Relatório da MVN

Santiago Quintero Hincapié

11726111

PCS 3216

USP - POLI

Funcionalidade da aplicação

A ideia era que o usuário puder escrever um programa utilizando mnemônicos num endereço de memória e também uns dados se quiser, e ver o código de maquina (baixo nível) na memória. O usuário também poderia ver o processo de execução se quiser, avançando paço a paço pelas linhas de código ou simplesmente executando o programa e ver o resultado.

¿Como qual ferramenta foi feita a aplicação?

A aplicação esta feita num entorno visual de java (JFrame). Se utilizaram duas JTables para que o usuário puder escrever o programa e os dados, um JTextArea para mostrar o conteúdo da memória, quatro JTextFields para que o usuário puder escrever a entrada e o endereço inicial para executar o programa e finalmente quatro JButtons para carregar o programa e os dados na memória, para executar o programa e para reiniciar tudo, a funcionalidade que tem cada uma destas ferramentas na aplicação esta descrita no manual de usuário.

¿Como foi feita a aplicação?

Inicialmente se tinha que ler e montar na memória o escrito pelo usuário, para isso se fiz uma função que recorre às tabelas de programa e dados para logo montar todo na memória.

```
/** Método para carregar em memoria os dados e o programa (LOADER).
 */
private void montarEmMemoria() {...12 lines }
```

Esta função executa 3 funções diferentes: transformar os mnemônicos, montar as instruções e os dados.

```
/** Método para transformar o mnemônico em símbolo de máquina.
 * @param mnemonico String - mnemônico.
 * @return String - símbolo de máquina.
 */
private String transformarCO(String mnemonico) {...21 lines }
/** Método para escrever na memoria uma instrução.
 * @param ci String - contador de instruções.
 * @param co String - código de operação.
 * @param op String - operando.
 */
private void InstrucaoEmMemoria(String ci, String co, String op) {...13 lines }
/** Método para escrever na memoria um dado.
 * @param ci String - contador de instruções.
 * @param dado String - dado.
 */
private void DadoEmMemoria(String ci, String dado) {...7 lines }
```

Depois de ter os dados montados na memória o usuário pode executar o programa.

```
/** Método para executar o programa.
 */
private void run() {...14 lines }
```

Finalmente pode-se reiniciar as tabelas para escrever um novo programa.

```
/** Método para inicializar as tabelas de programa, dados e memoria.
 */
private void initComponentsAux() {...32 lines }
```

Desafios encontrados no processo

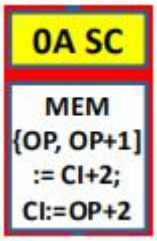
1. Entender a **funcionalidade** e a **lógica** de cada mnemônico, particularmente dos mnemônico **LV** e **SC**.

No caso de **load value**, a OP tem 12 bits e o AC tem 8 bits, então somente pego os dois últimos dígitos do OP para salva-los no AC.

```
LV    V      /3xxx     LOAD VALUE
```

Esta instrução tem uma restrição, o usuário só pode escrever números decimais positivos, mas tem uma alternativa, se quiser escrever um número decimal negativo pode ser feito na tabela de dados.

No caso de **subroutine call**, se armazena o endereço de retorno (CI + 2), em 12 bits, na posição de memória apontada pelo seu operando e seguinte, preenchendo com um zero o dígito mais significativo.

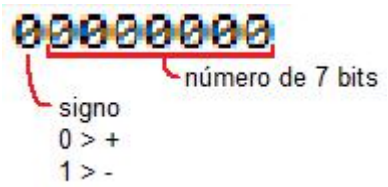


Acontece que zero é código de operação de uma operação de desvio incondicional, portanto foi armazenado aí uma instrução de retorno para a posição de memória que segue a instrução de subroutine call.

Em seguida, faz CI:=OP+2 ou seja, desvia para a posição de memória que segue este desvio incondicional colocado na entrada da subrotina. Feito este desvio, a subrotina pode ser executada.

Quando terminar tudo o que precisa ser feito, ela retorna ao programa que a chamou simplesmente fazendo-se um desvio para o desvio incondicional que foi colocado pela instrução subroutine call entrada da subrotina chamada, é importante aclarar que este desvio ao final da subrotina depende do programa, ou seja, depende do usuário utilizar corretamente a instrução **return from subroutine**.

2. Como salvar na memoria os **números enteros negativos**.



Esta é a estrutura que tem os números na memoria, então por isso os enteros devem estar entre [128, -128].