Migrando para o NASM

Este documento é GPL e é atualmente mantido por Brivaldo Junior
O NASM é um Assembler LGPL para a Arquitetura x86 e compatíveis
criado por *Julian* e *Simon*

Sumário

1 - Compilando com o NASM

nasm -f <formato> <nome_do_arquivo> [-o <saída>]

Por exemplo, suponhamos que nós temos um arquivo chamado Exemplo.asm e queremos que ele seja compilado com o NASM utilizando o formato ELF (padrão do Linux), o comando seria:

nasm -f elf Exemplo.asm

o objeto de saída seria Exemplo.o, é neste caso que vale o -o, podemos definir o nome do arquivo de saída, por exemplo:

nasm -f elf Exemplo.asm -o Exemplo.com

Caso você queira que o NASM retorne em um arquivo os códigos hexadecimais gerados, seria utilizando então o '-l', por exemplo:

nasm -f coff Exemplo.asm -l Exemplo.lst

Temos também a possibilidade de gerar dentro de nossos códigos "pistas" para que possamos debbugar um código após estar compilado, assim utilizamos o '-g' para tal propósito, veja o exemplo:

nasm -f elf Exemplo.asm -o Exemplo.com -g

Pronto, o código executável gerado possui nossas pistas para DEBUG.
Uma utilidade muito legal é podermos, quando criamos nossos códigos cheios de erros, de fazer com que o NASM envie a saída de erros do console para um arquivo específico, é lógico que poderíamos utilizar shell script para fazer esta façanha, mas se o NASM já tem a funcionalidade para que complicar? ;p, veja o exemplo abaixo em que os erros do programa Exemp01.asm são enviados para o arquivo QuantosErros.err:

nasm -E QuantosErros.err -f obj Exemp01.asm

Se ao invés de '-E' ussasemos '-s' teríamos a saída na STDOUT, a saída padrão.

Como em C/C++ e o nosso amigão GCC, temos a possibilidade de que o NASM nos mostre Warnings, ou seja, códigos ou variáveis que talvez tenham sido utilizadas de forma equivocada (no caso do gcc o parametro é -Wall), para isso utilizamos o '-w' para que esses "Warning's" sejam gerados.

Obs: Antes de continuarmos vale lembrar que para o NASM, as variáveis serem maiúsculas ou minúsculas faz diferença, ou seja, ele é CASE-SENSITIVE.

1.1 - Acessando Conteúdo de Vetores no NASM

Vamos supor um vetor como o descrito abaixo:

```
vetor   db              "01234"
```

Para manipular o conteúdo de "vetor" no TASM faríamos o seguinte:

```
mov     ax, offset vetor
```

Neste caso ao "caminharmos" por [ax + si] estaríamos manipulando o conteúdo do vetor. Para realizar a mesma operação no NASM seria muito mais simples, se quisermos manipular o endereço do ponteiro "vetor" faríamos o seguinte:

```
mov     ax, vetor
```

Agora, para manipular o conteúdo de "vetor" faríamos assim:

```
mov     ax, [vetor]
```

Isso simplesmente quer dizer, que no NASM não precisamos utilizar a palavra OFFSET para referenciar um ponteiro. Mas devo avisar que o NASM busca e prima sempre pela simplicidade, mesmo com os métodos de compatibilidade com o TASM e o MASM, o NASM só pode trabalhar com um método de codificação, ou seja, não suporta códigos híbridos.

2 - A Linguagem do NASM

O layout de uma linha de código em NASM:

```
rotulo:  operadores_de_instrucao                ;comentário
```

Este é um modo bem comum para quem esta acostumado a programar com outros assembler's comerciais ou não e o NASM segue este mesmo modelo. Uma coisa interessante é que se uma linha de código for maior que uma linha é possível utilizar a barra invertida '\', isso fará com que o NASM aceite a próxima linha como continuação da primeira. veja o exemplo:

```
mov    ax, [bx + si\
]
```

isso é igual a:

```
mov    ax, [bx +si]
```

Ambas as formas são válidas.

Os caracteres, letras, números e também, `_', `$', `#', `@',`~', `.', e `?'. Um fato importante é que somente letras podem iniciar uma varíavel, o `.' é um caso especial e será analizado mais tarde. Quando temos um `$' prefixado a um registrador, isso nos da a entender que o que queremos ler é um registrador e não uma palavra reservada. Isso é feito para distinguirmos palavras reservadas de registradores, por exemplo, se criamos uma variável com o nome "eax" o NASM poderia então confundir com o registrador, então com o $eax o NASM passa a ter certeza de que você neste momento esta trabalhando com o registrador e não com a variável "eax".

2.1 - Endereços Efetivos

Um endereço efetivo é uma referência a memória. No NASM temos uma sintaxe muito simples, veja:

```
wordvar       dw      123
              mov     ax,[wordvar]
              mov     ax,[wordvar+1]
              mov     ax,[es:wordvar+bx]
```

Qualquer coisa diferente do que esta escrito acima é considerada uma referência inválida de memória.

Endereços efetivos de memória podem envolver mais de um registrador e seguem mais ou menos este caminho:

```
              mov     eax,[ebx*2+ecx+offset]
              mov     ax,[bp+di+8]
```

No NASM também é possível realizar operações algébricas com os endereços efetivos, isso pode não parecer muito _CORRETO_ mas esta perfeitamente certo, veja

abaixo:

```
mov     eax,[ebx*5]           ; assemblado em [ebx*4+ebx]
mov     eax,[label1*2-label2]   ; ie [label1+(label1-label2)]
```

Em muitos casos é possível conseguir um endereço real, de diversas formas. No nosso caso, o NASM gera a menor forma possível para encontrar o endereço real a partir do segmento:deslocamento.

2.2 - Constantes no NASM

O NASM suporta quatro tipos diferentes de constantes: numeric, character, string and floating-point. (numérica, caracter, cadeia de caracteres e ponto-flutuante).

Constantes Numétricas:

Uma constante numérica é simplesmente um número. No NASM podemos utilizar e especificar diferentes tipos de bases numéricas utilizando os seguintes sufixos: `H', `Q' ou `O', e `B' para hexadecimal, octal e binário, ou você ainda pode utilizar a sintaxe de C para os números em Hexadecimal, no caso, `0x'. Veja abaixo alguns exemplos:

```
mov     ax,100          ; decimal
mov     ax,0a2h         ; hex
mov     ax,$0a2         ; hex denovo: o 0 é necessário
mov     ax,0xa2         ; hex mais uma vez
mov     ax,777q         ; octal
mov     ax,777o         ; octal denovo
mov     ax,10010011b    ; binario
```

Constantes de Caracteres:

Constantes de caracteres nada mais são que quatro caracteres definidos entre aspas simples ou duplas, o que na verdade não faz muita diferença. Se por algum motivo armazenassemos mais de um caracter em um registrador ele seria `rearranjado' no formato litte-endian (o menor na maior posição de memória e o maior na menor posição de memória), se você escrevesse o seguinte código:

```
mov eax,'abcd'
```

A constante gerada não seria `0x61626364', mas `0x64636261'.

Constantes de STRINGS:

Constantes de strings podem ser declaras de diversas formas, mas geralmente são

do tipo `DB' ou `INCBIN'. Veja como podemos representar diversos tipos de constantes de strings:

```
db    'hello'          ; constante de string
db    'h','e','l','l','o'   ; constante equivalente em caracteres
```

E as que se seguem são todas equivalentes:

```
dd    'ninechars'            ; constante de string doubleword
dd    'nine','char','s'      ; esta formada por três doublewords
db    'ninechars',0,0,0      ; e o que realmente acontece...
```

Constantes de Ponto-Flutuante:

CONTINUA (Linha 1094,8 12%)