Unidade: Interpretação, Compilação e Tradução de Programas

Unidade: Interpretação, Compilação e Tradução de Programas

1) Interpretação e compilação.

Os programas desenvolvidos por um programador irá ser interpretado, compilado ou compilado e interpretado, o método para se chegar a execução do programa dependerá da linguagem que está sendo usada. Cada linguagem define se irá compilar o código para depois executar ou se irá apenas interpretar o código desenvolvido.

A compilação consiste em avaliar o código primeiro antes de executá-lo, esta análise verifica se há erros de sintaxe e semântica, caso tenha algo errado é gerado uma mensagem indicando o tipo de erro e, muitas vezes, o local. Se o código estiver escrito corretamente então, geralmente, é gerado outro arquivo com conteúdo executável, ou seja, o código será executado a partir deste arquivo gerado na compilação, qualquer alteração feita no arquivo origem deverá ser recompilado para refletir no arquivo executável. Portanto, não será executado o arquivo fonte e sim o arquivo gerado pela compilação.

Há duas etapas na compilação de programas

1) Geração de um programa equivalente a linguagem original
2) Execução do programa gerado.

Estas etapas não ocorrem paralelamente, e sim isoladamente conforme a seqüência descrita acima.

Em uma linguagem interpretada o mecanismo é diferente, ou seja, o código será executado e se houver algum erro irá ser exibida a mensagem somente quando for executada a linha onde está o erro.

2) Tradução de programas.

Para analisarmos um código em linguagem de montagem, vamos usar um emulador para criar e executar o código desenvolvido. O emulador “emu886” pode ser baixado no link http://www.emu8086.com/ . O emu8086 é um emulador do microprocessador 8086 (Compatível com AMD e Intel) é um aplicativo muito bom para quem está iniciando e quer se aprofundar em

linguagem de máquina. Ele executa o programa semelhante a um processador real possibilitando ver passo a passo as alterações nos registradores, memórias, pilhas, variáveis e flags. A posição de memória pode ser analisada e editada.

Para verificar o funcionamento do emulador, nada melhor que usá-lo com exemplos e explicações. Após instalar o emu8086 você deverá abrir o aplicativo, a figura abaixo apresenta a tela de entrada.

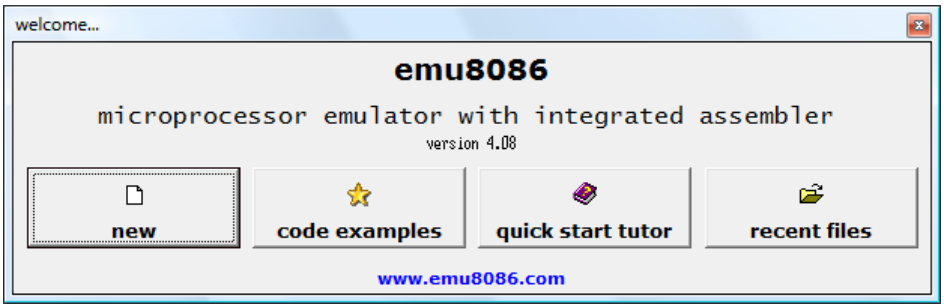

Figura 1: tela de entrada do emulador

Nesta tela há as opções de criar um código novo (new), exibir alguns exemplos de códigos (code example), acessar um tutorial resumido (quick start tutor) ou abrir arquivos recentes (recent files),

Vamos começar criando pequenos códigos e analisar o que acontece com os registradores, portanto você deve escolher a opção new. A próxima tela irá solicitar o tipo de arquivo que você irá querer criar, vamos escolher a primeira opção “COM template”, após a escolha a seguinte tela deverá ser apresentada.

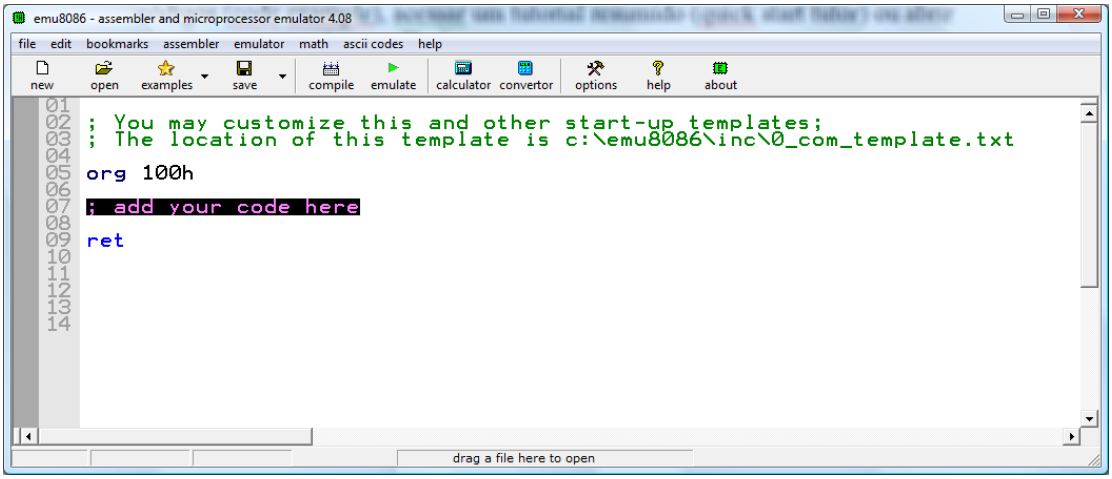

Figura 2: tela de edição do emulador

Agora já podemos digitar o código na área entre os códigos “org100h” e “ret”. Vamos digitar um código simples conforme ilustra a figura abaixo.

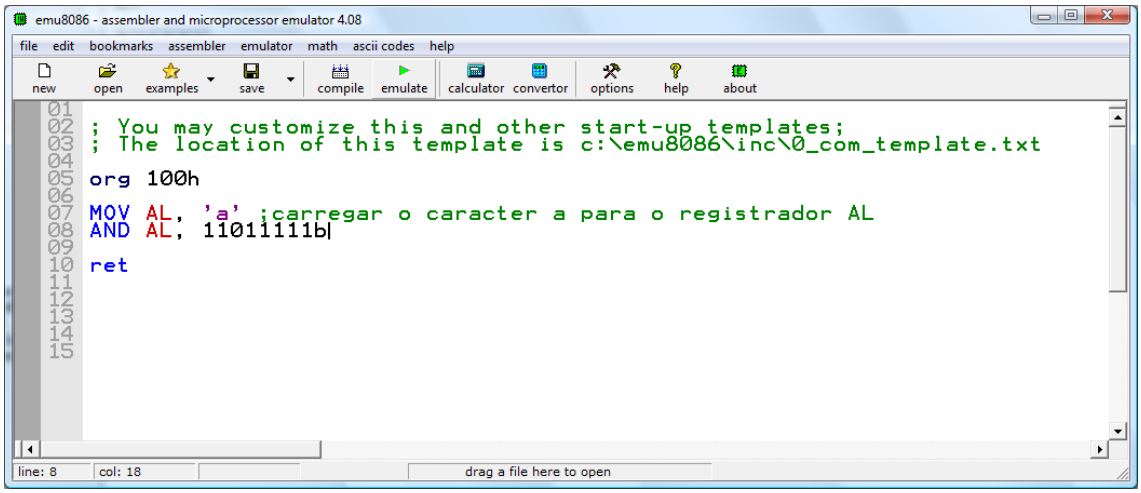

Figura 3: pequeno código em assembly.

Agora vamos executar o código, para isto clique no ícone emulate e irá abrir as seguintes telas.

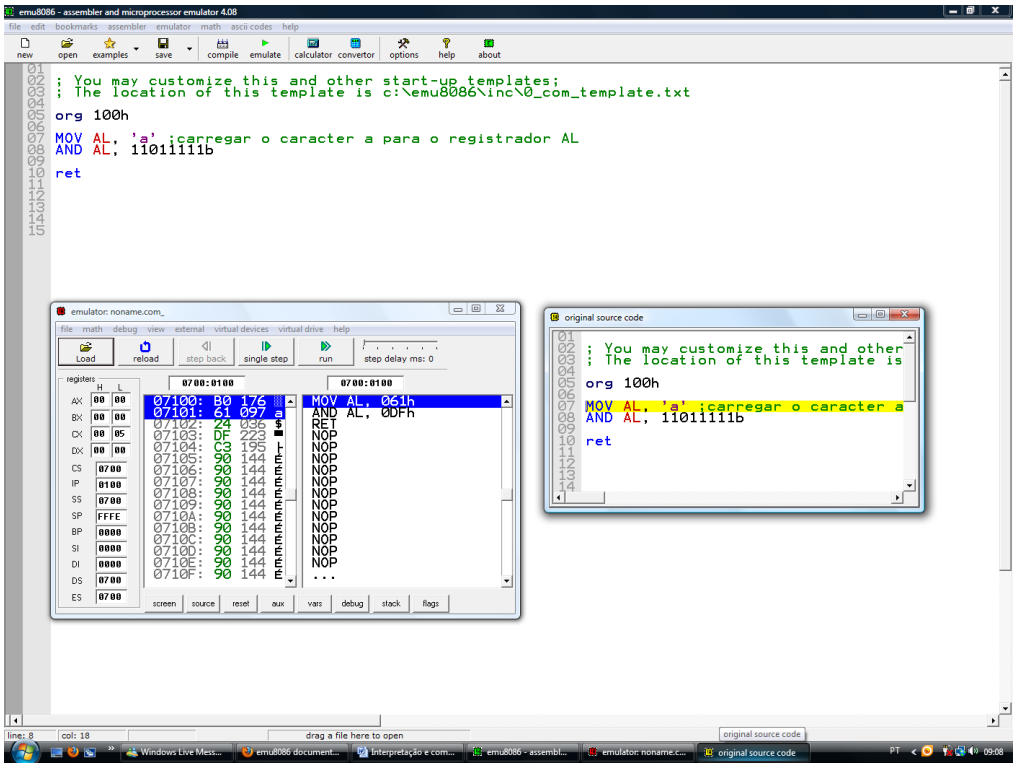

Figura 4: tela de execução

Como pode observar a tela a esquerda apresenta o status dos registradores (AX, BX, CX, DX, etc.) do processador, além disto é apresentada as posições de memória e qual informação está inserida em cada posição, há alguns ícones disponíveis que permitem carregar o código (Load), recarregar o código (reload), voltar passo a passo (step back), executar passo a passo (single stop) ou executar sem paradas (run). A tela do lado direito apresenta a execução do código. Vamos analisar o código passo a passo. A primeira linha

org 100h

Esta diretiva é requerida para um programa com extensões .COM contendo um simples segmento, por ter escolhido criar um arquivo “COM template” esta diretiva foi inserida automaticamente.

O código

MOV AL, ‘a’

Carrega o valor ASCII do caracter ‘a’ para a posição menos significativa do registrador A, veja que na tela apresentada no lado esquerdo há o

registrador AX que é composto de duas partes AH (mais significativo) e AL (menos significativo), a instrução acima carrega o valor 61 (01100001) para o AL. Veja na figura abaixo o valor ao registrador AL.

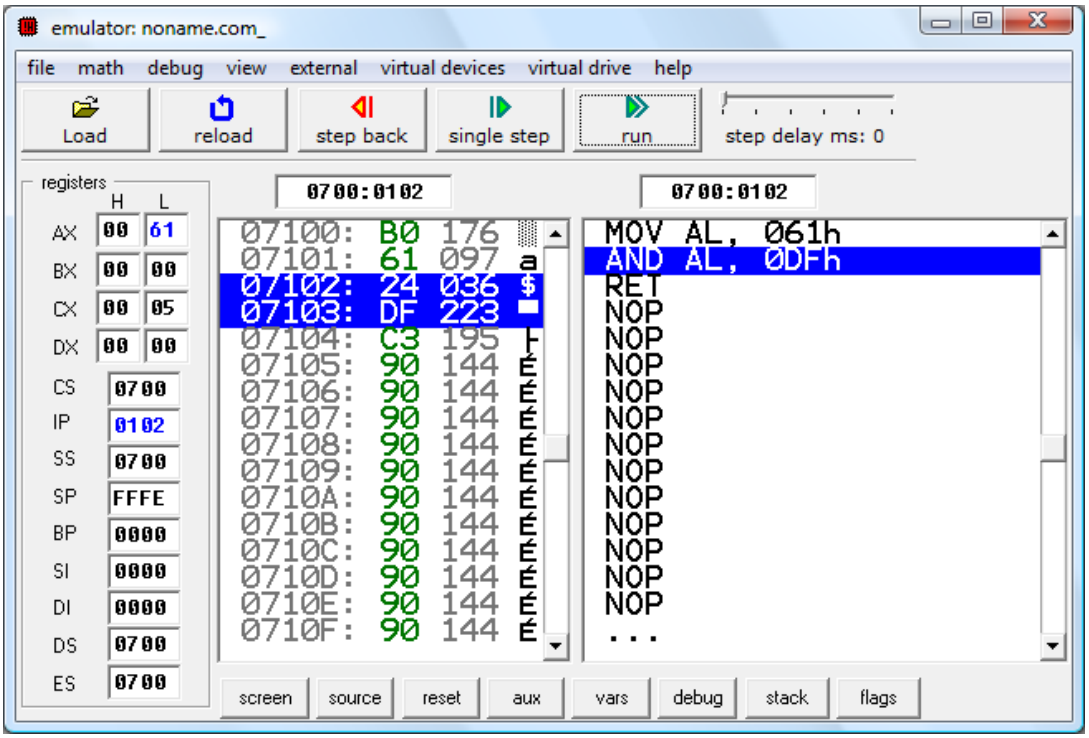

Figura 5: Registrador AX alterado.

A instrução

AND AL, 11011111b

Aplica a operação AND do valor contido no registrador AL e o valor em binário 11011111, o resultado 01000001 será inserido no registrador AL, abaixo encontra-se o cálculo da operação AND entre o valor contido no acumulador e o 11011111

Valor contido no acumulador	01100001
Valor a ser comparado	11011111
Resultado da operação AND	01000001

Veja na figura abaixo o valor do registrador AL após a operação AND, o valor inserido no acumulador está em hexadecimal, por este motivo é inserido o

número 41 (Hexadecimal), este número convertido para binário será 01000001

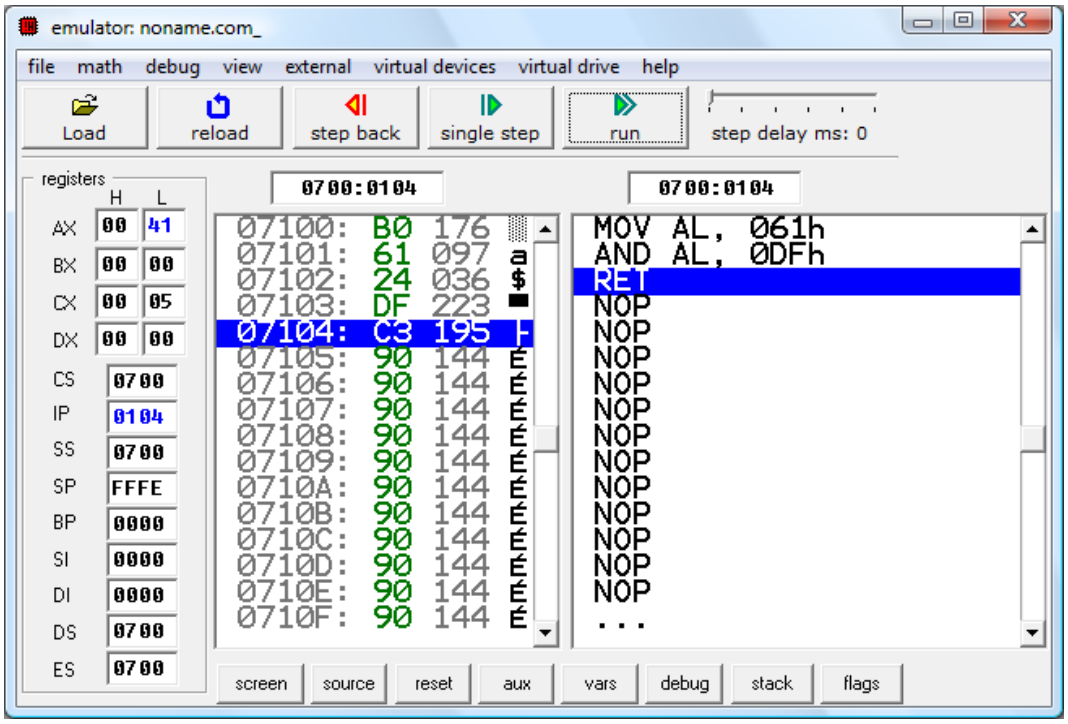

Figura 6: Valor de AL após execução da operação AND

Veja que os outros registradores foram sendo alterados durante a execução do programa, os registradores CS e IP, formam juntos, o endereço da memória onde se encontra o dado ou a instrução. O registrador CS, para este caso, inicia em 0700 e o registrador IP somado ao valor do CS forma o endereço de memória, neste caso iniciando em 07100 e chegando até 07104. O último comando

ret

Retorna o controle para o sistema operacional.

Para executar o código passo a passo, clique no botão “single step”. Observe as alterações nos registradores e na memória a cada clique dado neste botão

Soma de dois números

```
org 100h

MOV AL, 3 ;carregar o valor 3 a para o registrador AL
```

```
ADD AL, -4

ret
```

O programa acima soma dois números e o resultado fica armazenado no registrador AL. Este código afeta bits do registrador FLAG, para analisar o registrador você deve clicar no botão flag que se encontra na parte inferior da tela onde são apresentados os registradores, veja na figura abaixo.

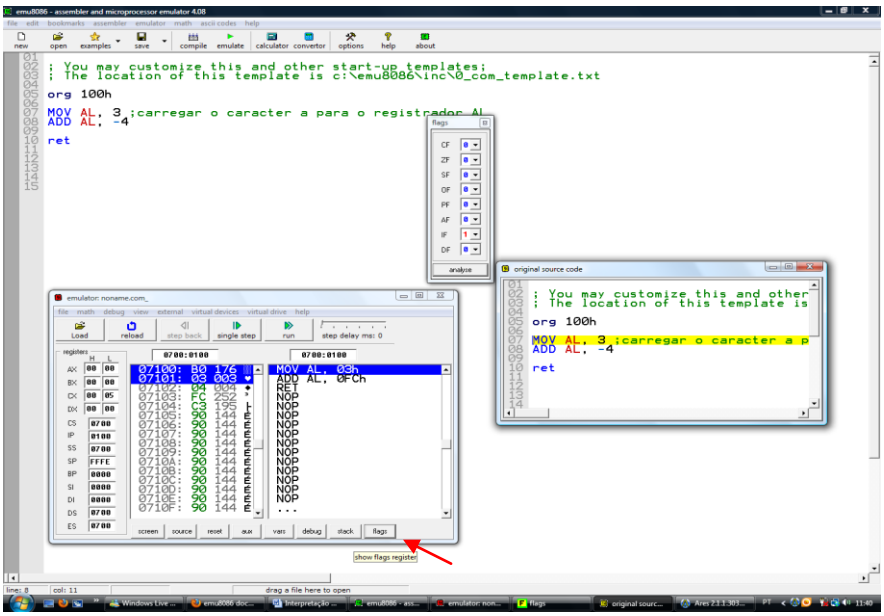

Figura 7: Registrador flag

Na execução deste código, os bits do registrador flag (CF, ZF, SF, DF, PF, AF e DF) encontram-se zerados, somente o bit IF tem o valor 1.

Vamos relembrar os bits do registrador flag.

CF (Carry Flag):- Este bit é configurado para 1 quando o valor de uma operação passa do número 255 (quantidade máxima aceita por um byte). Exemplo, quando é somado 255 + 1 o valor vai para 256, isto é considerado um overflow, sendo assim o bit CF vai para 1.

ZF (Zero Flag):- Configurada para 1 se o valor da operação for zero.

SF (Sign Flag):- Configurada para 1 se o resultado da operação for negativo. Quando o resultado de operação for positiva este bit permanece em 0.

DF (Direction Flag):- Este bit é usado em algumas instruções para indicar a direção em que as operações com strings são realizadas. Se ele estiver com o valor 1 então foi feito um decremento de memória, se o valor for igual a 0 houve um incremento de memória, ou seja, irá indicar se a posição de memória está sendo incrementada ou decrementada.

PF (Paraty Flag):- Este bit é configurado para 1 quando há no byte inferior do resultado de alguma operação aritmética ou lógica um número par de “1’s”, caso a quantidade de “1’s” for ímpar então este bit será zero.

AF (Auxiliary Carry Flag):- Este bit é configurado quando tem o vai um de uma operação de adição ou subtração.

IF (Interrupt Flag):- Se este bit for igual a 1 então foi habilitado a ocorrência de interrupção, se este bit for zero então a interrupção está inibida.

No código proposto da figura 7, os bits SF, PF são alterados, o bit IF estará mantendo a interrupção habilitada, a figura abaixo apresenta os bits afetados pela operação.

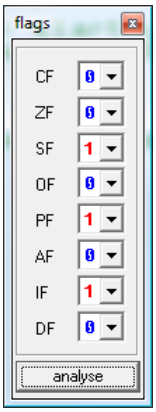

Figura 8: bits do registrador Flag

Comparação entre dois números

Para comparar dois números podemos usar o seguinte código

```
org 100h

MOV AL, 5
MOV BL, 5
```

```
CMP AL, BL
RET
```

O primeiro comando, depois do org 100h

MOV AL, 5 - irá carregar para a parte menos significativa do registrador AX o valor 5

O segundo comando

MOV BL, 5 - irá carregar para a parte menos significativa do registrador BX o valor 5

O terceiro comando CMP AL, BL, compara os dois valores contidos nos registradores AL e BL, o resultado não será armazenado, mas afetará bits do registrador FLAG, a figura 9 apresenta os bits afetados.

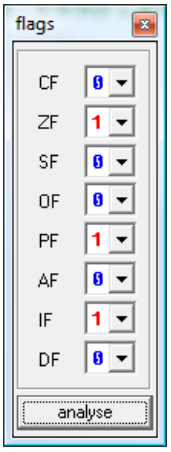

Figura 9: bits do registrador flag afetados

Comparando número

O código abaixo compara dois números, para o código em específico é comparado o 6 e o 7.

```
org 100h
  MOV AL, 6
```

```
    CMP AL, 7
    JLE label1
    MOV AL, 6
    MOV AH, 7
    JMP exit
label1:
    MOV AL, 7
    MOV AH, 6
exit:
ret
```

O primeiro comando “MOV AL, 6” carrega o valor 6 para o acumulador, o comando CMP AL,7 compara o valor 7 com o valor que foi armazenado no acumulador, ou seja, irá comparar o valor 7 com o valor 6. O próximo comando “JLE label1” determina que seja dado um salto para o código logo abaixo do “label1:” caso o resultado da comparação for menor ou igual, para este caso 6 é menor que 7, portanto o programa será desviado para “label1”. Os códigos abaixo do “label1:”, “MOV AL, 7” e “MOV AH, 6” carrega o valor 7 para os bits menos significativos do acumulador (AL) e os bits mais significativos do acumulador (AH) receberá o valor 6.

Caso não for dado o salto, ou seja, caso o valor no acumulador seja maior que o valor a ser comparado então o programa não será desviado e será executado os comandos “MOV AL, 6” e “MOV AH, 7”. O código “JMP exit” dá um salto incondicional para o label “exit:”, neste label há o código ret que retornará o controle para o sistema operacional. Para que você entenda o funcionamento deste código, troque os valores 6 e 7 e execute passo a passo para ver o comportamento dos registradores gerais e do registrador flag.

Referências

Reed D. **"**A Balanced Introduction to Computer Science and Programming" tradução do capítulo 14. Creighton University**,** Prentice Hall, 2004. disponível em http://professores.faccat.br/assis/davereed/14-DentroDoComputador.html

STALLINGS, W. **Arquitetura e Organização de Computadores:** Projeto Para o Desempenho. 5. ed. Sao Paulo: Prentice Hall, 2004.

TANENBAUM, A. S. Organização e Arquitetura de Computadores. %. Ed. São Paulo: Pearson, Pretice Hall, 2007.

Responsável pelo Conteúdo:
Prof. Ms. Vagner da Silva

www.cruzeirodosul.edu.br

Campus Liberdade

Rua Galvão Bueno, 868

01506-000

São Paulo SP Brasil

Tel: (55 11) 3385-3000