Sistemas de programação - PCS3216

**Projeto de máquina virtual**

**Professor**: João José Neto

**Aluno:**

Víctor de Sousa Lamarca - 9345922

**Sumário**

[**Introdução**](#_hbhokkqq1j) **3**

[Descrição do projeto](#_mduajb7qp05) 3

[Componentes e funcionamento](#_97izfa85k9u4) 3

[Divisão do site](#_hrimcb23ptgw) 4

[**Especificação de projeto**](#_gg0pb5dcoc22) **4**

[Linguagem](#_pxljmynbvzd3) 4

[Formato código objeto](#_tiijohesoo9h) 5

[Como rodar o programa](#_fedufjlxxol6) 6

[Comandos de interface](#_fedufjlxxol6) 6

[**Inicialização**](#_9n37ikrroh9w) **10**

[**O programa N quadrado**](#_bu7024juw3bn) **10**

[**Montador**](#_tpzfzwbmsq0r) **12**

[5.1 Descrição do montador](#_2vex4vkecv04) 12

[5.2 Diagrama de funcionamento do montador](#_otvcfcvrm0ur) 12

[5.3 Testes do montador](#_yuzkx5slpgd) 15

[**Loader e Dumper**](#_folaizvq7i6w) **15**

[6.1 Loader](#_392jmeuvmie) 15

[6.2 Dumper](#_kwc16ksrvjez) 17

[**Simulador de máquina virtual**](#_u7sm57uoqxpn) **19**

[**Conclusão**](#_hnvpyum2w5x3) **20**

# Introdução

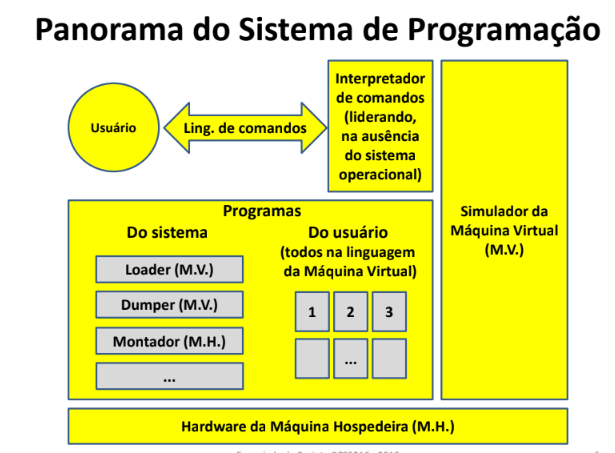
## Descrição do projeto

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 a 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.

## Componentes e funcionamento

Abaixo se encontra o esquema geral do funcionamento do projeto.



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++.

## Divisão do site

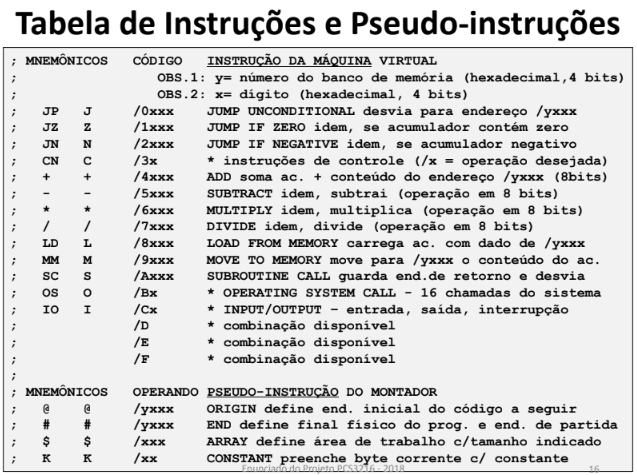
O site do projeto descrito neste relatório é <https://sites.google.com/site/2018pcs321609345922>

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.

# Especificação de projeto

## Linguagem

A linguagem utilizada é conforme a tabela a seguir:

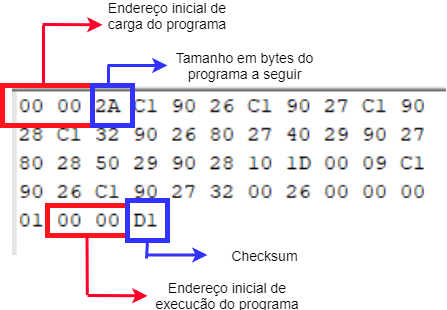


As **instruções do tipo controle** podem ser seguidas dos operandos HM (halt machine) que encerra o funcionamento da máquina virtual retornando o controle ao interpretador de comandos (interface), ou então do tipo IN, que indica que a instrução seguinte a ser executada, se acessa, deve acessar a memória 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’.

## 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 dividio 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).

## 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.

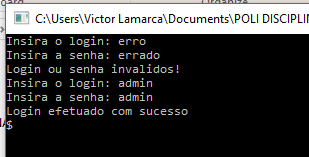
Tendo baixado a pasta, basta entrar nela em seu computador e clicar no arquivo executável de acordo com o sistema operacional em uso, projeto\_mv\_winows ou projeto\_mv\_linux. Ambos funcionam e são executáveis da mesma implementação.

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.

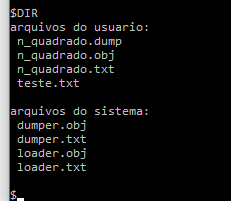
## Comandos de interface

**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:

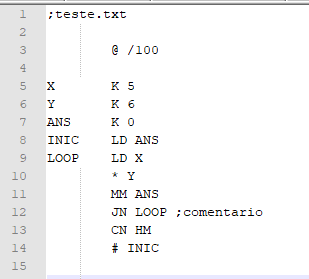


O único login e senhas aceitáveis são admin - admin, qualquer outra entrada acusará erro e requisitará login e senha novamente, conforme imagem acima.

**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.

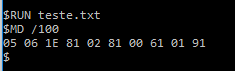


**RUN <nome\_arquivo>**: O comando RUN recebe como parâmetro o nome de um arquivo a ser executado. Após a execução do arquivo a memória principal deve ter sido alterada, o que pode será verificado co o comando seguinte (MD - memory display). O comando RUN foi testado, com o seguinte programa teste.txt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

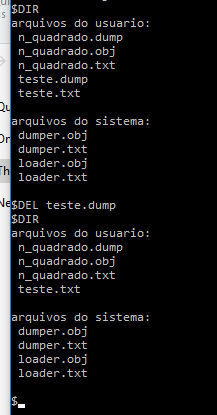


**MD <posicao\_memoria>**: O comando MD (memory display) recebe como parâmetro a posição de memória a partir da qual será mostrará os 10 bytes seguintes no terminal. (O comando pode ainda receber um terceiro parâmetro que indica quantos bytes seguintes deve mostrar se o usuário desejar outro valor que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



**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.

****

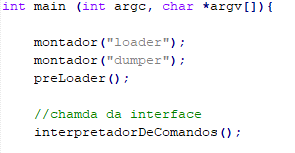
**END**: O comando END é bastante simples. Este conclui a simulação, encerrando-á. Assim, após sua execução o terminal fecha, como pode ser facilmente verificado baixando-se o programa.

# 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.

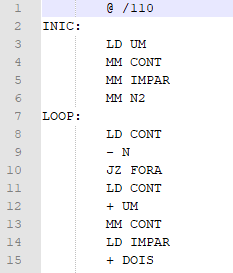
Abaixo encontra-se o trecho do código que exemplifica essa sequência de eventos que dizem respeito a inicialização do sistema:

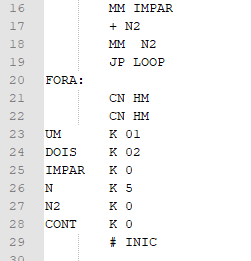
****

# 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.

****

****

# 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:

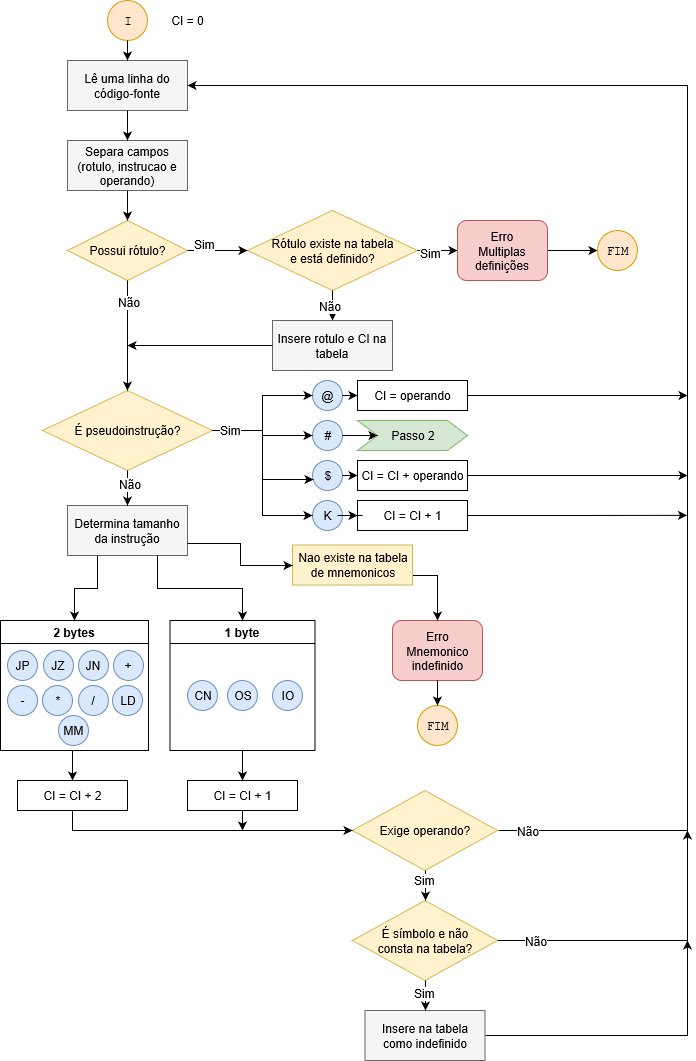
****

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

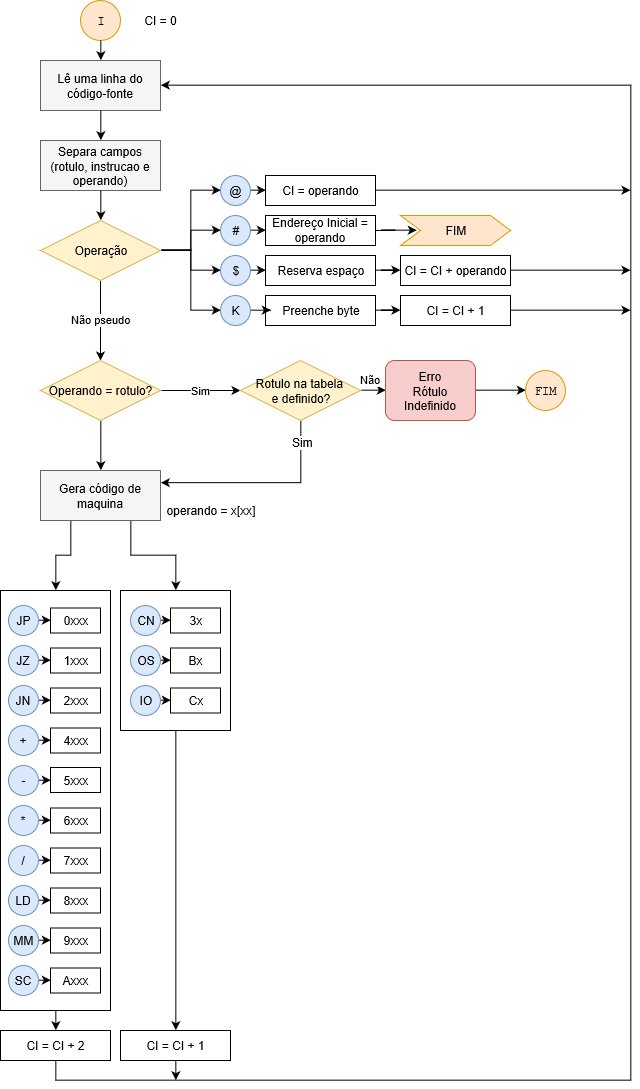
## 5.2 Diagrama de funcionamento do montador

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

Passo 1

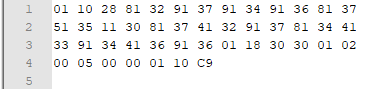


Passo 2



## 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:

****

Pode-se verificar que o código objeto é muito semelhante ao código objeto do porograma 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.

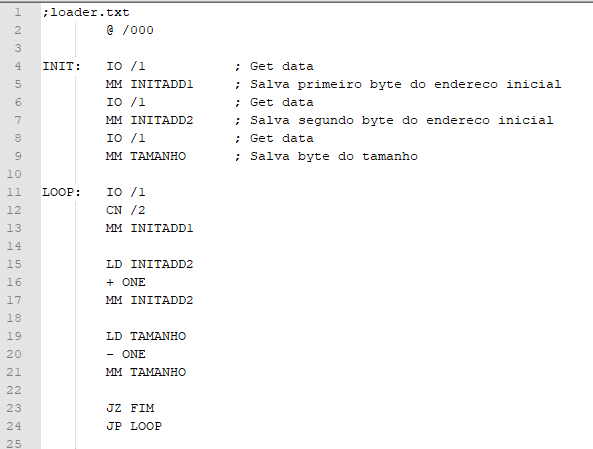
# Loader e Dumper

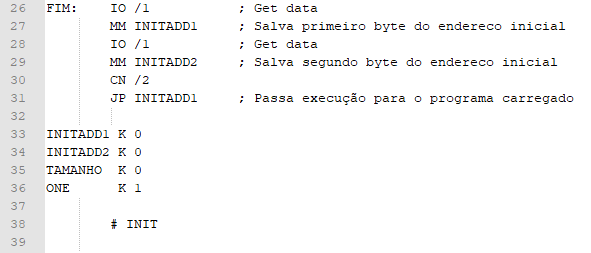
## 6.1 Loader

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:

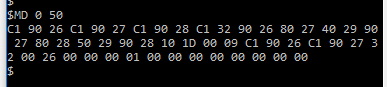
****

****

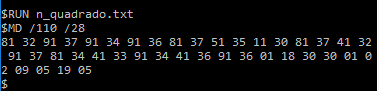
Um comando importante de ser discutido neste programa é o IO que vai lendo os bytes de um device definido . Neste caso, na verdade, este device é o código objeto especificado pelo comando RUN. IO /1 significa get data GD e, assim, lê o cabeçalho referente ao código objeto para conseguir carregá-lo corretamente.

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 á a partir do comando MD explicado acima. Assim, o carregamento pode ser visualizado da seguinte maneira:

****

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:



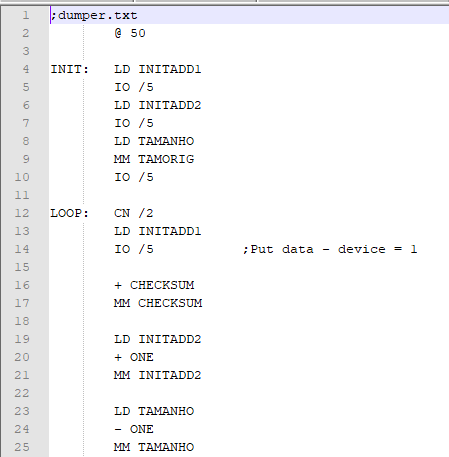
Verifica-se que os valores na memória são praticamente os mesmos do no código objeto mostrados acima, com exceção de alguns valores no final que já estão com os valores após a execução (já contém o valor de 5 ao quadrado = 25 = 19 em hexadecimal).

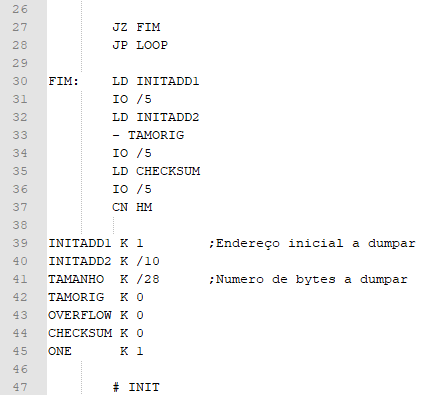
## 6.2 Dumper

O programa dumper é também carregado na memória pelo pré-loader na inicialização do sistema. Este é, mais precisamanente 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:

****



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.

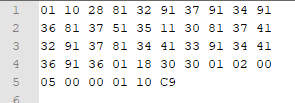
Estes valores devem ser avisados no próprio código do dumper, nas variáveis indicadas por INITADD1 e INITADD2 (endereço de memória inicial) e TAMANHO (que indica quantos bytes serão persistidos). Os valores que estão em específico são justamente os correspondentes ao programa n\_quadrado, que a seguir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

# Simulador de máquina virtual

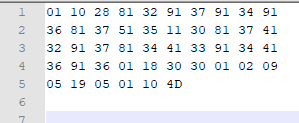
O simulador de máquina virtual é o responsável pelo efetivo funcionamento das instruções propostas na tabela de comandos apresentadas no início deste documento. Ele converte cada um dos códigos referentes a instruções em um funcionamento da máquina hospedeira (feita em C++) de maneira a permitir o funcionamento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



n\_quadrado.dump



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.

# 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.