizadas no projeto variaram, mas com uma gama bastante restrita, uma vez que em nenhuma delas foi utilizado o método de encadeamento. Elas foram:

TADs:

- A struct “instruction” contém uma instrução da máquina. Porém, para tornar a estrutura de uso mais geral, apesar da palavra da máquina ser apenas 2 bytes, inclui também um campo para conter as “labels” do código em linguagem S2. O campo “op” contém a representação em S2 para o “opcode” da linguagem de máquina. O campo “var” guarda a variável à qual as instruções e as pseudo-instruções referenciam. O campo “size” é utilizado para guardar de maneira mais prática o tamanho de um arranjo dessa estrutura (técnica bastante usada ao longo da implementação). Já o campo “value” foi uma decisão ruim tomada no início do projeto para guardar valores quando fosse detectado as pseudo-instruções “DC” e “DS”. Acredito que utilizar o comando “atoi” no campo “var” teria sido mais inteligente.

-A struct “table” representa a tabela de símbolos e endereços que tanto o montador utiliza para saber os endereços das labels, direcionar branches, fazer chamadas de procedimentos ou sub-rotinas, acessar endereços de variáveis para a leitura ou escrita, quanto o ligador utiliza para conectar módulos e fazer o cálculo dos novos endereços relativos e, por fim, os absolutos.

Variáveis globais:

Na biblioteca da máquina são declaradas as seguintes variáveis globais:

-PC: Representa o registrador conhecido como contador de programa ou “program counter”, é utilizado tanto para apontar para a próxima instrução que será executada, quanto para endereçar “branches”, chamadas de funções e acesso a “variáveis” (endereços de memória rotulados).

-AC: Registrador aritmético de uso geral. Com bastante criatividade, pode ser usado para guardar endereços de retorno caso queiramos chamar uma função dentro de uma função chamada.

-RC: Registrador contador, utilizado para repetições, também pode ser utilizado, com cautela, para enviar endereços de retorno caso necessário.

-RX: Registrador de índice utilizado para guardar endereços de retorno caso o programador utilize ingenuamente as instruções “CAL” (chamar procedimento) e “RET” (retornar do procedimento). Esse registrador é a melhor definição de ponteiro e é responsável pelos acessos a endereços absolutos com a pseudo-instrução “DA” (define endereços absolutos), para passar parâmetros por referência para a função chamada por “CAL”.

-MEM: Memória primária do programa, pode ter seu valor alterado mudando a macro “MAX_MEM” que por padrão vem definida como 256.

O Simulador:

O simulador é responsável por executar de fato as instruções, ele recebe apenas arquivos em linguagem de máquina (M2), previamente ligados, e os carrega na posição de memória que é passada para o programa no “stdin”. Esse endereço deve ser o mesmo do que o passado para o ligador (veremos mais à frente duas justificativas para essa limitação). A função “overall_instruction” recebe um ponteiro para o opcode e outro para o dado na memória e opera sobre eles. O procedimento “dump_MEM” tem caráter meramente informativo, para o programador saber o estado da memória principal ao final da execução de seu programa. O procedimento “load_program” faz o papel do carregador, carregando para a memória principal, no endereço desejado, o código M2 resultante da ligação.

O Ligador:

O ligador é responsável por ligar códigos objeto, utilizando diretivas do ligador deixadas pelo montador para que aquele faça os cálculos corretos de chamada de função e definição de endereços absolutos. Nesse programa temos o procedimento “fill_table”, que faz a primeira leitura pelo arquivo gerado pelo montador, e cria a tabela de símbolos e endereços. O procedimento “search_address” recebe uma label e uma tabela referente à um módulo (além de contadores) e retorna o novo endereço de uma label após a ligação dos módulos. Caso uma função tenha sido chamada sem ser declarada, o programa imprime uma mensagem de erro e encerra. O procedimento “write_final_file” é dependente de um outro procedimento chamado “write_intermed_file”, os quais são responsáveis, inversamente, por escrever um arquivo temporário, já ligado, porém diferenciando os módulos, com a finalidade de calcular os endereços de chamada de função e definição de endereço absoluto, salvando-os numa tabela. Essa tabela é usada no primeiro procedimento para substituir as diretivas do ligador pelos respectivos endereços recalculados.

O procedimento “adjust_table” é responsável por ajustar aqueles ditos endereços (chamada de função e definição absoluta) ao endereço de ligação, passado para o programa também por “stdin”. *Devo ressaltar aqui um grande erro cometido por mim na implementação do Trabalho Prático: no começo do trabalho, meu conhecimento sobre o funcionamento de computadores era extremamente básico, uma vez que eu não tinha tido contato com Sistemas Lógicos ou Organização de Computadores, portanto implementei um montador que traduzia a linguagem S2 para M2 em um arquivo puramente sequencial, ou seja, cada endereço de memória estava em uma linha do arquivo, não diferenciando opcodes de dados.* Esse erro se refletiu na falha da implementação do programa auto-realocável, uma vez que o carregador não podia diferenciar um opcode de um dado. Um erro catastrófico que impede o programa de rodar um código que foi ligado em um endereço físico, mas carregado para outro. A função “print_table” tem caráter informativo, caso o programador queira visualizar a tabela criada para cada módulo ligado.

O Montador:

Como mencionado anteriormente, no período em que construí essa parte da atividade, o pouco conhecimento sobre funcionamento de computadores levou a um notável erro em sua implementação. No entanto, para endereços de carga e ligação iguais, essa versão do montador funciona perfeitamente e conta com o procedimento “read_file” que preenche um arranjo, alocado dinamicamente de acordo com o comprimento do código S2 (adquirido pela função “read_LS2code”), de “instructions”. Os procedimentos irmãos “verify_call_existance” e “verify_label_existance” que são responsáveis por verificar se uma chamada de função está chamando outra naquele módulo ou não e verificar quais são os rótulos (labels) daquele arquivo S2 que ele está traduzindo. A função mais importante dessa parte da atividade, “map_labels” é responsável por retornar a um rótulo, seu respectivo endereço relativo. Quando o

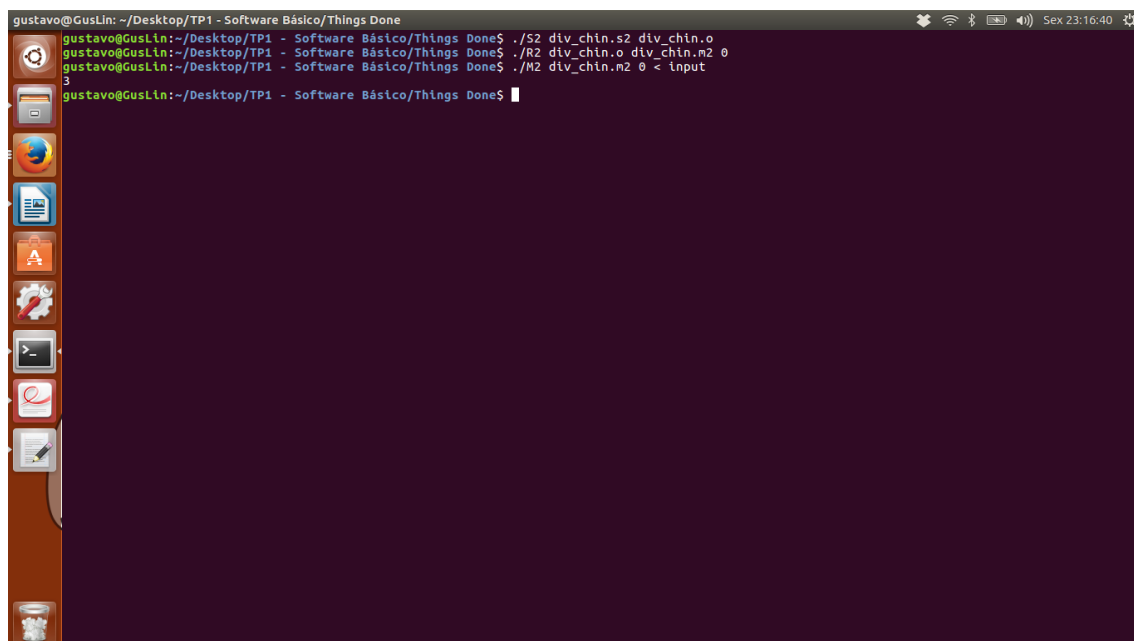
montador realiza a montagem, ele supõe o endereço 0 para a primeira instrução e constrói a partir daí sua tabela de símbolos e então a imprime no início do arquivo M2, para que o ligador possa usá-la. Por fim, a função “create_M2” é encarregada de fazer a tradução das instruções e pseudo-instruções para a linguagem M2.

Considerações do projetista:

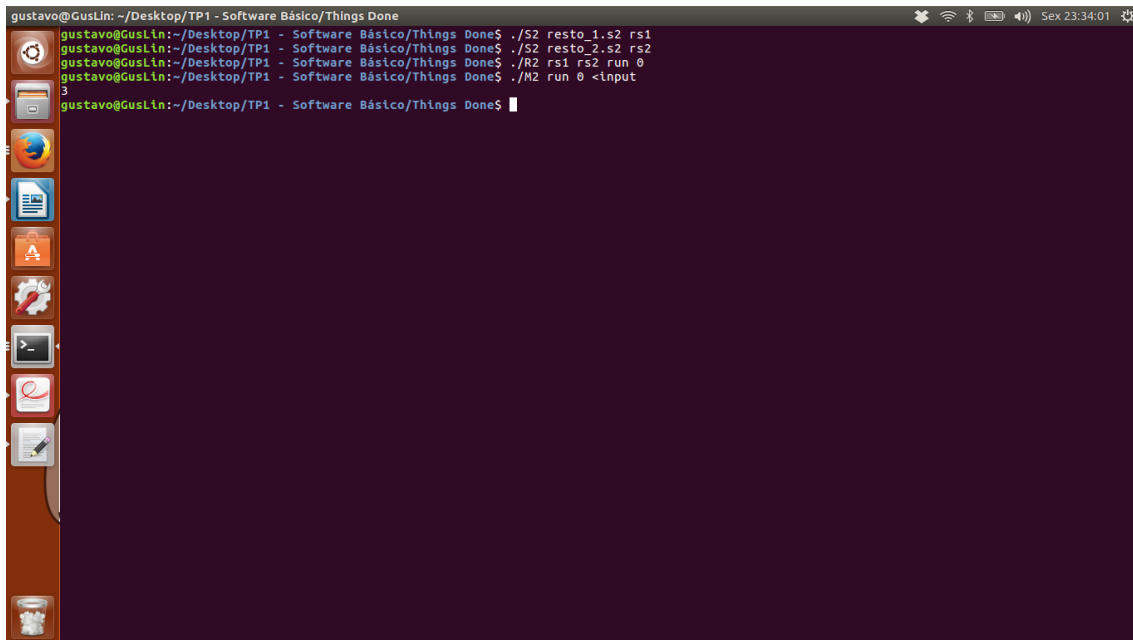
Como o programa trabalha com uma lógica bastante simples, contudo, nada fácil, e com dados bem pequenos, a maioria dos arranjos utilizados nas implementações das funções são de alocação estática, do mesmo tamanho da memória principal, para não haver risco de faltar espaço para que os procedimentos trabalhem. A simplicidade da máquina também a limita a chamadas de funções únicas, ou seja, não é possível chamar uma função dentro de outra de maneira segura.

Acompanha o projeto uma biblioteca em linguagem m4, que possibilita uma maior abstração do código em linguagem de montagem, portanto essa biblioteca ainda é bastante limitada, não contendo, por exemplo, macros para chamadas de função. Caso o programador deseje fazer um programa simples, mas extenso, escrevê-lo em linguagem m4 pode ser muito conveniente.

Testes: Mostro a seguir dois códigos completamente diferentes com a mesma finalidade, calcular o resto de uma divisão de inteiros, sendo que o primeiro utiliza só um módulo e o algoritmo da divisão chinesa e o segundo utiliza dois módulos com chamada de função. (Ambos os arquivos estão presentes na pasta do programa). Ambos os testes têm as mesmas entradas e apresentam o mesmo resultado.



```
gustavo@GusLin: ~/Desktop/TP1 - Software Básico/Things Done
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$ ./S2 div_chin.s2 div_chin.o
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$ ./R2 div_chin.o div_chin.n2 0
3
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$ ./M2 div_chin.n2 0 < input
3
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$
```



```
gustavo@GusLin: ~/Desktop/TP1 - Software Básico/Things Done
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$ ./S2 resto_1.s2 rs1
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$ ./S2 resto_2.s2 rs2
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$ ./R2 rs1 rs2 run 0
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$ ./M2 run 0 <input
3
gustavo@GusLin:~/Desktop/TP1 - Software Básico/Things Done$
```

Conclusão:

O trabalho prático trouxe muito conhecimento sobre o funcionamento de máquinas reais a partir do desenvolvimento de uma máquina virtual simples. Apesar de ter enfrentado muita dificuldade, sinto que aprendi muito no período que trabalhei na atividade. Devo dizer que dediquei uma quantidade considerável do meu tempo ao trabalho e mesmo assim não consegui terminá-lo a tempo. Espero que para os próximos alunos o trabalho seja mais bem distribuído e documentado do que o atual, pois ele contribui muito para a melhor compreensão dos assuntos tratados na disciplina.