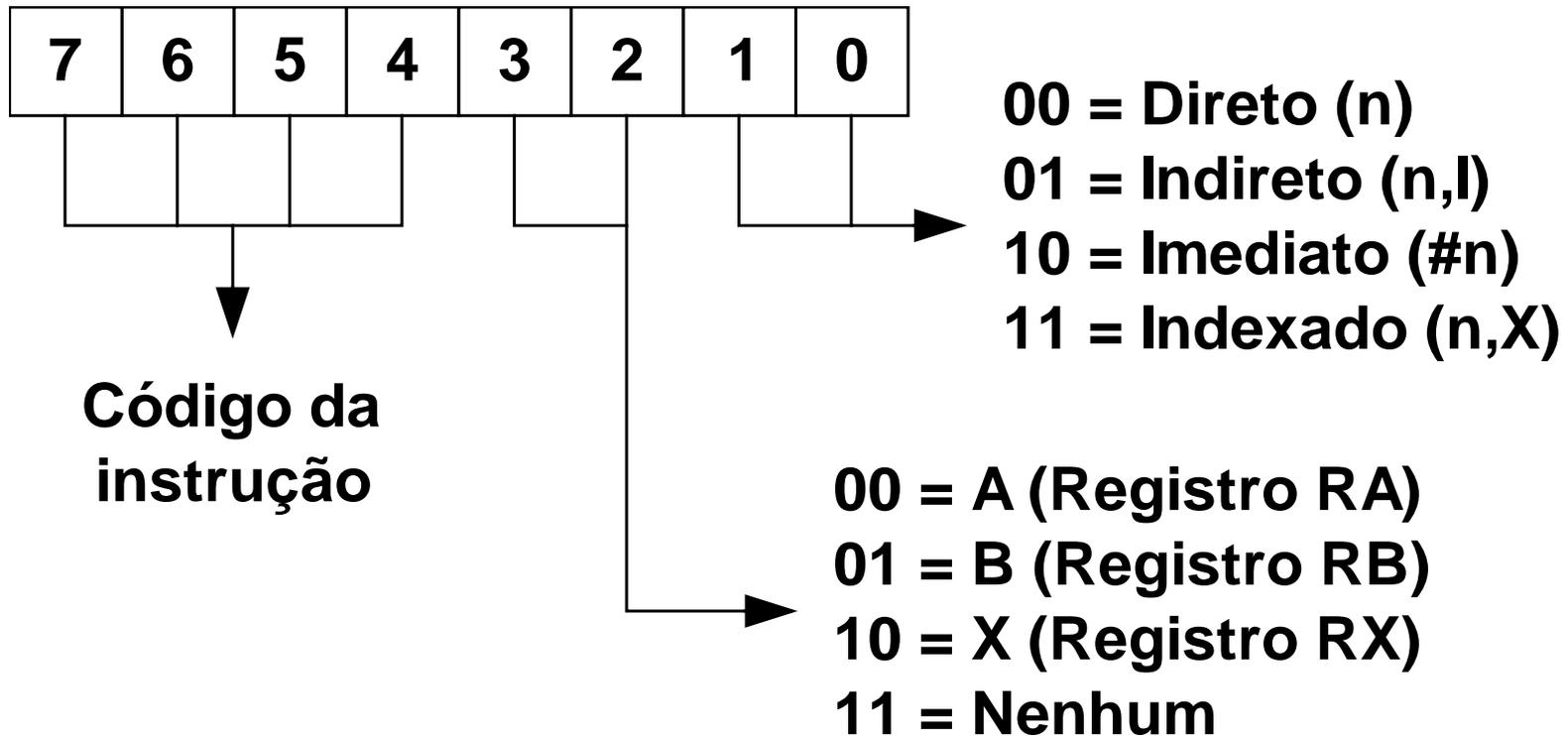


RAMSES =  
NEANDER + Novas Instruções

Prof. Sérgio Luis Cechin

# Relembrando...



# Relembrando...

- Modo **direto** - “n”
  - Compatível com o NEANDER
  - Ex: LDR A,NOME
- Modo **indireto** - “ n,l”
  - Ex: LDR A,NOME,l
- Modo **imediato** - “#n”
  - Ex: LDR A,#NOME
- Modo **indexado** - “n,X”
  - Ex: LDR A,NOME,X

# Códigos de Condição

- Refletem o estado da última operação executada
- Não refletem o conteúdo atual de nenhum registrador
- N
  - Indica se o resultado da última instrução foi um número negativo (“interpretação” de complemento de 2)
- Z
  - Indica se o resultado da última instrução foi “zero”
- C
  - Seu significado depende da última instrução
  - As instruções são: ADD, SUB, NEG e SHR

# Instruções

# Gerenciamento (iguais ao NEANDER)

- NOP: Nenhuma operação
- HLT: Para a execução

# Desvios

- Relembrando: nos desvios, o último acesso à memória não é efetuado durante a execução da instrução
- Este último acesso é realizado no próximo ciclo de busca (*fetch*) de instrução
- Portanto, não faz sentido usar desvios com modo de endereçamento imediato

# Desvios

## (Iguais ao NEANDER)

- JMP      *<end>*
- JN        *<end>*
- JZ        *<end>*

# Desvios (Novidade)

- JC            *<end>*  
  *if* (C==1) PC ← n

Vamos usá-los junto com as próximas instruções

# Transferências

- LDR  $\langle reg \rangle \langle end \rangle$   
reg ← MEM(end)
- STR  $\langle reg \rangle \langle end \rangle$   
reg → MEM(end)

# Exercício 1<sub>5</sub>

- Codificar as seguintes instruções

**LDR        A, 128**  
**STR        B, 129, I**  
**LDR        X, #130**  
**LDR        B, 131, X**  
**STR        X, 132, X**



LDR = 0010  
STR = 0001

A = 00  
B = 01  
X = 10

Direto = 00  
Indireto = 01  
Imediato = 10  
Indexado = 11

# Exercício 2<sub>5</sub>

- Indique a sequência de leitura e escritas, a partir do “PC”, realizadas em cada uma das seguintes instruções

**LDR            A, 128            Ex:  $A \leftarrow \text{MEM}(\text{MEM}(\text{PC}))$**

**STR            B, 129, I**

**LDR            X, #130**

**LDR            B, 131, X**

**STR            X, 132, X**

- Perceba que o único endereço necessário é o valor do PC!

# Exercícios 3<sub>10</sub>

- Implementar, em *assembler* do RAMSES, as seguintes atribuições “C” abaixo
- Utilize programação simbólica
- No caso de 16 bits, utilize o menor endereço para o byte menos significativo

```
char c;  
unsigned char k;  
unsigned short int i;  
  
k = c;  
k = i;  
i = k;
```

Observe que:

**char** = 8 bits com sinal

**unsigned char** = 8 bits sem sinal

**unsigned short int** = 16 bits sem sinal

# Lógicas

## (Iguais ao NEANDER)

- OR            <reg>            <end>
- AND          <reg>            <end>
- NOT          <reg>
- Onde
  - <reg> = A, B ou X
  - <end> = direto, indireto, imediato ou indexado

# Exercício 4<sub>5</sub>

- Implementar em *assembly* do RAMSES, o seguinte trecho em “C”

```
unsigned char j,k;  
  
if ( bit(5,k) == 1 ) {  
    j = k;  
}
```

Observar a “criação”  
de uma função

Dica: usar máscara AND para isolar o bit 5

# Aritméticas

## (Igual ao NEANDER)

- **ADD**      **<reg> <end>**
- Onde
  - <reg> = A, B ou X
  - <end> = direto, indireto, imediato ou indexado

# Aritméticas (Novidade)

- **SUB**      *<reg> <end>*

$$r \leftarrow r - \text{MEM}(\text{end})$$

- **NEG**      *<reg>*

$$r \leftarrow 0 - r$$

- **SHR**      *<reg>*



# Exercício 5<sub>5</sub>

- Implementar em *assembly* do RAMSES o seguinte trecho de código

```
char c;  
short int x;  
  
x = c;
```

sizeof(char) = 8  
sizeof(short int) = 16



Como é implementada a atribuição quando:

- “c” é positivo?
- “c” é negativo?

# Exercício 6<sub>10</sub>

- Implementar em *assembly* do RAMSES o seguinte trecho de código

```
char x,y;  
  
if ( x >= y ) {  
    x = x - y;  
}
```

Utilize SUB,

- para realizar a comparação
- para realizar a diferença

Para realizar a comparação de números **com sinal**, deve-se usar JN

# Código de Condição “C”

- É gerado por quatro instruções:
  - ADD, SUB, NEG e SHR
- Instrução ADD
  - Corresponde ao “vai-um” do bit 7
- Instruções SUB e NEG
  - Corresponde ao “empresta um” do bit 7
- Instrução SHR
  - Corresponde ao bit 0

# Exemplo – ADD

- Sequência de instruções

- LDR A,#H85

- ADD A,#H93



ADD

C

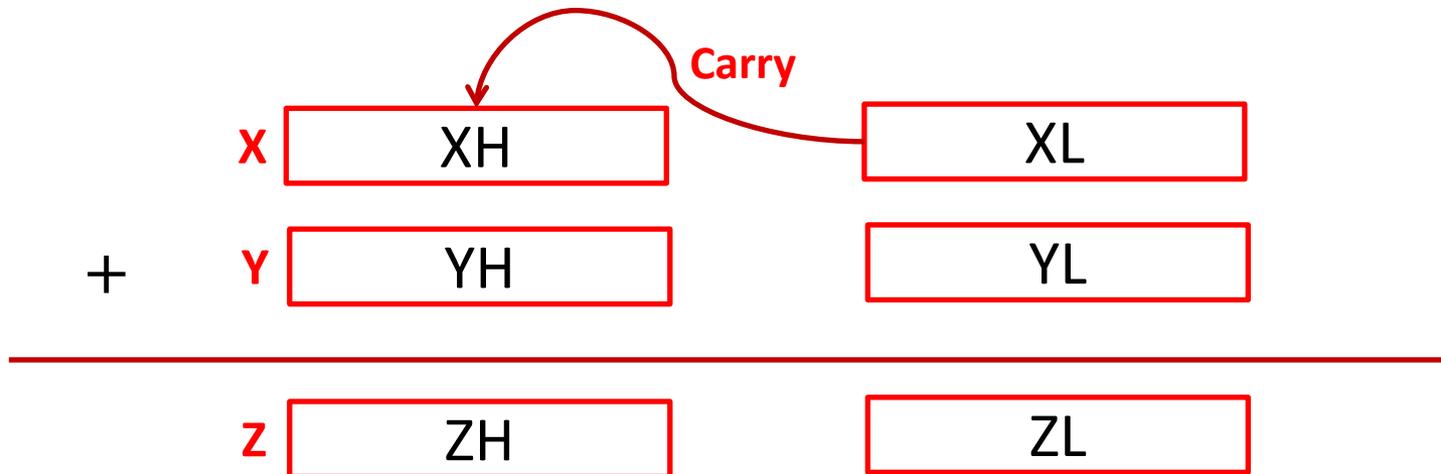


Carry  
(vai-um)

# Exercício 7<sub>10</sub>

- Implementar em *assembly* do RAMSES a soma de dois números de 16 bits sem sinal

$$Z = X + Y$$

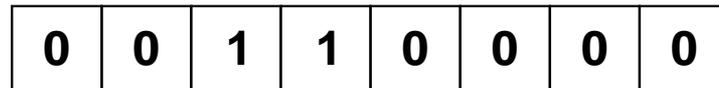


# Exemplo – SUB

- Sequência de instruções

LDR     A,#H45

SUB     A,#H30



SUB

C

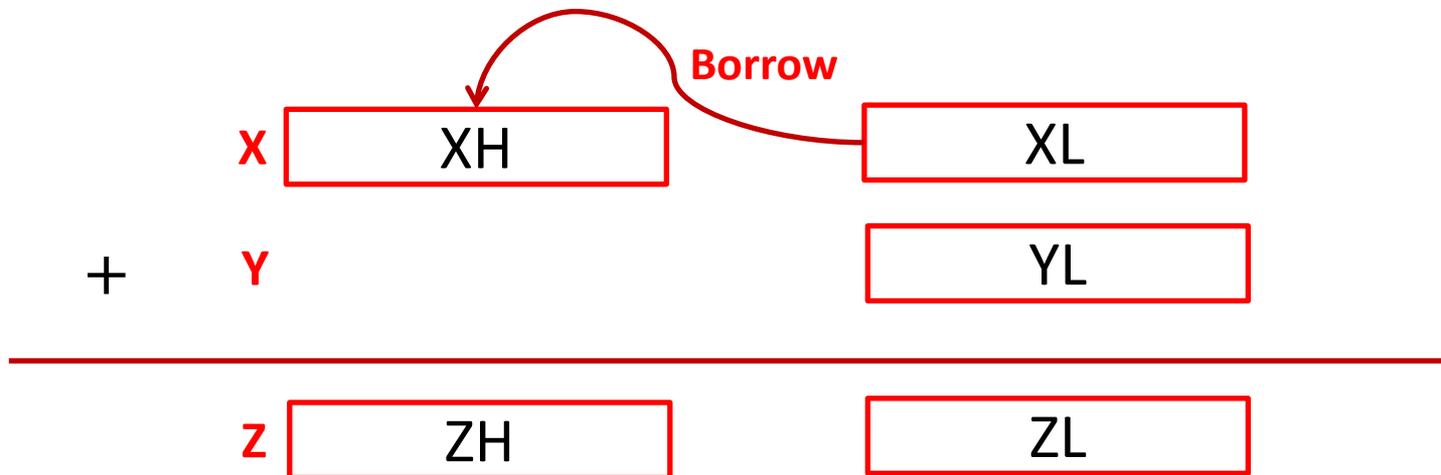


**!Borrow  
(empresta-um)**

# Exercício 8<sub>10</sub>

- Implementar em *assembly* do RAMSES a diferença de dois números: X com 16 bits e Y com 8 bits. Ambos sem sinal

$$Z = X + Y$$

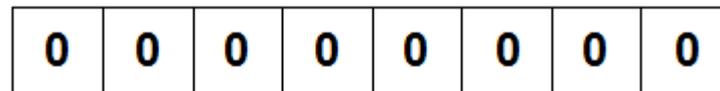


# Exemplo – NEG

- Sequência de instruções

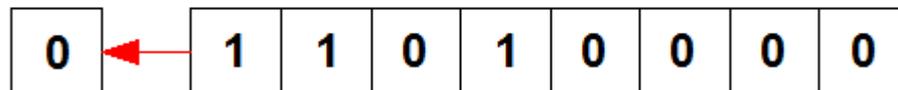
LDR X,#H30

NEG X



NEG

C



**!Borrow  
(empresta-um)**

# Exemplo – SHR

- Sequência de instruções

LDR B,#H45

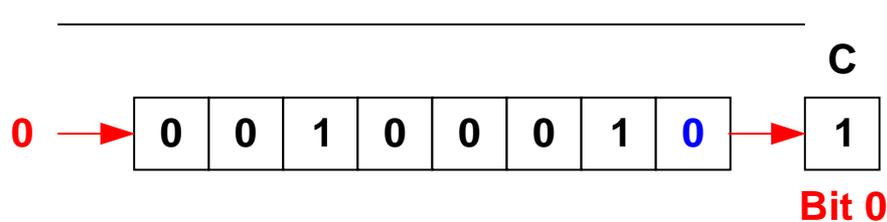
SHR B

SHR B

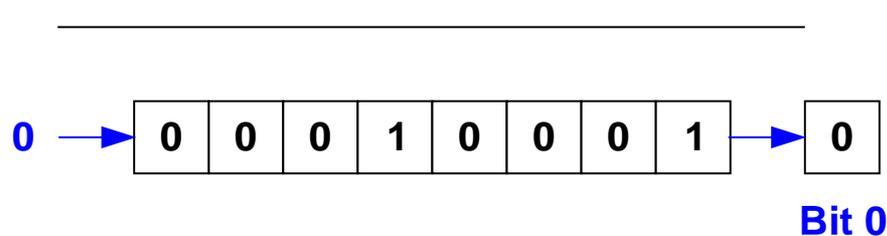
B, #H45

B

B



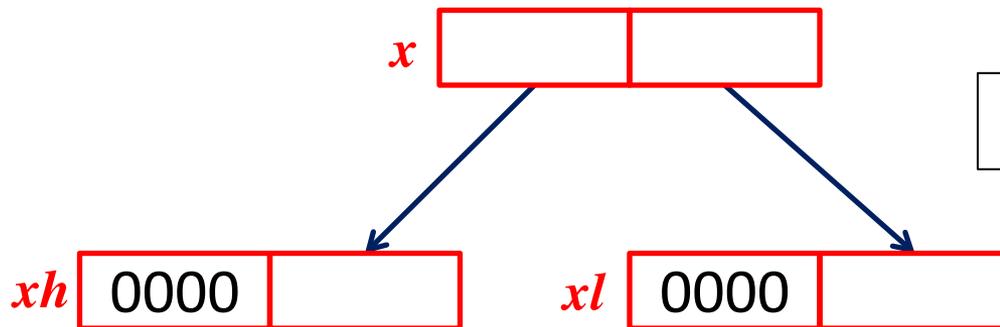
SHR



SHR

# Exercício 9<sub>10</sub>

- Implementar em *assembly* do RAMSES o algoritmo representado abaixo
  - Considere “*x*”, “*xh*” e “*xl*” como “*unsigned char*”



Utilize SHR para deslocar bits

# RAMSES

Instrução	Operação	N	Z	C	Descrição
NOP	Nenhuma operação				Nenhuma operação
STR r end	MEM(end) $\leftarrow$ r				Armazena registrador na memória ( <b>store</b> )
LDR r end	r $\leftarrow$ MEM(end)	$\updownarrow$	$\updownarrow$		Carrega registrador da memória ( <b>load</b> )
ADD r end	r $\leftarrow$ r + MEM(end)	$\updownarrow$	$\updownarrow$	$\updownarrow$	Adição ( <b>soma</b> memória ao registrador)
OR r end	r $\leftarrow$ r $\vee$ MEM(end)	$\updownarrow$	$\updownarrow$		“ou” lógico
AND r end	r $\leftarrow$ r $\wedge$ MEM(end)	$\updownarrow$	$\updownarrow$		“and” lógico
NOT r	r $\leftarrow$ $\neg$ r	$\updownarrow$	$\updownarrow$		Inverte ( <b>complementa</b> os bits do registrador)
SUB r end	r $\leftarrow$ r - MEM(end)	$\updownarrow$	$\updownarrow$	$\updownarrow^{(1)}$	Subtração ( <b>subtrai</b> memória do registrador)
JMP end	PC $\leftarrow$ end <sup>(2)</sup>				Desvio incondicional ( <b>jump</b> )
JN end	if N=1 then PC $\leftarrow$ end <sup>(2)</sup>				Desvio condicional se < 0 ( <b>jump on negative</b> )
JZ end	if Z=1 then PC $\leftarrow$ end <sup>(2)</sup>				Desvio condicional se =0 ( <b>jump on zero</b> )
JC end	if C=1 then PC $\leftarrow$ end <sup>(2)</sup>				Desvio condicional se carry=1 ( <b>jump on carry</b> )
NEG r	r $\leftarrow$ 0 - r	$\updownarrow$	$\updownarrow$	$\updownarrow$	Troca de sinal ( <b>negate</b> )
SHR r	0 $\rightarrow$ <span style="border: 1px solid black; padding: 2px;"> </span> $\rightarrow$ <span style="border: 1px solid black; padding: 2px;"> </span> $\rightarrow$ C	$\updownarrow$	$\updownarrow$	$\updownarrow$	Deslocamento para direita ( <b>shift right</b> )
HLT	parada				Parada ( <b>halt</b> )

# Exercício 10

- Implementar um programa capaz de calcular o **número de caracteres** de um vetor, conforme o programa abaixo
  - O resultado deverá ser colocado no endereço H80
  - O vetor inicia no endereço H81
- O “vetor” é um *string* em “C”, que termina com ‘\0’

```
unsigned char n,i;  
char vetor[50];  
  
n = i = 0;  
while ( vetor[i] != 0 ) {  
    n++;  
    i++;  
}
```

Utilize os três registradores!

Utilize modo **indexado** para acesso ao vetor

(usar indexado sempre que o vetor estiver em endereço fixo)

# Exercício 11

- Implementar um programa capaz de calcular o número de caracteres de um vetor, conforme o programa abaixo
- O “vetor” é um *string* em “C”, que termina com ‘\0’

```
unsigned char n;  
char *p;  
char vetor[50];  
  
p = vetor;  
n = 0;  
while ( *p != 0 ) {  
    ++n;  
    ++p;  
}
```

Utilize os três registradores!

Utilize modo **indireto** para acesso ao vetor

(usar indireto sempre que forem usados ponteiros)

# Exercício 12

- Implementar o cálculo da raiz quadrada, conforme o seguinte algoritmo
  - N: Número que se deseja calcular a raiz
  - K: Valor da raiz
- O algoritmo implementado baseia-se na expressão

$$N = \sum_{k=1}^{\sqrt{N}} (2k - 1)$$

- Conta-se quantas vezes é necessário somar (2K-1), para atingir (N)
  - Esse número de vezes (K) é a raiz inteira de (N)

```
/*
    K = SQRT(N)
*/
unsigned char N, I, S, K;
S = K = I = 1;
while (S < N) {
    I += 2;
    S += I;
    K++;
}
```

# RAMSES

## Instruções

Prof. Sérgio Luis Cechin