ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES I

Computador Cleópatra

Programação Assembly e Alto Nível

prof. Dr. César Augusto M. Marcon
prof. Dr. Edson Ifarraguirre Moreno

Descrição de Vetores

- Implementação de vetores na Cleópatra deve ser realizada com ponteiros → modo de endereçamento indireto

- Exemplo:

Codifique o programa C abaixo, para o assembly da arquitetura Cleópatra. O programa DEVE ter todos os modos de endereçamento definidos na arquitetura. Comente todas as linhas do assembly para identificar o efeito de cada uma na arquitetura alvo

```
#define TAM_VET 4
int cont = 0;
int vet[TAM_VET] = {0, 0, 0, 0};

void main()
{
    while(cont < TAM_VET)
    {
        vet[cont] = cont + 3;
        cont++;
    }
}
```

Descrição de Vetores

```

.code
  Enquanto:
    lda cont           ; AC ← cont
    not
    add #1             ; AC ← (-cont)
    add #4             ; AC ← (4-cont) e C, V, N e Z setadas
    jn Continua       ; Se (cont < 4) PC ← Continua, senão PC++
    jmp Fim,R         ; PC ← Fim (relativo)
  Continua:
    lda cont           ; AC ← cont
    add #3             ; AC ← cont + 3
    sta p,I           ; *p ← AC, ou seja, vet[cont] ← AC;
    lda p             ; +
    add #1             ; |
    sta p             ; +--> p++
    lda cont          ; +
    add #1             ; |
    sta cont          ; ---> cont++
    jmp Enquanto      ; PC ← Enquanto
  Fim:
    hlt
.endcode
.data
  cont: db #0
  p:    db #vet
  vet:  db #0, #0, #0, #0
.enddata

```

Descrição de Funções

- Chamadas de função requerem o salvamento do endereço subsequente ao da chamada, para que este seja recuperado no retorno da função
- Na Cleópatra, a chamada a função é feita pela instrução **JSR ENDEREÇO** e o retorno é feito pela instrução **RTS**
- Exemplo:

Fazer a função maior, que lê o maior de dois valores colocados nas variáveis **A** e **B** e retorna o resultado no **AC** (acumulador)

```
int C, A = 7, B = 13;
```

```
void main()  
{  
    C = maior();  
}
```

```
int maior()  
{  
    if(A > B)  
        return A;  
    return B;  
}
```

Descrição de Funções

```
.code
main:          ; void main() {
    jsr maior  ; |
    sta C      ; +-> C = maior(A, B);
    hlt        ; }

maior:         ; int maior() {
    lda A      ; |
    not        ; |
    add #1     ; | (-A)
    add B      ; | (B - A)
    jn a_ah_maior ; +-> if(A > B) goto a_ah_maior
    lda B      ; |
    rts        ; +-> return B;
a_ah_maior:   ; |
    lda A      ; |
    rts        ; +-> return A;
endcode

.data
a: db #7
b: db #13
c: db #0
enddata
```

Exercícios

1. Fazer um programa que escreve o valor 0x3A em um vetor de 10 posições
2. Fazer um programa com as seguintes características/funcionalidades: (i) contém 3 vetores (**A**, **B**, **C**), cada qual com 6 posições; (ii) deve percorrer os vetores **A** e **B** pesquisando em cada índice, qual é o maior valor entre as posições de cada vetor (i.e. **A[i]** comparado com **B[i]**), sendo que este valor deve ser armazenado no índice equivalente do vetor **C**. Este programa deve ser feito inicialmente em C/Java e posteriormente convertido para o assembly da Cleópatra
3. Faça o mesmo programa acima, só que agora utilizando uma chamada para função **maior** descrita anteriormente
4. Dado o programa descrito em linguagem C/C++, obtenha uma descrição em linguagem da Cleópatra com funcionalidade equivalente

```
int A[16], B[16];
int iguais = 0, k = 0;
void main()
{
    for(int i = 0; i < 4; i++)
    {
        if(memcmp(&A[k], &B[k], 4) == 0)
            iguais++;
        k = k + 4;
    }
}
```

```
int memcmp(int *v1, int *v2, int tam)
{
    for(int i = 0; i < tam; i++)
    {
        if(*v1 != *v2)
            return 1;
        v1++;
        v2++;
    }
    return 0;
}
```

Exercícios

5. Apresente uma descrição equivalente em C/Java para o programa descrito em linguagem de máquina da Cleópatra que segue

```
.code
    lda m2
    sta n21
    lda m1
    add #-1
    sta m1
loop:
    lda m1
    add #-1
    sta m1
    lda m2
    sta n11
    jsr soma16bits
    lda r1
    sta n2
    lda r11
    sta n21
    lda m1
    jz fim
    jmp loop
fim:
    hlt

soma16bits:
    lda n1
    add n2
    sta r1
    jc carry1
volta:
    lda n11
    add n21
    sta r11
    jc carry
    jmp fim_soma16
carry:
    lda r1
    add #01h
    sta r1
fim_soma16:
    rts
carry1:
    lda #01h
    sta c
    jmp volta
.endcode

.data
    n1:    db #00h
    n11:   db #00h
    n2:    db #00h
    n21:   db #00h
    c:     db #00h
    r1:    db #00h
    r11:   db #00h
    m1:    db #1Fh
    m2:    db #3Fh
.enddata
```