1º Trabalho Prático

MC404 E/F - Organização Básica de Computadores e Linguagens de Montagem

Ricardo Pannain
(adaptação de material produzido por Daniel Aranha, Rafael Auler e Edson Borin)

2º Semestre de 2015

1 Introdução

O trabalho consiste em implementar em C um método de tradução de uma linguagem de montagem simples para uma representação intermediária adequada para simulação em uma arquitetura moderna.

2 Método

Como já visto em aula, um montador (*assembler*) é uma ferramenta que converte código em linguagem de montagem (*Assembly*) para código em linguagem de máquina. Neste trabalho, você irá implementar um montador para a linguagem de montagem do computador IAS, que produza como resultado um mapa de memória para ser utilizado no simulador do IAS (o simulador pode ser encontrado no ambiente compartilhado da disciplina.

A linguagem de montagem utilizada será a linguagem simbólica do IAS apresentada em sala, consistindo em um conjunto de apenas 20 instruções e 5 diretivas. A linguagem hipotética não é sensível ao caso, não havendo diferenciação entre maiúsculas e minúsculas. Você deve utilizar o material "Programando o computador IAS" como referência do conjunto de instruções e diretivas.

3 Especificação

O arquivo de saída do montador é um mapa de memória que contém linhas no formato:

```
AAA DD DDD DD DDD
```

Na linha acima, `AAA` é uma sequência de 3 dígitos hexadecimais que representa o endereço de memória, totalizando 12 bits. Já `DD DDD DD DDD` é uma sequência de 10 dígitos hexadecimais, que totaliza 40 bits e representa um dado ou duas instruções do IAS, conforme já visto em aula. Note que existem caracteres de espaço na linha, num total de exatos 4 espaços. Apesar de o simulador tolerar outras representações, é importante que seu montador produza um mapa de memória **exatamente** nesse formato para permitir a execução dos casos de teste. Não deve haver caracteres extras ou linhas em branco, apenas linhas no formato acima.

O arquivo de entrada do montador deve ser um arquivo de texto tal que cada linha deve ser como uma das seguintes:

```
[rotulo:] [instrucao] [# comentário]
```

ou

```
[rotulo:] [diretiva] [# comentario]
```

Nas linhas acima, os colchetes determinam elementos opcionais, ou seja, qualquer coisa é opcional, podendo então haver linhas em branco no arquivo de entrada, ou apenas linhas de comentário, ou linhas só com uma instrução, etc. É possível que haja múltiplos espaços em branco no início ou fim da linha ou entre os elementos. Tente estruturar seu montador em um analisador

léxico (que recupera sequências contíguas de caracteres da entrada) e um analisador sintático (que analisa a forma das sequências), conforme visto em aula para manter o código organizado. Como exemplo, as seguintes linhas são válidas num arquivo de entrada para o montador:

```
vetor:
vetor:    .word 10
vetor:    .word 10 # Comentario apos diretiva

.word 10
.word 10 # Comentario apos diretiva
# Comentario sozinho

vetor: ADD M(0x100)
vetor: ADD M(0x100) # Comentario apos instrucao

ADD M(0x100)
```

e as seguintes linhas são inválidas:

```
vetor: outro_vetor: mais_um_vetor:

vetor: .word 10 ADD M(0x100)

.word 10 .align 5

vetor: ADD M(0x100) .word 10

ADD M(0x100) ADD M(0x200)
```

Algumas regras gerais do montador:

- É possível que um programa possua palavras de memória com apenas uma instrução (veja a diretiva `.align`). O seu montador deve completar a palavra não preenchida com zeros.
- O montador deve ser insensível ao contexto, devendo aceitar letras maiúsculas e minúsculas sem fazer distinção entre elas. Os casos de teste serão bastante mistos, fazendo uso de letras maiúsculas e minúsculas alternadamente.
- Não é necessário (e inclusive é não-recomendável) tratar acentos no montador. No caso de comentários, até pode se usar palavras acentuadas, visto que os comentários são descartados no processo de montagem. Mas para rótulos, não é necessário aceitar o uso de acentos. Nos casos de teste, não haverá nenhum acento, nem nos comentários.
- O executável do montador deve aceitar 2 argumentos, sendo que o primeiro é obrigatório e o segundo é opcional. O primeiro parâmetro deve ser o nome de um arquivo de texto que representa a entrada para o montador, ou seja, tal arquivo deve estar de acordo com as regras aqui explicadas, contendo instruções, rótulos, etc. O segundo parâmetro, facultativo, é o nome de um arquivo de saída a ser gerado pelo montador. Caso o segundo argumento não seja fornecido, deve-se gerar um arquivo de saída exatamente com o mesmo nome do arquivo de entrada, adicionando-se a extensão **.hex**. Dois exemplos válidos são: **./raXXXXXX entrada.in saida.out**, que lê um arquivo denominado **entrada.in** e gera o mapa de memória no arquivo **saida.out**, e **./raXXXXXX prog.ias**, que lê um arquivo de

nome **prog.ias** e cria um arquivo de nome **prog.ias.hex**, contendo o mapa de memória gerado pelo montador.

As próximas seções descrevem as regras sintáticas referentes à linguagem de montagem.

4 Sintaxe

4.1 Comentários

Comentários são cadeias de caracteres que servem para documentar ou anotar o código. Essas cadeias devem ser desprezadas durante a montagem do programa. Todo texto à direita de um caractere # (denominado cerquilha) é considerado comentário.

4.2 Rótulos

Rótulos são compostos por até 100 caracteres alfanuméricos e podem conter o caractere **_** (*underscore*). Um rótulo é definido como uma cadeia de caracteres que deve ser terminada com o caractere **:** (dois pontos) e não pode ser iniciada com um número. Exemplos de rótulos válidos são:

```
varX:
var1:
_varX2:
laco_1:
__DEBUG__:
```

Exemplos de rótulos inválidos são:

```
varx::
:var1
1var:
laco
```

4.3 Diretivas

Todas as diretivas da linguagem de montagem do IAS começam com **.** (ponto). As diretivas podem ter um ou dois argumentos. Tais argumentos podem ser dos tipos apresentados na tabela 1.:

Tipo	**Especificação**
CONSTANTE	quaisquer valores numéricos.
CONSTANTE+	valores numéricos não-negativos (inclui o zero)
CONSTANTE*	valores numéricos positivos, i.e., maiores do que 0.
NOME	até 100 caracteres alfanuméricos e , mas não pode começar com número.

Tabela 1: Tipos de argumentos

A Tabela 2 abaixo apresenta a sintaxe das diretivas de montagem e os tipos de seus argumentos. A descrição do comportamento de cada uma das diretivas está na apostila de programação do computador IAS ou no material de aula.

Apenas as diretivas `.wfill` e `.word` podem receber um rótulo como operando, pois cada rótulo define uma constante implicitamente, determinada pelo endereço da palavra que o rótulo anota. Todas as diretivas podem receber como operandos constantes simbólicas declaradas com a diretiva `.set`, desde que as restrições do operando sejam obedecidas pela constante. Sugere-se expandir as constantes simbólicas em um passo anterior de pré-processamento, para melhor organização do programa. Constantes simbólicas declaradas com `.set` precisam ser declaradas antes do primeiro uso. A diretiva `.align` apenas terá efeito sobre a próxima linha do programa sendo montado.

Sintaxe	**Argumento 1**	**Argumento 2**
.org	CONSTANTE+	
.align	CONSTANTE*	
.wfill	CONSTANTE*	NOME ou CONSTANTE
.word	NOME ou CONSTANTE	
.set	NOME	CONSTANTE

Tabela 2: Diretivas válidas e seus argumentos.

Desta forma, a ocorrência `.align` 0x1 vista em sala tem um efeito mais intuitivo, exigindo o alinhamento apenas da palavra seguinte.

4.4 Instruções

As instruções do programa seguem o seguinte formato:

`mnemônico M([rótulo|endereço])`.

O mnemônico da instrução informa ao montador qual é a instrução que deve ser gerada no mapa de memória. O campo seguinte (opcional) informa qual o endereço a ser colocado no campo endereço da instrução.

Note que neste segundo campo é possível usar um rótulo ou endereço. Caso seja usado endereço, esse deve ser informado dentro de parênteses, após a letra `M` – como exemplo, `M(0x200)`. Endereços que diferenciam a instrução à esquerda da instrução à direita devem referenciar os bits específicos, na forma `M(0x200,0:19)`. No caso de se utilizar rótulo, esse deve ser informado utilizando a mesma sintaxe, por exemplo `ADD M(varX)`.

O segundo campo de uma instrução é opcional pois algumas instruções não exigem que se passe um endereço, por exemplo a instrução `RSH`. Nesse caso, ainda que na linguagem de montagem não seja passado endereço, é preciso preencher os *bits* referentes ao endereço no mapa de memória - isso deve ser feito colocando-se zeros neste campo. A Tabela 3 abaixo apresenta os mnemônicos válidos e a instruções que devem ser emitidas para cada um dos casos.

Mnemônico	Instrução a ser emitida
LDMQ	LOAD MQ
LDMQM	LOAD MQ,M(X)
STR	STOR M(X)
LOAD	LOAD M(X)
LDN	LOAD -M(X)
LDABS	LOAD |M(X)|
JMP	JUMP M(X,0:19) se operando for rótulo/endereço de instrução à esquerda.
JMP	JUMP M(X,20:39) se operando for rótulo/endereço de instrução à direita.
JGEZ	JUMP+ M(X,0:19) se operando for rótulo/endereço de instrução à esquerda.
JGEZ	JUMP+ M(X,20:39) se operando for rótulo/endereço de instrução à direita.
ADD	ADD M(X)
ADDABS	ADD |M(X)|
SUB	SUB M(X)
SUBABS	SUB |M(X)|
MUL	MUL M(X)
DIV	DIV M(X)
LSH	LSH
RSH	RSH
STM	STOR M(X,8:19) se operando for rótulo/endereço de instrução à esquerda.
STM	STOR M(X,28:39) se operando for rótulo/endereço de instrução à esquerda.

Tabela 3: Mnemônicos válidos e as instruções que devem ser emitidas para cada um dos casos.

O programa de tradução deve ser capaz de realizar as fases de análise e síntese, mantendo informação intermediária armazenada em estruturas de dados e interrompendo a execução mediante a ocorrência de qualquer erro. A escolha apropriada de estruturas de dados faz parte do escopo do trabalho.

4.5 Erros

O tratamento de erros deve abranger a lista não-exaustiva abaixo:

- Erros léxicos causados pela presença de caracteres inválidos no programa de entrada;
- Erros sintáticos causados pela presença de linhas que desviam por qualquer motivo da gramática elementar vista em sala;
- Erros semânticos causados por declarações ausentes ou repetidas, desvios condicionais e incondicionais para rótulos inexistentes.

5 Entrega e Avaliação

A linguagem de programação a ser utilizada para desenvolver o montador deve ser, obrigatoriamente, a linguagem C. Não serão aceitos trabalhos que façam uso de alguma biblioteca não-padrão, ou seja, apenas podem ser utilizadas as bibliotecas acessíveis ao GCC. O trabalho deverá ser entregue **em dupla**, que poderão ter arguição adicional. Caso haja qualquer tentativa de fraude, como plágio, todos os envolvidos receberão nota 0 na média da disciplina. Em princípio, todos os valores numéricos aceitos pelo montador na linguagem de montagem devem ser hexadecimais, do tipo `0xN` com um sinal negativo opcional a ` sua frente (`-0xN`). No entanto, um bônus será concedido para os alunos que aceitarem outras bases numéricas.

Foi compartilhado com você um diretório (no Google Drive) nomeado com seu respectivo RA; lá você encontrará outro diretório chamado "Trabalho 1" – você deve colocar os arquivos de sua solução neste último diretório até o dia **08 de outubro de 2015, até às 15hs59min**. A entrega consistirá em:

- Código-fonte completo e comentado, além de um arquivo **Makefile** que gere o executável do montador como regra padrão; Todos esses arquivos devem ser comprimidos em um único arquivo **.tar.gz**, com o nome **raXXXXXX.tar.gz**, em que **XXXXXX** é o número de seu **RA**. O executável do montador, que será gerado pelo **Makefile**, deve ser nomeado **raXXXXXX** (sem extensão mesmo!). Em caso de trabalho feito em dupla, utilize a convenção **raXXXXXX_raYYYYYY** tanto no arquivo comprimido quanto no executável do montador e envie o arquivo com a conta de e-mail de um dos envolvidos;
- Programas de exemplo que demonstrem o funcionamento correto do montador.

6 Bônus

6.1 Bases numéricas

O montador deve aceitar números hexadecimais, conforme explicado acima. Contudo, é interessante que ele seja capaz de ler números em diferentes bases, como decimal, binária e octal. Será concedido 1 ponto extra para os montadores capazes de lidar com as 4 bases: hexadecimal, decimal, binária e octal.

Os caracteres que identificam uma determinada base podem preceder um número. Se um número iniciar com os caracteres `0x`, o montador deve esperar um número na base hexadecimal e aceitar os caracteres `a`, `b`, `c`, `d`, `e` e `f` como algarismos válidos, além dos numerais de `0` a `9`. Novamente: o montador não deve diferenciar letras maiúsculas e minúsculas durante o processamento de números.

Se um número for precedido pelos caracteres `0b`, o montador deve esperar um número na base binária. Finalmente, um número precedido por 0o indica que os próximos caracteres correspondem a um número na base octal (de `0` a `7`).

Caso nenhum caractere especial preceda o número, deve ser considerado que esse número está na base decimal. Note que se pode misturar todas as bases no mesmo código. Por fim, caso haja alguma incoerência da base com o número no código, por exemplo um binário com dígito 2 ou um hexadecimal com o dígito `Z`, o montador deve apontar um erro e parar a montagem.

Exemplos de números em diferentes bases que devem ser aceitos:

```
varw: .word 0x10    #Constante 16 == 0x10 em hexa
varx: .word 22      #Constante 22 decimal
LOAD M(0b1011)      #Carrega uma palavra da posição 11 == 1011 em
                    #binário
varz: .word 0o377   #Constante 255 == 377 octal
```

6.2 Desempenho

Além disso, será premiado o trabalho que obtiver o melhor desempenho de montagem. A análise de desempenho será realizada pelo professor com programas de *benchmarking* suficientemente extensos, utilizando o mesmo nível de otimização - `O2` para a compilação de todos os trabalhos, e apenas se aplicará a trabalhos que produzem o resultado correto dos programas de *benchmarking*. Desta forma, a bonificação total do trabalho é de no máximo 2 pontos, divididos em até 2 alunos diferentes, a serem aplicados exclusivamente na primeira avaliação dissertativa.

7 Dicas

Uma função recomendada da biblioteca padrão da linguagem de programação C que pode reduzir substancialmente o trabalho envolvido é `strtok()` para implementação do analisador léxico. Observe também os parâmetros para formatação de entrada e saída das funções `printf()`/`scanf`.