

Fascículo Arq1 Lab

Utilização de emulador x86 (Jasmin)

1 O emulador x86, 32 bits, Jasmin

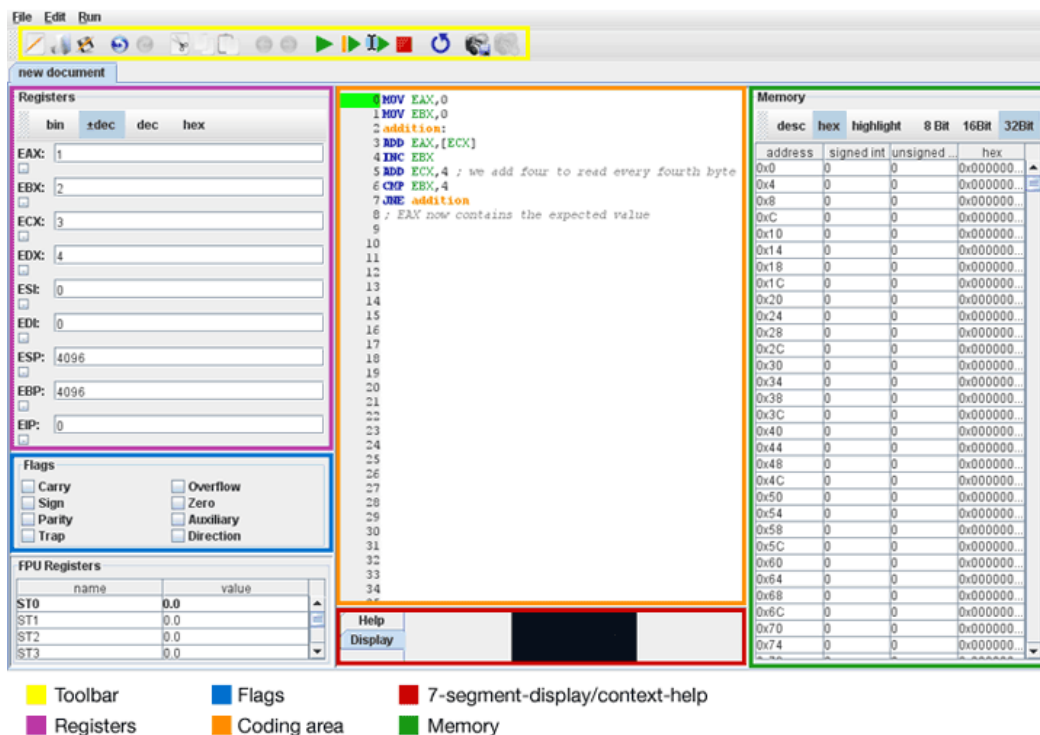
Para simplificar a produção e o teste de programas em assembly, vamos recorrer a um emulador chamado Jasmin. O emulador usa a sintaxe Intel, e as instruções funcionam do mesmo modo. Assim, é possível trabalhar num ambiente em que as instruções assembly podem ser facilmente testadas, e onde se pode rapidamente visualizar o seu efeito nos registos do processador, na memória, e nas *flags*.

1.1. Instalação







O emulador é implementado em java, por isso consegue correr em qualquer sistema que tenha uma JVM instalada (Linux, Windows, Mac, ...).

1. Descarregue o emulador disponível na página da disciplina no moodle, e guarde em qualquer pasta.
2. Dê permissões de leitura e execução ao ficheiro, por ex.:
`chmod 500 Jasmin.jar`
3. Execute o ficheiro usando o comando “java -jar Jasmin.jar”, ou através do ambiente gráfico fazendo “Open with > Java runtime”.

1.2. Interface



Toolbar:

-  **Run:** Executa todas as instruções a partir do atual EIP (*instruction pointer*, sinalizado na janela dos registos), até o programa acabar.
-  **Stop:** Força que o programa termine.
-  **Step:** Executa uma linha de código (atual EIP), e passa para a instrução seguinte.
-  Executa a instrução onde atualmente se encontra o cursor de escrita.
-  Faz **reset** à memória, a todos os registos, e à execução do programa.
-  Clicando na barra esquerda da janela de código (nos números de linha), permite definir **breakpoints**. O programa quando for executado vai pausar forçosamente nas instruções que tiverem um *breakpoint* definido.

Salvar ficheiros:

Os ficheiros de código devem ser guardados frequentemente, para não se perder o trabalho em casa de falha do Jasmin. Existem botões para salvar os documentos na toolbar. Adicionalmente, são também feitas cópias automáticas que podem ser recuperadas em caso de falha. Estas cópias encontram-se dentro da pasta Jasmin, na área do utilizador.

Coding area:

Onde se escreve o programa. A sintaxe e as operações funcionam do mesmo modo que funcionariam num programa real escrito em assembly com a sintaxe Intel.

Nota importante: as diretivas NASM que definem as secções do código (*section .data*, *section .text*, ...) e a diretiva "global main" **NÃO** devem ser escritas no emulador. Adicionalmente a instrução **ret** que termina o programa também não deve ser escrita.

Registers:

Mostra o conteúdo atual dos registos do processador. Os valores podem ser exibidos em diferentes bases de numeração, que podem ser escolhidas no topo da janela.

Memory:

Mostra o conteúdo atual da memória. É possível escolher a base de numeração na qual os valores são exibidos, tal como o número de bits (8, 16 ou 32) que são utilizados para visualizar cada bloco de memória (linha na tabela).

Flags:

Mostra as *flags* que estão atualmente ativas.

Console / Context-help:

Mostra a documentação da instrução que for selecionada na janela de código.

A consola mostra o output textual que possa eventualmente ser produzido pelo programa.

2 Exercícios fundamentais

1. Inicie o emulador *Jasmin* e crie o seguinte programa.

```
SECTION .data
asc: dd 35          ;variável asc
tp:  dd -35         ;variável tp

SECTION .text
global main

main:
    mov eax,[asc]
    mov ebx,eax
    mov eax,[tp]
    nop
    mov edx,65535
    xchg edx,eax
    xchg ebx,eax
    ret
```

a) Que modificações tiveram que ser feitas ao código de modo a ser aceite pelo emulador?

b) Coloque dois pontos de paragem (*breakpoints*): um na linha a seguir ao rótulo “main:”, e outro na instrução `nop`.

c) De seguida clique em “**Run**” e constate que a execução do programa teve início e parou no primeiro ponto de paragem (sinal verde). Indique abaixo o conteúdo do registo EIP:

d) Active a opção para ver as células de memória agrupadas em 32 bits. Verifique que as variáveis `asc` e `tp` foram inicializadas na memória. Indique o endereço de memória correspondente a cada uma destas variáveis, e o valor em hexadecimal do conteúdo da memória nesses endereços.

e) Active agora a opção para ver as células de memória individualmente, isto é, em 8 bits. Quantas células de memória estão diferentes de zero no espaço alocado à variável `asc` e à variável `tp`? Como explica a diferença entre as duas variáveis?

f) Mantendo ainda a visualização em 8 bits. Em que endereços é que se encontra a parte menos significativa de cada número?

- g) Uma vez que os números podem ter mais que 1 byte, é necessário seguir uma convenção que indique em que endereços de memória devem ficar os bytes mais e menos significativos. Numa arquitetura **Little Endian** os bytes menos significativos de um número ficam nos endereços de memória mais pequenos. Numa arquitetura **Big Endian** os bytes menos significativos ficam nos endereços maiores. Qual a arquitetura que está a ser usada? Justifique.

- h) Clique em “**Run**” novamente para continuar a execução até ao próximo ponto de paragem e verifique que o conteúdo dos registos corresponde ao esperado. Indique abaixo o conteúdo do registo EIP e dos registos de uso geral utilizados neste programa:

- i) Compare o valor do EIP na alínea h) e na alínea c). O que é o registo EIP? Qual a sua função na execução do programa?

- j) Finalmente avance com o “**Step**” até chegar ao final do programa. Em cada passo, registre as alterações que verificou no conteúdo dos registos.

2. Considere agora o seguinte excerto de um programa:

```
; SECTION .data
; declara um array de caracteres e coloca o NUL character (0) no final.
str: db 'hello!',0
var_a: db '1'      ; declara um caracter
var_b: db 1        ; declara um número um número em 8 bits
```

Apague o programa anterior, clique em “**Reset**”, e crie um novo programa com este excerto de código. Finalmente, clique em “**Run**” para executar o programa. Analisando a memória, responda às seguintes questões:

- a) Identifique o endereço de memória atribuído a cada variável, e quantos bytes foram reservados para cada uma delas:

b) No caso da variável `str`, o que é representado por cada byte?

c) Usando uma tabela ASCII (ex: <http://www.asciitable.com>), identifique a que corresponde cada um dos bytes inicializados com a variável `str`:

d) Registe agora o conteúdo dos bytes de memória da variável `var_a` e da variável `var_b`:

e) Como explica a diferença entre os dois?

3 Material de Apoio

Intel Assembler 80x86 CodeTable (referência rápida): <http://www.jegerlehner.ch/intel/IntelCodeTable.pdf>