

Projeto de máquina virtual

Professor: João José Neto

Aluno:

Víctor de Sousa Lamarca - 9345922

Sumário

Introdução	3
Descrição do projeto	3
Componentes e funcionamento	3
Divisão do site	4
Especificação de projeto	4
Linguagem	4
Formato código objeto	5
Como rodar o programa	6
Comandos de interface	6
Inicialização	10
O programa N quadrado	10
Montador	12
5.1 Descrição do montador	12
5.2 Diagrama de funcionamento do montador	12
5.3 Testes do montador	15
Loader e Dumper	15
6.1 Loader	15
6.2 Dumper	17
Simulador de máquina virtual	19
Conclusão	20

1. Introdução

1.1 Descrição do projeto

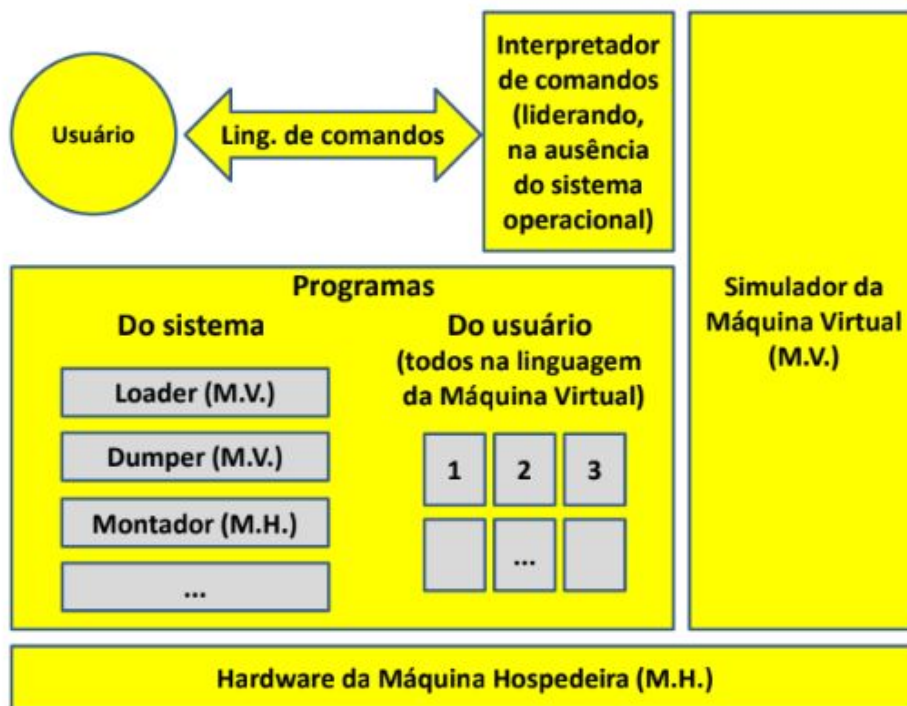
O presente relatório diz respeito ao projeto da disciplina Sistemas de Programação que visa planejar e implementar uma máquina virtual com um dado conjunto mínimo de instruções capazes de serem executados. A linguagem é do tipo assembly e o projeto aborda desde a sua montagem e carregamento até a sua execução propriamente dita.

O projeto foi desenvolvido na linguagem C++, utilizando-se do IDE Eclipse, mais precisamente no sistema operacional Ubuntu 16.04.

1.2 Componentes e funcionamento

Abaixo se encontra o esquema geral do funcionamento do projeto.

Panorama do Sistema de Programação



A interface entre o usuário e o resto do sistema é feita por um interpretador que faz o papel de um primitivo sistema operacional que aceita 5 comandos que serão explicados em mais detalhe nos próximos itens. Basicamente os comandos permitem a execução de um certo arquivo do usuário em linguagem simbólica sendo que primeiro este deve ser montado para o formato de código objeto, em seguida carregado na memória principal pelo arquivo de sistema Loader e, então,

deve ser executado pelo simulador da máquina virtual. Após a execução, o dumper pode ainda fazer a persistência de uma parte da memória principal pertinente a execução do programa executado.

Todos os comandos são, no fundo, convertido para comandos na máquina hospedeira que, neste projeto, é emulada pelo C++.

1.3 Divisão do site

O site do projeto descrito neste relatório é

<https://sites.google.com/site/2018pcs321609345922>

Nele é possível baixar as pastas referentes ao projeto. Baixando a pasta bin é possível rodar de fato o projeto a partir dos executáveis projeto_mv_linux ou projeto_mv_windows (de acordo com o sistema operacional usado). Já a pasta src permite acessar os códigos referentes ao projeto, que contém arquivos para os principais módulos do sistema.

2. Especificação de projeto

2.1 Linguagem

A linguagem utilizada é conforme a tabela a seguir:

Tabela de Instruções e Pseudo-instruções

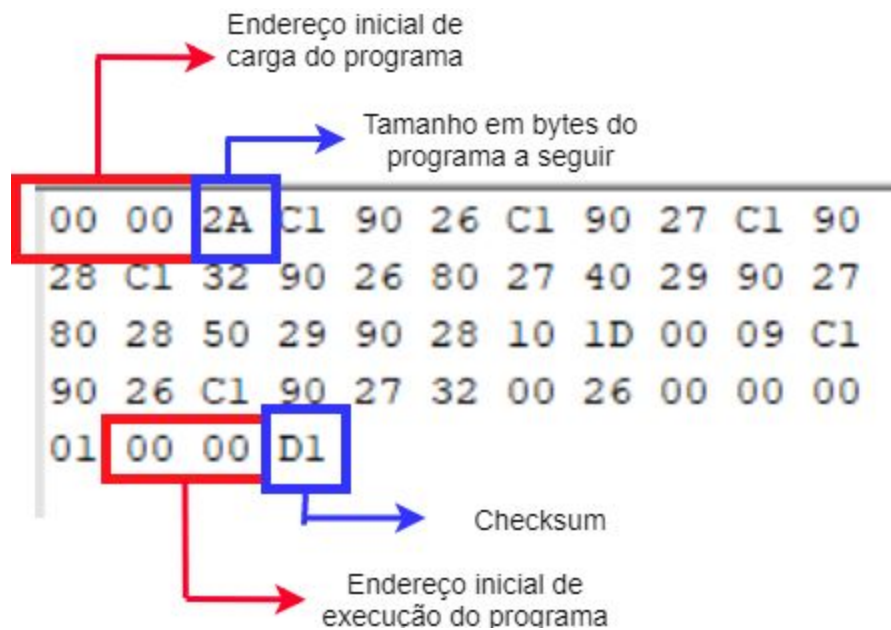
;	MNEMÔNICOS	CÓDIGO	INSTRUÇÃO DA MÁQUINA VIRTUAL
;			OBS.1: y= número do banco de memória (hexadecimal, 4 bits)
;			OBS.2: x= dígito (hexadecimal, 4 bits)
;	JP J	/0xxx	JUMP UNCONDITIONAL desvia para endereço /yxxx
;	JZ Z	/1xxx	JUMP IF ZERO idem, se acumulador contém zero
;	JN N	/2xxx	JUMP IF NEGATIVE idem, se acumulador negativo
;	CN C	/3x	* instruções de controle (/x = operação desejada)
;	+	/4xxx	ADD soma ac. + conteúdo do endereço /yxxx (8bits)
;	-	/5xxx	SUBTRACT idem, subtrai (operação em 8 bits)
;	*	/6xxx	MULTIPLY idem, multiplica (operação em 8 bits)
;	/	/7xxx	DIVIDE idem, divide (operação em 8 bits)
;	LD L	/8xxx	LOAD FROM MEMORY carrega ac. com dado de /yxxx
;	MM M	/9xxx	MOVE TO MEMORY move para /yxxx o conteúdo do ac.
;	SC S	/Axxx	SUBROUTINE CALL guarda end.de retorno e desvia
;	OS O	/Bx	* OPERATING SYSTEM CALL - 16 chamadas do sistema
;	IO I	/Cx	* INPUT/OUTPUT - entrada, saída, interrupção
;		/D	* combinação disponível
;		/E	* combinação disponível
;		/F	* combinação disponível
;			
;	MNEMÔNICOS	OPERANDO	PSEUDO-INSTRUÇÃO DO MONTADOR
;	@ @	/yxxx	ORIGIN define end. inicial do código a seguir
;	# #	/yxxx	END define final físico do prog. e end. de partida
;	\$ \$	/xxx	ARRAY define área de trabalho c/tamanho indicado
;	K K	/xx	CONSTANT preenche byte corrente c/ constante

As **instruções do tipo controle** podem ser seguidas dos operandos HM (halt machine) que encerra o funcionamento da máquina virtual retornando o controle ao interpretador de comandos (interface), ou então do tipo IN, que indica que a instrução seguinte a ser executada, se acessa, deve acessar a memória de forma indireta. Vale ressaltar que todo código deve conter o comando CN HM, do contrário nunca finalizará ao se rodar o programa.

Outro detalhe importante é quanto aos **números** lidos. Neste projeto os números precedidos de / tem seu valor interpretado em hexadecimal. Já os números sem nada são interpretados em decimal. As duas maneiras a seguir representam o número 16: '/10' e '16'.

2.2 Formato código objeto

Abaixo encontra-se o código objeto do programa loader, que será mais detalhadamente discutido em itens a seguir. Quanto ao código objeto em si, pode-se ver sua estruturação. Ele é dividido em pares de dígitos hexadecimal que representam um byte a ser carregado na memória.



Os dois primeiros bytes indicam o endereço inicial, o byte seguinte indica o tamanho, em bytes, do programa. Em seguida há o programa em si e, por fim, há mais dois bytes para o endereço inicial de execução do programa e mais

um byte para o checksum, que é tal que a soma de todos os bytes do programa objeto seja múltiplo de 2^8 (256).

2.3 Como rodar o programa

Para rodar o programa basta baixar a pasta bin presente no site deste projeto. Ela contém um arquivo binário referente a implementação do projeto e outros arquivos, tanto de sistema como de usuário.

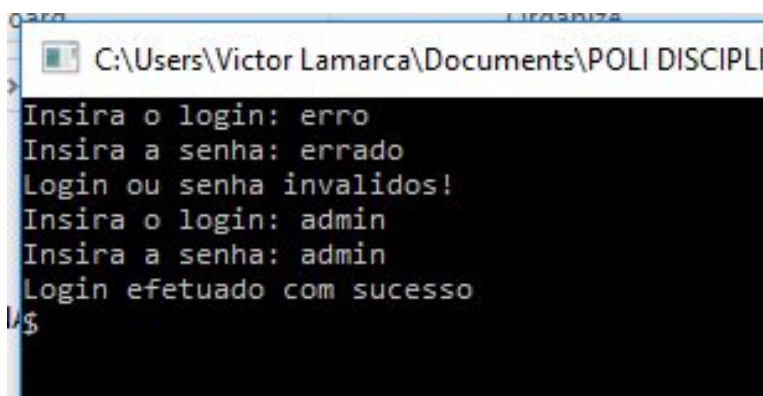
Tendo baixado a pasta, basta entrar nela em seu computador e clicar no arquivo executável de acordo com o sistema operacional em uso, projeto_mv_winows ou projeto_mv_linux. Ambos funcionam e são executáveis da mesma implementação.

Ressalta-se que os arquivos, de usuário e sistema, devem estar no mesmo diretório que o executável. Basta deixá-los na pasta bin por exemplo. Na pasta bin é possível acessar as implementações

Ao clicar no executável em questão deve abrir uma janela que pedirá o login, conforme melhor explicado no item a seguir.

2.4 Comandos de interface

Login e senha: A interface permite o contato entre o usuário e o resto do sistema. Primeiramente é necessário efetuar um login para se ter acesso ao sistema, conforme mostra a seguinte imagem:



```
C:\Users\Victor Lamarca\Documents\POLI DISCIPLI...
>
Insira o login: erro
Insira a senha: errado
Login ou senha invalidos!
Insira o login: admin
Insira a senha: admin
Login efetuado com sucesso
l/$
```

O único login e senhas aceitáveis são admin - admin, qualquer outra entrada acusará erro e requisitará login e senha novamente, conforme imagem acima.

DIR: O comando DIR mostra os arquivos presente no mesmo diretório do executável, indicando quais são do usuário e quais são do próprio sistema. Estes últimos não podem ser invocados diretamente pelo próprio usuário.

```
$DIR
arquivos do usuario:
  n_quadrado.dump
  n_quadrado.obj
  n_quadrado.txt
  teste.txt

arquivos do sistema:
  dumper.obj
  dumper.txt
  loader.obj
  loader.txt

$
```

RUN <nome_arquivo>: O comando RUN recebe como parâmetro o nome de um arquivo a ser executado. Após a execução do arquivo a memória principal deve ter sido alterada, o que pode ser verificado com o comando seguinte (MD - memory display). O comando RUN foi testado, com o seguinte programa teste.txt (além do programa n_quadrado que será abordado mais adiante).

```
$RUN teste.txt  
$_
```

```
1 ;teste.txt  
2  
3 @ /100  
4  
5 X K 5  
6 Y K 6  
7 ANS K 0  
8 INIC LD ANS  
9 LOOP LD X  
10 * Y  
11 MM ANS  
12 JN LOOP ;comentario  
13 CN HM  
14 # INIC  
15
```

MD <posicao_memoria>: O comando MD (memory display) recebe como parâmetro a posição de memória a partir da qual será mostrará os 10 bytes seguintes no terminal. (O comando pode ainda receber um terceiro parâmetro que indica quantos bytes seguintes deve mostrar se o usuário desejar outro valor que não 10).

O uso deste comando, após a execução do “RUN teste.txt”, permitiu verificar o funcionamento do programa rodado, de maneira que o byte rotulado por “ANS” contém o valor $5 \times 6 = 30$ (1E em hexadecimal). Note que o byte do ANS é o byte número 102 (2 bytes a frente de /100, o início do programa).

```
$RUN teste.txt  
$MD /100  
05 06 1E 81 02 81 00 61 01 91  
$
```


DEL <nome_arquivo>: O comando DEL recebe como parâmetro o nome de um arquivo a ser apagado,, sendo possível verificar seu funcionamento ao executar um comando DIR antes e outro depois de um comando DEL, o que evidenciará que o arquivo apagado de fato não está mais lá.

```
$DIR
arquivos do usuario:
  n_quadrado.dump
  n_quadrado.obj
  n_quadrado.txt
  teste.dump
  teste.txt

arquivos do sistema:
  dumper.obj
  dumper.txt
  loader.obj
  loader.txt

$DEL teste.dump
$DIR
arquivos do usuario:
  n_quadrado.dump
  n_quadrado.obj
  n_quadrado.txt
  teste.txt

arquivos do sistema:
  dumper.obj
  dumper.txt
  loader.obj
  loader.txt

$_
```

END: O comando END é bastante simples. Este conclui a simulação, encerrando-a. Assim, após sua execução o terminal fecha, como pode ser facilmente verificado baixando-se o programa.

3. Inicialização

A inicialização do sistema se dá da seguinte maneira. Primeiro são montados os arquivos de sistema (loader e dumper) que são, em seguida, carregados na memória principal pelo pré-loader (feito em linguagem de alto nível pela máquina hospedeira). Se os programas de sistema não forem achados (não estiverem no mesmo diretório do executável) o programa acusará o erro e encerrará.

Em seguida é chamado o interpretador de comandos (interface) que requisita, então, o login e senha para, por fim, permitir a execução dos comandos descritos acima.

Abaixo encontra-se o trecho do código que exemplifica essa sequência de eventos que dizem respeito a inicialização do sistema:

```
int main (int argc, char *argv[]){  
  
    montador("loader");  
    montador("dumper");  
    preLoader();  
  
    //chamda da interface  
    interpretadorDeComandos();  
}
```

4. O programa N quadrado

O programa N quadrado foi o algoritmo discutido em aula e usa o fato de que a soma dos n primeiros quadrados equivale ao n-ésimo quadrado perfeito.

A codificação foi apresentada em aula e, a seguir, encontra-se um código muito semelhante ao de aula usado neste projeto que serve para testes dos componentes que serão apresentados a seguir, o montador, loader, dumper e o próprio simulador de máquina virtual.

1		@ /110
2	INIC:	
3		LD UM
4		MM CONT
5		MM IMPAR
6		MM N2
7	LOOP:	
8		LD CONT
9		- N
10		JZ FORA
11		LD CONT
12		+ UM
13		MM CONT
14		LD IMPAR
15		+ DOIS
16		MM IMPAR
17		+ N2
18		MM N2
19		JP LOOP
20	FORA:	
21		CN HM
22		CN HM
23	UM	K 01
24	DOIS	K 02
25	IMPAR	K 0
26	N	K 5
27	N2	K 0
28	CONT	K 0
29		# INIC
30		

5. Montador

5.1 Descrição do montador

O montador do projeto é um montador de dois passos que, primeiramente identifica os rótulos de um arquivo programa, fazendo a correspondência em uma tabela e, em seguida, é feita a transformação dos rótulos no endereço de forma explícita, fazendo o código objeto. Além disso, o montador também toma conta das pseudoinstruções, tratando-as e as eliminando para o código objeto.

A tabela que faz a correspondência dos respectivos rótulos com seus endereços é, na prática, implementada com um mapa do c++ da seguinte maneira:

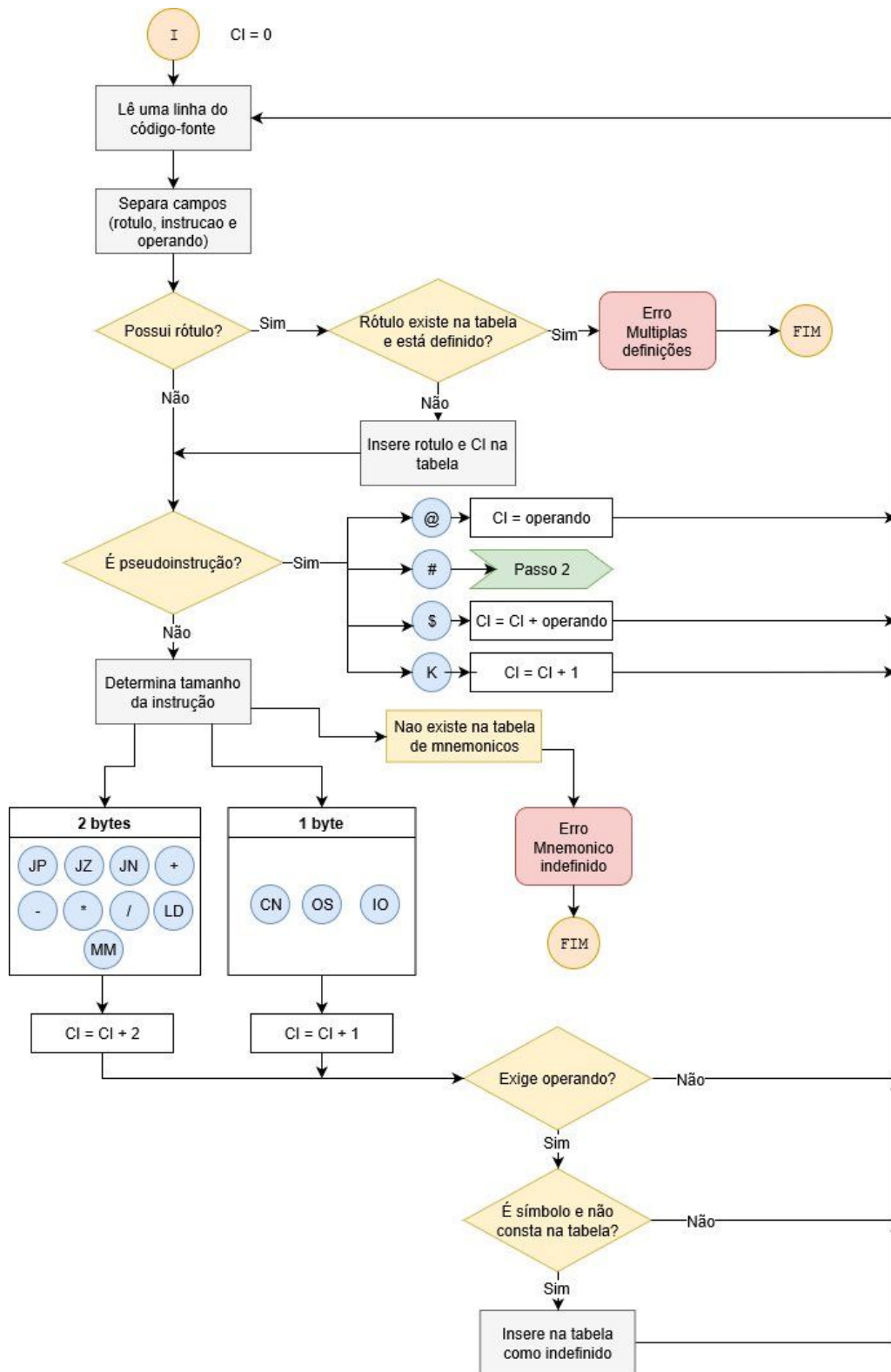
```
//tabela de rótulos para correspondência com endereços  
map<string,int> tabela;
```

Trata-se de um mapa de string (rótulos) para inteiro (que são posteriormente convertido em BYTES para representar o endereço).

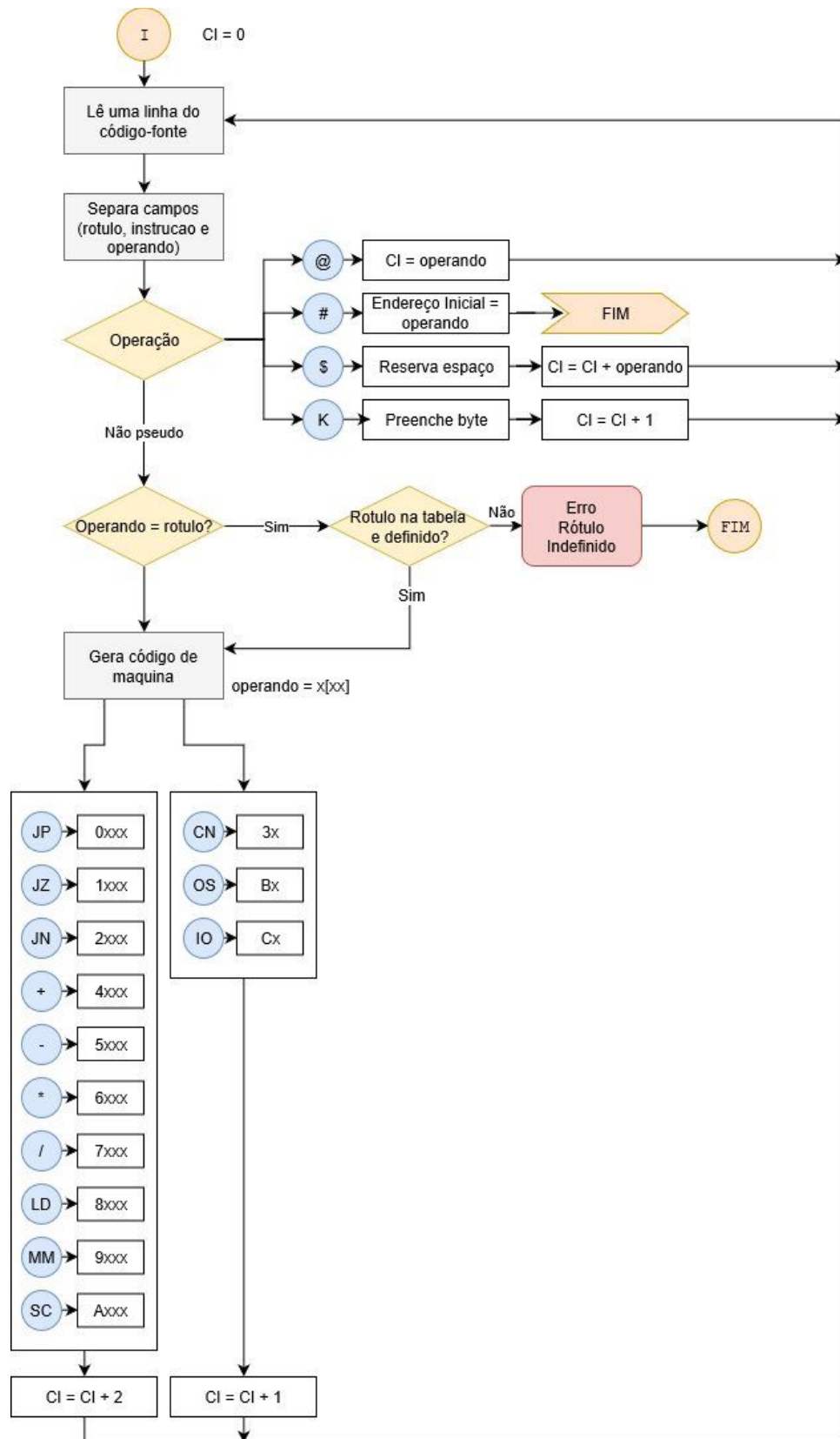
5.2 Diagrama de funcionamento do montador

Os diagramas de funcionamento do montador, indicando os passos 1 e 2, podem ser vistos a seguir:

Passo 1



Passo 2



5.3 Testes do montador

O montador gera código objeto a partir de códigos fonte. Para ilustrar seu correto funcionamento peguemos o programa de `n_quadrado`, já mostrado anteriormente, e mostremos o arquivo `n_quadrado.obj` correspondente gerado:

1	01 10 28 81 32 91 37 91 34 91 36 81 37
2	51 35 11 30 81 37 41 32 91 37 81 34 41
3	33 91 34 41 36 91 36 01 18 30 30 01 02
4	00 05 00 00 01 10 C9
5	

Pode-se verificar que o código objeto é muito semelhante ao código objeto do programa `n_quadrado` apresentado em sala na aula, com exceção de que este foi carregado a partir da posição /110 de memória para não conflitar com o carregamento do loader e dumper.

Sendo assim verifica-se que o montador está, de fato, correto.

6. Loader e Dumper

6.1 Loader

A função do programa loader é carregar os programas de usuário para permitir sua execução, sendo ele, por sua vez é carregado pelo programa pré-loader.

Ao se executar um comando de RUN o arquivo é primeiro montado e, em seguida carregado pelo loader na memória que, em seguida, chama a execução do programa carregado.

Abaixo há o código em assembly referente ao loader:

```

1 ;loader.txt
2 @ /000
3
4 INIT: IO /1 ; Get data
5 MM INITADD1 ; Salva primeiro byte do endereço inicial
6 IO /1 ; Get data
7 MM INITADD2 ; Salva segundo byte do endereço inicial
8 IO /1 ; Get data
9 MM TAMANHO ; Salva byte do tamanho
10
11 LOOP: IO /1
12 CN /2
13 MM INITADD1
14
15 LD INITADD2
16 + ONE
17 MM INITADD2
18
19 LD TAMANHO
20 - ONE
21 MM TAMANHO
22
23 JZ FIM
24 JP LOOP
25
26 FIM: IO /1 ; Get data
27 MM INITADD1 ; Salva primeiro byte do endereço inicial
28 IO /1 ; Get data
29 MM INITADD2 ; Salva segundo byte do endereço inicial
30 CN /2
31 JP INITADD1 ; Passa execução para o programa carregado
32
33 INITADD1 K 0
34 INITADD2 K 0
35 TAMANHO K 0
36 ONE K 1
37
38 # INIT
39

```

Um comando importante de ser discutido neste programa é o IO que vai lendo os bytes de um device definido . Neste caso, na verdade, este device é o código

objeto especificado pelo comando RUN. IO /1 significa get data GD e, assim, lê o cabeçalho referente ao código objeto para conseguir carregá-lo corretamente.

Após o carregamento chega-se ao fim do LOADER, que lê os últimos bytes do código objeto para saber onde deve ser começada a execução do programa que acabou de ser carregado e, então, o programa pula para essa posição, para executar o programa. Estes comandos finais vêm logo após o rótulo FIM, na linha 26.

Após a inicialização do sistema sabe-se que o programa loader já está carregado na memória a partir da posição /000. Assim, uma maneira de verificar isso é a partir do comando MD explicado acima. Assim, o carregamento pode ser visualizado da seguinte maneira:

```
$  
$MD 0 50  
C1 90 26 C1 90 27 C1 90 28 C1 32 90 26 80 27 40 29 90  
27 80 28 50 29 90 28 10 1D 00 09 C1 90 26 C1 90 27 3  
2 00 26 00 00 00 01 00 00 00 00 00 00 00  
$
```

Já o funcionamento do programa loader propriamente dito pode ser verificado checando-se o endereço do próprio programa carregado. Usando como exemplo o programa n_quadrado, verifica-se o seu carregamento em sua posição indicada da seguinte maneira:

```
$RUN n_quadrado.txt  
$MD /110 /28  
81 32 91 37 91 34 91 36 81 37 51 35 11 30 81 37 41 32  
91 37 81 34 41 33 91 34 41 36 91 36 01 18 30 30 01 0  
2 09 05 19 05  
$
```

Verifica-se que os valores na memória são praticamente os mesmos do no código objeto mostrados acima, com exceção de alguns valores no final que já estão com os valores após a execução (já contém o valor de 5 ao quadrado = 25 = 19 em hexadecimal).

6.2 Dumper

O programa dumper é também carregado na memória pelo pré-loader na inicialização do sistema. Este é, mais precisamente carregado a partir da posição /50 (50 em hexadecimal) de memória.

O objetivo do programa dumper é fazer a persistência de uma parte da memória principal em arquivo (mais especificamente nesse caso em um arquivo .dump).

A seguir pode-se verificar o código do programa dumper:

```
1 ;dumper.txt
2 @ 50
3
4 INIT: LD INITADD1
5       IO /5
6       LD INITADD2
7       IO /5
8       LD TAMANHO
9       MM TAMORIG
10      IO /5
11
12 LOOP: CN /2
13       LD INITADD1
14       IO /5           ;Put data - device = 1
15
16       + CHECKSUM
17       MM CHECKSUM
18
19       LD INITADD2
20       + ONE
21       MM INITADD2
22
23       LD TAMANHO
24       - ONE
25       MM TAMANHO
```

```

26
27      JZ FIM
28      JP LOOP
29
30 FIM:   LD INITADD1
31        IO /5
32        LD INITADD2
33        - TAMORIG
34        IO /5
35        LD CHECKSUM
36        IO /5
37        CN HM
38
39 INITADD1 K 1      ;Endereço inicial a dumpar
40 INITADD2 K /10
41 TAMANHO  K /28   ;Numero de bytes a dumpar
42 TAMORIG  K 0
43 OVERFLOW K 0
44 CHECKSUM K 0
45 ONE      K 1
46
47      # INIT

```

Nota-se o uso do comando IO /5 (put data - PD) no dumper, que diz respeito a escrita do código objeto no arquivo correto. O dumper também leva cuidados para gerar o código objeto nos padrões propostos por esse projeto e já explicados acima.

Um detalhe importante do funcionamento do loader é que este deve receber os parâmetros para indicar o intervalo de memória do qual irá fazer a persistência.

Estes valores devem ser avisados no próprio código do dumper, nas variáveis indicadas por INITADD1 e INITADD2 (endereço de memória inicial) e TAMANHO (que indica quantos bytes serão persistidos). Os valores que estão em específico são justamente os correspondentes ao programa n_quadrado, que a seguir ajudarão na validação do funcionamento do simulador de máquina virtual.

7. Simulador de máquina virtual

O simulador de máquina virtual é o responsável pelo efetivo funcionamento das instruções propostas na tabela de comandos apresentadas no início deste documento. Ele converte cada um dos códigos referentes a instruções em um funcionamento da máquina hospedeira (feita em C++) de maneira a permitir o funcionamento de qualquer programa escrito na linguagem simbólica.

Ao rodar-se o comando “RUN n_quadrado.txt” é gerado o código objeto n_quadrado.obj e o código persistido após a execução do programa. Assim é possível ver o valor de N especificado (5 no caso) e o campo do byte que deve conter a resposta (N²) antes e depois da execução, que contém o valor correto (19 em hexadecimal que corresponde, de fato, a 5 ao quadrado, 25).

n_quadrado.obj										
1	01	10	28	81	32	91	37	91	34	91
2	36	81	37	51	35	11	30	81	37	41
3	32	91	37	81	34	41	33	91	34	41
4	36	91	36	01	18	30	30	01	02	00
5	05	00	00	01	10	C9				
6										

n_quadrado.dump										
1	01	10	28	81	32	91	37	91	34	91
2	36	81	37	51	35	11	30	81	37	41
3	32	91	37	81	34	41	33	91	34	41
4	36	91	36	01	18	30	30	01	02	09
5	05	19	05	01	10	4D				
6										
7										

Assim é possível atestar o funcionamento do DUMPER bem como do simulador de eventos por ter conseguido executar o programa n_quadrado. O primeiro byte da quinta linha indica o número n a ser elevado ao quadrado (5 no caso). E o segundo byte dessa mesma linha, como se pode ver, no arquivo .dump, contém o valor de n ao quadrado (19 em hexadecimal que é 25) e, assim, é possível atestar o funcionamento de ambos componentes mencionados.

8. Conclusão

Em conclusão é possível dizer que o presente projeto apresenta uma versão funcional de um sistema capaz de rodar a linguagem simbólica proposta e fazendo uso dos componentes típicos de um sistema de programação de baixo

nível, permitindo seu melhor entendimento, envolvendo este montador, loader, dumper, simulador e interface.