DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UNIVERSIDADE FEDERAL DE MINAS GERAIS, BRAZIL

(UFMG)

Máquina Simple86 e montador

Um emulador e montador para a arquitetura exemplo Simple86

Belo Horizonte, 13 de Maio de 2016

Autores:

Breno Rodrigues Marques da Silva

Eugênio Pacceli Reis da Fonseca

1 - Introdução

Para o trabalho prático de Software Básico, matéria lecionada por Daniel Macedo pelo Departamento de Ciência da Computação, da Universidade Federal de Minas Gerais, foi proposta a implementação de um emulador e montador para a arquitetura exemplo chamada Simple86.

A arquitetura Simple86 contém um conjunto de instruções de tamanho variado, e, as palavras vindas da memória, assim como os registradores, contém 16 bits. As especificações completadas da arquitetura estão descritas no enunciado que esse trabalho responde, “TP1 – Software Básico.pdf”.

A implementação do emulador e do montador para a Simple86, foram feitas na linguagem de programação C++, para aproveitamento dos conceitos do paradigma orientado por objetos.

Essa documentação se limita ao montador, também chamado aqui de compilador, e o emulador está documentado na primeira documentação.

2 – Escolha da linguagem e do paradigma

A linguagem C++, por implementar o paradigma orientado por objetos, permite uma modularização mais completa do código, sem perder os poderes que herdou da linguagem C.

Fazer o emulador dividido por módulos (descritos pelas classes), significou reduzir o problema em vários pedaços menores. O controle de acesso por meio das definições de público e privado de cada classe, permite à cada módulo garantir seu bom funcionamento, sem perder a capacidade de interagir com os outros. O conceito de instanciar objetos das classes (módulos) permite que condições iniciais sejam satisfeitas na criação de cada instância de um módulo, e o reaproveitamento de código torna a tarefa de programar mais produtiva.

A divisão por módulos também permitiu que a equipe trabalhasse em trechos de códigos diferentes, por integrante, e através da definição de interfaces e boas normas, foi possível produzir código válido sem precisar saber de todos os detalhes das outras partes do programa, dado que essas também devem funcionar como unidades separadas e manter suas condições de validez.

3 - Apresentação da implementação

Classe Compiler (definida em Compiler.h)

A classe Compiler é o módulo que implementa um programa montador para a máquina Simple86. Ela implementa streams de entrada e saída, assim como um vetor de objetos do tipo Instruction onde o programa é guardado. Os valores e métodos públicos da classe são apresentados a seguir:

```
Compiler(ifstream* input, ofstream* output, bool verboseEnabled)
```

É o construtor da classe Compiler. Ele define as streams de entrada e saída (input e output), garante que o vector program está vazio e inicializa outras variáveis, como o modo de saída textual e detalhes internos.

```
void readProgram(ifstream* input)
```

Lê um programa de um arquivo de texto de entrada e popula o vetor program do objeto. Representa a primeira passada de um montador. Cada linha de código do arquivo de entrada deve conter uma instrução. As instruções são processadas para o formato do objeto Instruction. Labels ainda não são decodificados. Endereços de instruções são inferidos pela soma dos tamanhos das instruções já processadas, os operandos que referenciam labels são deixados intactos.

```
void resolveLabels(vector<Instruction>& instructions)
```

É a segunda passada do compilador. Substitui todos os labels e variáveis no código com o endereço de memória correspondente no código binário final. Descobre o

endereço que cada label e variável representa e corrige os operandos que os referenciam.

`void compile()`

Método principal da classe. Aplica a primeira (readProgram) e segunda (resolveLabels) passadas no programa recebido, gerando o código binário do Simple86 (writeBin). Case o modo *verbose* esteja ativo, também usa a saída padrão para imprimir o programa gerado. Ao final do método as streams de entrada e saída são fechadas.

`void writeTextOutput(vector<Instruction> program)`

Imprime o resultado do processamento e das duas etapas de compilaçã, em texto.

`int16_t bitSpaceToBytes(int16_t bits)`

Faz uma conta simples para transformar um número de bits em bytes. Retorna um resultado inteiro com o resto perdido.

`void writeBin(vector<Instruction> toWrite)`

Transforma o vetor de objetos Instruction recebido em uma saída em binário para o arquivo de saída. O vetor continua intacto.

`void debugReceivedInstruction(Instruction& i)`

Imprime a Instruction formatada e os seus conteúdos atuais. É usado se o modo *verbose* estiver ativo.

A classe Compiler guarda o programa recebido e todo o resultado de seu processamento durante as etapas da compilação num objeto de vector do C++, esse possui objetos Instruction em suas posições. As propriedades de cada Instruction vão sendo alteradas durante o avanço, e no final da segunda etapa, todos esses objetos representam instruções prontas para serem traduzidas para binário.

“vector” foi usado por ser dinamicamente expansível e fácil de ser iterado, além de que é um tipo genérico, e foi inicializado para guardar objetos da classe Instruction.

O compilador, em seu segundo passo, armazena as words reservadas pelo comando “dw” após o último byte de instrução do programa, e o programa sempre

começa, na memória, na posição 0. As words reservadas apontam, então, para espaços imediatamente após todas as instruções do programa na memória, e não é necessário recomputar o endereço de todas as intruções e jumps após o resolvimento das alocações, ou preocupar-se com a pilha, se fosse o caso de armazenar em outro lugar da memória sem ser imediatamente antes do programa ou após.

Como o endereço inicial do programa sempre será 0, o primeiro word do binário é, também, sempre 0, que será lido para o IP da máquina Simple86, e indica o início do programa.

Classe Instruction (definida em Instruction.h)

A classe Instruction é o objeto que representa as várias instruções e commandos da arquitetura Simple86. Seus atributos são: string fullText (uma linha lida da entrada), string id (código da instrução ou label em formato de string), string opA e string opB (operandos da instrução em formato de string), InstructionType type (tipo de instrução, definido por um enum), InstructionCode code (código da instrução, de acordo com as especificações, inferido de id), OperandType opType (tipos dos operandos usados, de acordo com as especifiações), int16_t address (endereço) e int16_t size (tamanho em bits da instrução). Nota-se que labels e variáveis, como as definidas por DW, também são definidas através desse objeto.

Um objeto Compiler inicializa objetos da classe Instruction com linhas recebidas (cada linha para cada objeto) do arquivo de entrada, e o Instruction faz um pré-processamento dessa string, para ser completado pelo Compiler em seus passos posteriores. A classe Compiler manipula esses objetos a todo momento para chegar ao resultado final.

Os operandos de Instruction são do tipo string pois eles podem representar tanto números imediatos e endereços, quanto labels. O Compiler trata do bom manuseio dessas strings na fase final de escrita para binário, onde essas são convertidas para números, e esses transformados para little endian.

`enum OperandType`

Enumeração dos tipos de combinações de operandos possíveis, especificados pelo padrão definido para a máquina Simple86.

`enum InstructionCode`

Enumeração das instruções definidas para a a máquina Simple86.

`enum RegisterCode`

Enumeração dos registradores acessíveis por assembly, especificados pelo padrão definido para a máquina Simple86.

`enum InstructionType{ LABEL, INSTRUCTION, VAR }`

Enumeração das possibilidades de representação do objeto Instruction. LABEL se refere a uma posição na memória. INSTRUCTION se refere a uma instrução, VAR se refere a uma declaração de word. LABELs e VARs serão usados para computação de seus endereços no segundo passo da compilação, não são instruções que serão convertidas para binário.

`Instruction(string full)`

É o construtor da classe Instruction. Recebe uma linha de código de assembly e monta o objeto que representa a instrução presente na string. Faz uma normalização na string e consegue ler qual tipo de comando essa é, qual instrução representa, os operandos, o tamanho de acordo com a especificação. O endereço ainda não é definido, é arrumado pelo compilador.

`InstructionCode getInstructionCode(string id)`

Recebe uma string contendo o nome de instrução. Retorna o código da mesma na representação interna dessa estrutura de dados (Instruction).

`RegisterCode getRegisterCode(string id){`

Recebe uma string representando um registrador. Retorna o código do mesmo na classe Instruction.

`int16_t getInstructionSize(InstructionCode code)`

Recebe o código de uma instrução e retorna o tamanho em bits da mesma.

`OperandType determinOperandType(string a, string b){`

Recebe duas strings, cada uma contendo um dos operandos. Retorna o código do tipo de operandos usados Consegue detectar imediatos, memória, registradores, e ausência de operandos.

`string debugInstruction()`

Imprime para a saída padrão uma string que representa o resultado do estado atual desse objeto Instruction.

Main e montador (mainCompiler.cpp)

O arquivo mainCompiler.cpp é o ponto de entrada do compilador Simple86, interpreta os parâmetros recebidos (especificados na documentação, arquivo de entrada, flag de saída do resultado do processamento para texto “-v”, e arquivo de saída binário “-o <nome>”, esses últimos dois opcionais) em qualquer ordem e monta o objeto Compiler, manda-o processar o código recebido.

4 – Compilação

Esse projeto contêm arquivos de código fonte tanto do Compilador (detalhados nessa documentação), como arquivos do Emulador atualizado para ler onde é o início do programa da primeira word na memória. Os detalhes do Emulador estão na documentação do primeiro trabalho prático.

O programa foi escrito no padrão C++11, e é necessário um compilador com suporte à tal padrão da linguagem C++. Tudo foi testado num ambiente Linux, com o g++ 5.3.1. O arquivo Makefile trata de chamar o compilador g++ com os parâmetros necessários para a compilação correta do projeto.

```
g++ -Wall -std=c++11 mainEmulator.cpp -o Simple86_Emulator
g++ -Wall -std=c++11 mainCompiler.cpp -o Simple86_Compiler
```

Para compilar o projeto todo, “make” irá produzir tanto o compilador quanto o emulador. “make emulator” produzirá apenas o binário Simple86_Emulator, que é o emulador da máquina Simple86 e “make compiler” apenas o binário Simple86_Compiler, o compilador.

5 – Testes

Na pasta tst, desse projeto, se encontram arquivos assembly prontos para serem compilados e testados no emulador.

“fatorial.asm” produz o fatorial do número recebido por entrada de teclado.

“malloctest.asm” testa a alocação de varíaveis usando o comando “dw”, armazena 2 e 3, carrega-os e soma-os, deve produzir 5.

“reverse.asm” usa a pilha para inverter 3 entradas recebidas pelo teclado, e imprimi-las de trás para frente.

“tp1ex1.asm” é o teste inicial do emulador passado no enunciado da primeira parte do trabalho.