Montador e simulador risc

Renato Borges Boaventura[1]

[1] Instituto Federal de Minas Gerais - Formiga.MG (IFMG)

RESUMO

Este artigo tem como propósito mostrar em linhas gerais de como o código fora implementado e quais as escolhas feitas durante sua implementação. Serão mostradas cada TAD usadas no programa de simulador e do montador, e em que situações cada um delas foram usadas.

PALAVRA CHAVE:

ARQUITETURA; ALGORÍTMOS; PROGRAMAÇÃO; TAD.

INTRODUÇÃO

No início desse artigo, mostrará de forma breve de como fora feito o trabalho. Será mostrando as TADs e as estruturas utilizadas, como o programa deve ser executado via linha de comando e quais as principais escolhas feitas na construção do algoritmo. O programa foi escrito utilizando o editor de texto *Sublime Text*, e todos os testes foram realizados no sistema operacional *linux*.

DESENVOLVIMENTO

Usando o campo de bit em na linguagem de programação c, foram feitas três estruturas de 32 bits que variam de acordo com as instruções geradas pelo montador. A estura *operacao3* contém quatro campos de 8 bits cada, a estrutura *operacao2* contém três campos, no qual o mais significativo contém o opcode, mais os 16 bits do campo central e mais 8 bits no campo menos significativo. E, por ultimo, temos a estrutura *operacao4*, que contém 8 bits no campo mais significativo, e 24 bits no restante.

Primeiramente o programa lê o endereço inicial, depois transforma cada linha de instrução em um número decimal inteiro. Após a conversão do binário presente no arquivo de entrada, é feito a conversão do número em todas as três estruturas já mencionadas. Com o opcode e o *programm counter* obtidos, é feito a verificação de qual instrução será executada de acordo com o opcode. O programa entrará em um loop até que se encontre um halt (instrução para finalizar o programa).

O montador utilizou duas TADs previamente feitas: *hash_externa* e *lista_hash*, pois no primeiro passo do montador, todos os labels encontrados, são colocados numa

hash com seu respectivo *pc*. No segundo passo, é feita a conversão das strings que contém os nomes dos opcodes e os possíveis e respectivos registradores e números, para um binário de 32 bits. Ao final da execução, é gerado um arquivo de saída, contendo as instruções em binário, que poderá ser utilizado como arquivo de entrada pelo simulador. Vale ressaltar que o endereço inicial gerado no arquivo de saída será sempre 0.

Para compilar o simulador, é necessário passar o parâmetro *-lm* ao final do comando. O simulador necessita de apenas um arquivo de entrada, os resultados e alterações no programa, serão mostrados na tela do usuário. Para executar o montador, é necessário passar o nome do arquivo de entrada, contendo corretamente as instruções, e o nome do arquivo de saída.

Segue abaixo a tabela com instruções extras do trabalho, e em seguida as descrições de cada uma delas. Não será dissertado sobre as instruções que o próprio trabalho requer, pois todas elas foram implementadas exatamente como descritas no enunciado do trabalho.

Tabela 1. Tabela de instruções

Instruções	Opcode (em decimal)	Formato no montador	Formato no simulador
mult	37	mult rc, ra, rb	mult ra, rb, rc
div	38	div rc, ra, rb	div ra, rb, rc
elev	39	elev rc, ra, rb	elev ra, rb, rc
somai	40	somai rc, valor1, valor2	somai valor1, valor2, rc
subi	41	subi rc, valor1, valor2	subi valor1, valor2, rc
inc	42	inc rc, ra, rb	inc ra, rb, rc
dec	43	dec rc, ra, rb	dec ra, rb, rc
senzero	44	senzero ra, rb, end	senzero ra, rb, end

mult

Operação:
$registrador[rc] = registrador[ra] * registrador[rb]$
Descrição:
Multiplica o conteúdo do registrador a pelo conteúdo do registrador b, guardando o resultado no registrador c.

div

Operação:
$registrador[rc] = registrador[ra] / registrador[rb]$
Descrição:
Divide o conteúdo do registrador a pelo conteúdo do registrador b, guardando o resultado no registrador c.

elev

Operação:
$registrador[rc] = pow(registrador[ra], registrador[rb])$
Descrição:
Eleva o conteúdo do registrador a pelo conteúdo do registrador b, guardando o resultado no registrador c.

somai

Operação:
$registrador[rc] = valor1 + valor2$
Descrição:
Soma os dois valores passados pelo programador, e armazena o resultado no registrador rc.

subi

Operação:
$registrador[rc] = valor1 - valor2$
Descrição:
Subtrai os dois valores passados pelo programador, e armazena o resultado no registrador rc.

inc

Operação:
$registrador[rc]++$
Descrição:
Incrementa o valor contido no registrador rc.

dec

Operação:
$registrador[rc]--$
Descrição:
Decrementa o valor contido no registrador rc.

senzero

Operação:
$registrador[ra]--$
$if(registrador[ra]! = 0)$
$PC = end$

Descrição:
Decrementa o dado presente no registrador e desvia se o resultado da operação for diferente de zero.

Foram realizados diversos testes para testar a robustez do código. Fora utilizado diversas variações de um código que calcula o fatorial de 5, além de outros testes que envolvia diversas operações aritméticas. A seguir, uma das versões do programa que calcula o fatorial do número 5:

*lcl r*1, 1
*lcl r*4, 5
*label*2 :
*beq r*4, *r*0, *label*1
*mult r*1, *r*4, *r*1
*dec r*4
*j label*2
*label*1 :
halt

CONCLUSÃO

Após o término do programa, conclui-se que a implementação dos simuladores e montadores são de extrema. Apesar das dificuldades encontradas para a abstração e implementação do algoritmo, o campo de bit foi de essencial ajuda no que tange a manipulação dos bits e sua organização, proporcionando assim uma fácil implementação e decodificação do código. A implementação do trabalho, foi de extrema importância para a compreensão de como os computadores funcionam ”por d’baixo dos panos”.

Referências

PATTERSON, David A.; HENNESSY, John L. Organização e Projeto de Computadores: A interface Hardware/Software, 4. ed. São Paulo: Campus, 2005.

ANENBAUM, A. S. Organização Estruturada de Computadores. 5a edição. São Paulo: Pearson Prentice Hall, 2007.