UM MONTADOR PARA UM SIMULADOR DE PROCESSADORES VETORIAIS

C.L. Sales L.E. Favre M.C.S. de Castro C.L. Amorim
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ
Caixa Postal 68511 - Ilha do Fundão - Rio de Janeiro 21945 - RJ

RESUMO

Em geral, as linguagens de Montagem exibem uma regra simples de formação de instruções, utilizando mnemônicos para definir a operação e os operandos sobre os quais ela se aplica. Devido à relação direta entre mnemônicos e códigos de operação, os projetistas de montadores para essas linguagens normalmente usam tabelas e códigos de operação, para facilitar o processo de construção e manutenção do montador.

Este artigo discute a implementação de um Montador feito para a linguagem simbólica do processador **CRAY-1** que não possui essas características. Nessa Linguagem, praticamente todas as instruções não possuem mnemônico e a relação entre as instruções e os códigos de operação da linguagem de máquina não é direta. Devido a essa particularidade, o desenvolvimento de um eficiente montador exige solução especial que é apresentada.

ABSTRACT

Most of Assembly Languages have a simple rule to build instructions based on mnemoni s which define the operation and their operands. Due to the direct relation between mnemonic and operation codes, the assembler designers for these languages use tables for the opcodes in order to simplify the process of assembler construction and maintenance.

In this article we discuss the implementation of an Assembler for the **CRAY-1** Assembly Language which doesn't have these features. In that language, most of the instructions don't have mnemonic and the relation between instructions and opcodes isn't direct. Due to this particularity, the development of an eficient Assembler requires a special solution which is presented.

1. INTRODUÇÃO

A Linguagem de Montagem do processador **CRAY-1** apresenta algumas características não convencionais, tais como a presença de instruções vetoriais e uma regra de formação das instruções que não utiliza mnemônicos para definir a operação. Por exemplo, uma instrução de transferência entre registradores de uma linguagem convencional seria:

MOV A,B, onde a operação é especificada por MOV e os operandos são A e B.

No **CRAY-1**, uma instrução equivalente para os registradores escalares seria:

SiSk, onde os operandos são S_i e S_k (i,k = 0..7), e o mnemônico não existe, ou seja, nesse caso a operação é definida pela posição relativa dos operandos.

Esta particularidade da linguagem, como veremos posteriormente, acrescenta uma maior complexidade no desenvolvimento de um eficiente montador para a linguagem simbólica do **CRAY-1**. Neste trabalho, discutimos o projeto da implementação de um montador capaz de gerar código para o simulador de arquiteturas compatíveis com a do **CRAY-1** |1| que se encontra disponível no laboratório de processamento paralelo/distribuído da COPPE/Sistemas. Após uma breve descrição da arquitetura do **CRAY-1** e de seu conjunto de instruções, discutimos a solução do problema relativo à identificação do código de operação que irá influenciar a concepção e a implementação do montador. Nas seções seguintes serão apresentados os resultados obtidos e o atual estágio do trabalho.

2. A ARQUITETURA DO **CRAY-1**

Descreveremos os aspectos da arquitetura do **CRAY-1** que consideramos relevantes para o entendimento do conjunto de instruções. Um melhor conhecimento sobre a máquina pode ser obtido em |4|.

A arquitetura do **CRAY-1** (Figura 1) é composta por uma memória principal com 1M palavras de 64 bits, organizadas em 16 bancos de 64K intercalados, com tempo de acesso de 50 ns.

O arquivo de registradores contém oito registradores de endereço de 24 bits (A0 a A7), oito registradores escalares de 64 bits (S0 a S7), oito registradores vetoriais (V0 a V7), cada um com 64 elementos de 64 bits, dois conjuntos de registradores intermediários (B e T) com 64 elementos cada, sendo utilizados como "buffers" entre a memória principal e os conjuntos de registradores de endereço e escalares.

O **CRAY-1** também possui doze unidades funcionais que trabalham independentemente e que foram divididas em quatro grupos: unidades de endereçamento, escalares, vetoriais e de ponto-flutuante.

A unidade de controle possui quatro "buffers" de instruções, cada um com 64 pacotes de instruções de 16 bits, com facilidade de pré-carregamento de instruções e é responsável pelo envio de instruções para as unidades funcionais correspondentes aos modos de processamento existentes (de endereço, escalar e vetorial).

2.1 O Conjunto de Instruções do **CRAY-1**

O **CRAY-1** possui uma linguagem de montagem que traduz a variedade de operações vetoriais, escalares e de controle que podem ser realizadas.

As instruções possuem um ou dois pacotes (16 ou 32 bits) e têm a seguinte forma geral:

4	3	3	3	3	16
g	h	i	j	k	m

onde o primeiro pacote é composto dos campos **g**, **h**, **i**, **j** e **k** e o segundo pacote contém o campo **m**.

Existem cinco tipos de formatos, onde os campos **g** e **h** formam, com algumas exceções, o código de operação. Os campos **i**, **j** e **k** definem os índices dos registradores dentro dos conjuntos, ou formam, junto com o campo m, constantes/endereço nas instruções de dois pacotes. Exemplos de instruções com seu código de máquina e de montagem associados são mostrados a seguir.

1. Formato Lógico/Aritmético

   Geral: 171ijk V_i V_j + FV_k "Floating sums of (V_j) and V(k) to V_i"

   Ex.1.: 171456 V_4 V_5 + FV_6

   Ex.2.: 171057 V_0 V_5 + FV_7

2. Formato Máscara/Deslocamento

   Geral: 056i0k S_i S_i < A_k "Shift(S_i) left (A_k) places to S_i"

   Ex.1.: 056103 S_1 S_1 < A_3

3. Formato Constante Imediata

   Geral: 031i10 A_i - 1 "Transmit - 1 to Ai"

   Ex.1.: 031700 A_7 - 1

4. Formato de Transferência à Memória

   Geral: 11hi000 ,A_h A_i "Store (Ai) to (Ah)"

   Ex.1.: 1112000 ,A_1 A_2

5. Formato Desvio

   Geral: 017ijkm JSM (exp) "Branch to exp if (S_o) negative"

   Ex.1.: 017000 JSM (3 * 2 - 6)

Os índices **i**, **j**, **k** e eventualmente **h**, identificam o registrador desejado dentro do grupo de registradores **A**, **S**, **V**, **B** ou **T**.

A expressão (exp) pode ser uma constante imediata ou uma expressão aritmética (soma, subtração, multiplicação e divisão).

No primeiro item (formato lógico/aritmético) existem duas formas da mesma instrução na linguagem de montagem gerando códigos de operação iguais. Essa ambiguidade é resolvida pela identificação dos registradores de trabalho através dos índices **i**, **j** e **k**. Situações semelhantes são encontradas nos outros exemplos de instruções acima.

3. METODOLOGIA DE IMPLEMENTAÇÃO DO MONTADOR

Na primeira etapa do trabalho foi feita uma análise do conjunto de instruções do **CRAY-1** |3| e constatou-se a inexistência de

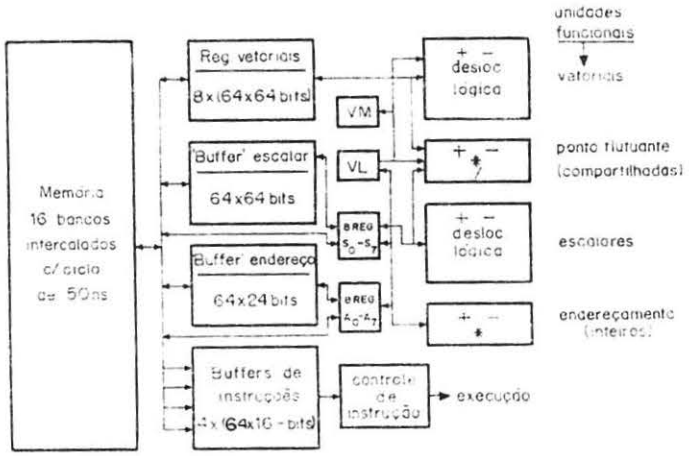

Figura 1. A Arquitetura do Processador **CRAY-1**

mnemônicos, mais especificamente, as instruções são compostas de operandos e caracteres especiais, sendo que esses dois elementos definem o código de operação na linguagem de máquina. Os quatro exemplos abaixo ilustram algumas das características especiais da linguagem:

1) A instrução S_i (exp) ,

- transmite (exp) para S_i.
- contém o operando $S_i(s_o...S_7)$ e uma expressão aritmética.
- não existe mnemônico ou caracteres especiais.
- o "opcode" em octal pode ser "040" ou "041" dependendo do valor de (exp).

2) A instrução S_i 0 ,

- Zera S_i.
- contém o operando $S_i(S_o...S7)$ e a constante "0".
- não contém mnemônico.
- o "opcode" em octal é "043".

3) As instruções S_o S_i < (exp) e S_i S_i < (exp)

- a primeira faz um deslocamento a esquerda de (exp) em (S_i) e guarda o resultado em S_o.
- a segunda faz um deslocamento a esquerda de (exp) em (S_i).
- o primeiro operando da primeira instrução é S_o, e na segunda instrução esse operando pode variar de S_1 a S_7.
- os "opcodes" em octal são "052" e "054" respectivamente.

4) As instruções

S_i S_j!S_k (soma lógica de (S_j) e (S_k) para S_i).

S_i S_k (Transmite (S_k) para S_i) ,
S_i S_j !S_B (Soma lógica de (S_j) e SB para S_i) ,
S_i SB!S_j (Idem, mas j ≠ 0) e
S_i SB (Transfere o SB para S_i) ,

onde SB é o bit de sinal.

- geram o mesmo "opcode" em octal "051".
- os índices **i**, **j** e **k** podem variar de 0 a 7.

Devido a essas características, descartamos a possibilidade de se utilizar uma tabela de "opcodes", onde o mnemônico serviria como chave de acesso. A aplicação de autômatos finitos, para analisar as instruções caractere a caractere, foi abandonada devido a quantidade de instruções e a correspondente complexidade de programação e depuração do montador. Outro inconveniente seria a pouca flexibilidade que se teria para alterar o conjunto de instruções processado pelo montador, caso desejado.

Diante dessas primeiras dificuldades, examinamos com mais cuidado a possibilidade de se montar uma tabela de "opcodes", na qual para cada instrução de montagem houvesse uma entrada na tabela e as alterações no conjunto de instruções se restringissem, na medida do possível, a alterações nessa tabela. Infelizmente, as dificuldades permaneciam: inexistência de mnemônico e grande número de instruções, o que tornaria o acesso à tabela ineficiente. Para contornar esses dois problemas criamos **representações internas** ao montador denominadas **mnemônicos** para as instruções e usamos pesquisa binária para acessar à tabela.

A criação interna de mnemônicos baseou-se na

Tabela 1

Instrução	Mnemôn	Opcode	Tam.	Operandos	Ambig
Ai Ak	AA	030	1	ik	0
Ai Aj + Ak	AA+A	030	1	ijk	0
Ai Aj+1	AA+	030	1	ij	0
Si Sj!Si&Sk	SS!S&S	050	1	ijik	0
Ai (exp)	A			i	1

análise comparativa entre as instruções do **CRAY-1** e as instruções de máquinas convencionais. Nas últimas, os operandos são variáveis e os mnemônicos são fixos e possuem uma relação de um para um com os "opcodes" da linguagem de máquina, podendo-se usá-los como chave de acesso a uma tabela de "opcodes" |5|. Nas instruções do **CRAY-1**, tentamos definir operandos e "mnemônicos". Observamos que:

- existem instruções específicas para os vários tipos de registradores. Por exemplo, as instruções "A_i(exp)," e "S_i(exp)", geram "opcodes" diferentes, "100" e "120" respectivamente.

- os operandos poderiam ser os índices que identificavam os elementos dos conjuntos de registradores, uma vez que, para cada (instrução, "opcode") os elementos variáveis serão os índices. Por exemplo, no segmento de programa abaixo:

  S_3 S_4

  S_5 S_6

Para essas instruções têm-se o par ("S_i S_k", "051"). Podemos, fazendo uma analogia com as instruções convencionais, tomar como operandos os elementos variáveis na instrução (**3** e **4** na primeira, e **5** e **6** na segunda) e considerar os elementos fixos como mnemônicos (**S** e **S**).

Assim, para cada instrução do **CRAY-1**, foi estabelecido como mnemônicos as letras (no exemplo, **SS**) e os caracteres especiais, pela ordem de ocorrência. Os operadores correspondem aos índices das letras de identificação dos conjuntos de registradores (Tabela 1).

Após a obtenção de um mnemônico para cada instrução, verificamos que para algumas instruções eles eram idênticos. Para resolver essas ambiguidades foi preciso ainda considerar o número de operandos e o valor das expressões, caso existissem.

Como podemos ver na (Tabela 1), além do tamanho (número de pacotes) das instruções, foi incluído um código para identificação de instruções ambíguas e um campo onde informamos quais e quantos operandos cada instrução contém, e em que ordem eles aparecem. A chave de pesquisa da tabela é o mnemônico. Foram previstos também pseudo-códigos para que o usuário forneça diretivas ao montador. Os pseudo-códigos implementados são equivalentes aos das linguagens de máquina convencionais tais como ORG (informa ao montador o endereço de memória onde deve ficar o programa), DSEG (informa ao montador o início das linhas de dados), etc. Por uma questão de uniformidade, os pseudo-códigos foram tratados como instruções comuns pelo montador, evitando apenas a geração de código.

Optamos por um montador de dois passos por possibilitar uma melhor modularidade, facilitando a divisão e a depuração do trabalho. No primeiro passo são detectados erros de sintaxe e é gerada a tabela de símbolos. Se não ocorrer nenhum erro, o segundo passo será executado, sendo então feitas consultas à tabela de símbolos para a geração do código objeto.

Em ambos os passos, a tabela de códigos de operação será consultada. No primeiro passo, verifica-se a existência da instrução e obtém-se o seu tamanho para os cálculos necessários à formação da tabela de símbolos. No segundo passo, consultamos as informações sobre os operandos para a geração do código objeto. Cada instrução analisada pelo montador tem letras e caracteres especiais separados para a obtenção da chave de pesquisa da tabela de "opcodes".

4. IMPLEMENTAÇÃO E RESULTADOS

O montador foi programado na extensão da linguagem PASCAL, entendida pelo compilador TURBO PASCAL, versão 3.0, e implementando em um computador PC-compatível. Uma versão do montador foi instalada também no DIGIREDE 8000, sistema operacional DIGIX.

Dois programas foram usados para testes, um programa de "sort" e um de resolução de

equações lineares, com 316 e 292 instruções, com tempos de montagem de 38s e 35s, sendo que o passo 1 do montador consumiu 11s e 10s, respectivamente.

Para uma primeira validação do montador, os códigos objetos gerados foram executados pelo simulador de arquiteturas compatíveis com o **CRAY-1** |2| e produziram resultados idênticos aos dos códigos objetos gerados pelo compilador Fortran CFT1.09 da Cray Research.

5. CONCLUSÕES

A disponibilidade de um montador facilitará o uso do simulador de arquiteturas compatíveis com o **CRAY-1** que vem sendo utilizado pelos alunos de mestrado dentro da disciplina de processamento paralelo na COPPE/Sistemas.

Como instrumento de apoio à pesquisa em processamento vetorial, o montador e simulador estão sendo utilizados na avaliação de um compilador vetorizado e no desenvolvimento de um compilador para a linguagem paralela ACTUS.

6. REFERÊNCIAS

|1| C.L. Amorim, "Simulação de uma Classe de Processadores Vetoriais". Anais do I Simpósio Brasileiro de Arquitetura de Computadores - Processamento Paralelo. Jul. 1987, Gramado, RS.

|2| C.L. Amorim, "The Vector Processor Simulator User's Guide", Relatório Técnico, Programa de Sistemas, COPPE/UFRJ, 1986.

|3| Cray Research, Inc., "**CRAY-1** Hardware Reference Manual", publ, nº 2240009, rev. F., 1979.

|4| K. Hwang e F.A. Briggs, "Computer Architecture and Parallel Processing". McGRAW HILL 1987.

|5| Tannenbaum, A.S., "Structered Computer Organization", Prentice - Hall 1984.