# Instruções de Salto e Loops

Instruções de salto alteram o fluxo sequencial de um programa. Uma determinada instrução de salto faz com que a próxima instrução não esteja no próximo endereço de memória. Como o PC é o registrador que controla o ponto da execução, quando temos uma instrução de salto, o PC é carregado com o novo endereço do salto.

## Tipos de instruções de salto:

As instruções de salto pode ser classificadas como incondicionais e condicionais.

As incondicionais saltam sem nenhuma condição estabelecida, já as condicionais só saltam se uma condição for satisfeita.

### Incondicional

Instrução de salto incondicional do *assembly* do 8085:

`JMP` *endereço*.

Esta instrução simplesmente carrega o valor do endereço no PC. Este endereço, que é o complemento da instrução, tem 2 bytes e portanto trata-se de uma instrução de 3 bytes. O montador *assembly* (*assembler*) nos oferece uma facilidade: Ao invés de precisarmos de um endereço de memória para por como complemento do JMP, Ele nos oferece o conceito de rótulos (*labels*) sendo um rótulo um substituto *string* para o endereço. Abaixo o exemplo de um *loop* infinito usando rótulo e a instrução JMP:

Exemplo 1

```
      MVI A, 5
loop: INR A
      JMP loop
```

A instrução INR faz o incremento do registrador (neste exemplo, o A). Vemos a *string loop* apenas marcando o endereço da instrução INR A. Quando a instrução JMP *loop* é executada o PC é carregado com o endereço da instrução INR A (o montador faz essa tradução para nós). E começa tudo novamente. O 8085 entra em um *loop* infinito.

### Condicional

As instruções de salto condicionais dependem de uma condição ser satisfeita para que o salto ocorra. Estas condições estão associadas aos flags da ULA.

Na instrução JNZ *endereço* significa: salta se o flag Z for zero. Lembrando que o flag Z é zero quando o resultado da última instrução executada na ULA não foi zero. Podemos fazer uma associação mais rápida e pensar que JNZ salta se o último resultado da ULA não foi zero. Da

mesma forma JZ *endereço* indica um salto somente se o flag Z for 1 (quando o resultado da última operação da ULA foi zero). Da mesma forma podemos pensar que se trata de um salto se o resultado da última operação da ULA foi zero.

Abaixo um exemplo de *loop* finito controlado pelo número de iterações

Exemplo 2

```
       MVI A, 0
       MVI C, 5
loop:  INR A
       DCR C
       JNZ loop
       HLT
```

Lembrando que todo *loop* tem uma inicialização (duas primeiras instruções, no caso do exemplo), um corpo (que se repete e no nosso caso é a instrução INR A) e o controle do *loop* (neste caso as instruções DCR C e JNZ loop). A instrução DCR faz o decremento do registrador C que trabalha como o contador do *loop*. O programa funciona da seguinte forma: C é carregado com 5 e na primeira iteração, o A é incrementado (vai para 1), o C é decrementado (vai para 4) e é feito o teste do *flag* com o JNZ *loop*. Se o resultado da última operação na ULA (no caso o DCR C) for diferente de zero, o 8085 salta para *loop*. Isso acontece até o C ser zero, quando o teste falha para o salto. Quanto não há o salto, a instrução HLT será executada e o programa acaba. É importante notar que trocar as instruções INR A e DCR C pois o teste se refere à última operação executada na ULA.

No exemplo abaixo, usamos JMP e JZ para fazer a mesma coisa do Exemplo 2

Exemplo 3

```
       MVI A, 0
       MVI C, 5
loop:  INR A
       DCR C
       JZ fim
       JMP loop
fim:   HLT
```

Veja que neste exemplo 3 a condição para o primeiro salto muda. Se a última operação na ULA for zero, salta para o fim (HLT) caso contrário segue e existe um salto incondicional para o INR A. O corpo do loop vai ser executado até que C seja zero como no Exemplo 2. Veja que utilizar o modelo do Exemplo 2 é a forma mais fácil de se implementar um loop quando se sabe o número de vezes que o corpo do loop deve ser executado.

No exemplo abaixo, vamos adicionar o valor de B ao de A (inicialmente zero) C vezes. Isto é o mesmo que fazer a multiplicação dos conteúdos de B e C.

Exemplo 4

```
      MVI A, 0
      MVI B, 3
      MVI C, 5
loop: ADD B
      DCR C
      JNZ loop
      HLT
```

Ao final das instruções B será adicionado ao A C vezes tendo o valor 15 (OF em hexadecimal)