CESAR

Conjunto de Instruções
(Um operando, Flags, subrotinas e laços)

Prof. Sérgio L. Cechin

Grupos de Instruções

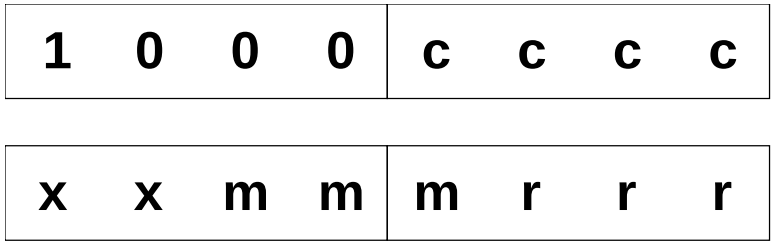
- Uso geral
- Instruções de dois operandos
- Instruções de um operando

Parte 2

- Instruções de desvio
- Manipulação de códigos de condição
- Chamada de subrotinas
- Controle de laço

Instruções de um operando

Codificação



- **cccc**
  - Indica o código da operação
- **xx**
  - Don´t care
- **mmm**
  - Indica o modo de endereçamento do operando
- **rrr**
  - Indica o registrador a ser usado para obter o operando

Utilização de NZ e C

TST, ADC e SBC

Instruções de um operando (continuação)

- Instruções aritméticas e de teste
  - **TST op**
    - Ajusta os valores de N e Z, de acordo com o operando
    - Não afeta o valor do operando
  - **ADC op**
    - Soma o carry ao operando
    - op + Cy → op
  - **SBC op**
    - Diminui o carry do operando
    - op - Cy → op

Exercício

- Implementar, em assembly do CESAR, o trecho (em “C”) abaixo. Inclua a declaração das variáveis:
- Considere que “int” tem 32 bits
  - Solução 1: tradicional (teste do carry)
  - Solução 2: usando ADC

```
unsigned int a, b;

a += b;
```

3 min

Solução Tradicional

- É necessários realizar a soma com “vai um” (carry)

```
        ORG     H8000
a:      DAW     [2]
b:      DAW     [2]

        ORG     0
        ADD     b, a            ; Soma parte mais significativa
        ADD     b+2, a+2        ; Soma parte menos significativa
        BCC     Pula
        ADD     #1, a           ;
Pula:
```

Solução
Com ADC

- É necessários realizar a soma com “vai um” (carry)

```
        ORG   H8000
a:      DAW   [2]
b:      DAW   [2]

        ORG   0
        ADD   b, a          ; Soma parte mais significativa
        ADD   b+2, a+2      ; Soma parte menos significativa
        ADC   a
```

Efeito sobre os CCs

cccc	instrução	significado	N	Z	C	V
0101	TST	op ← op	t	t	0	0
1010	ADC	op ← op + c	t	t	t	t
1011	SBC	op ← op - c	t	t	t	t

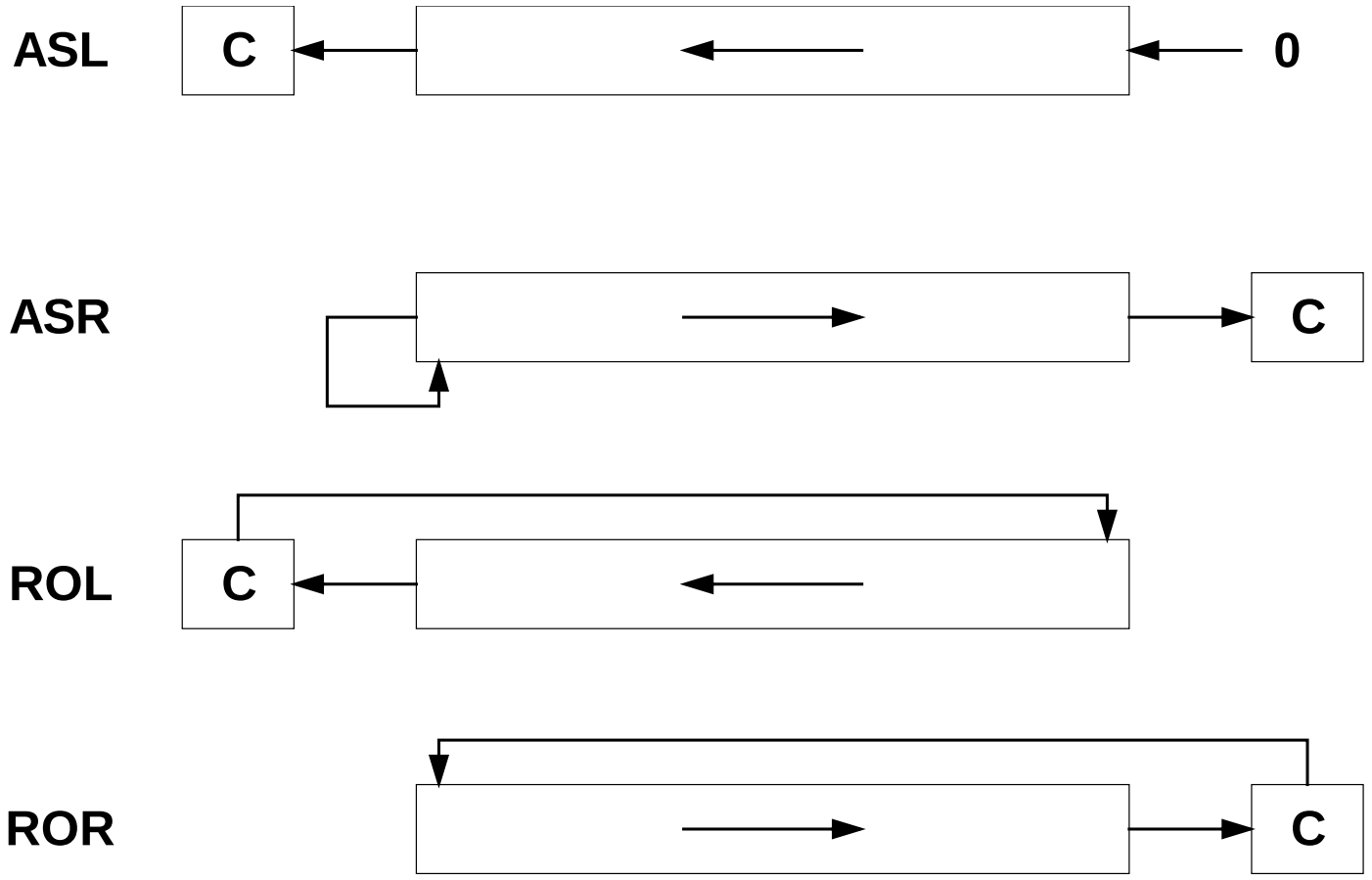
Shifts e Rotates

ASL, ASR, ROL e ROR

Instruções de um operando

- Instruções de movimento de bits
  - **ASL op** ; Aritmetic Shift Left
  - **ASR op** ; Aritmetic Shift Right
  - **ROL op** ; Rotate Left
  - **ROR op** ; Rotate Right

Instruções de um operando



Exercício

- Implementar, em *assembly* do CESAR, o trecho (em “C”) abaixo. Inclua a declaração das variáveis:
- Considere que “int” tem 32 bits
- Observe que se trata de uma multiplicação por uma constante

```
unsigned int a;

a *= 10;
```

3 min

Algoritmo

- Deseja-se calcular A = 10 x A, que pode ser manipulado da seguinte forma:

  $10 \times A = (8 \times A) + (2 \times A) = (4 \times (2 \times A)) + (2 \times A)$

- Algoritmo de cálculo

  $A = (2 \times A)$

  $A = A + (4 \times A)$

- Multiplicação por 2 ou 4 é equivalente a “shift”

  A <<= 1

  A += (A<<2)

Solução

```
        ORG     H8000
a:      DAW     [2]

        ORG     0
        ASL     a+2         ; a <<= 1
        ROL     a

        MOV     a, R0       ; R0R1 = a<<2
        MOV     a+2, R1
        ASL     R1
        ROL     R0
        ASL     R1
        ROL     R0

        ADD     R0, a       ; a += R0R1
        ADD     R1, a+2
        ADC     a
```

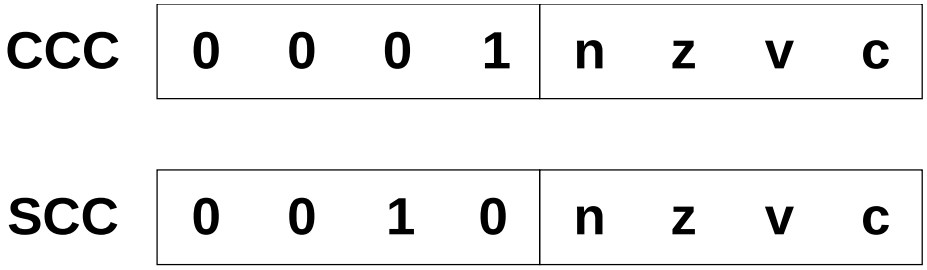
Efeito sobre os CCs

cccc	instrução	significado	N	Z	C	V
0110	ROR	op ← SHR(c & op)	t	t	lsb	xor
0111	ROL	op ← SHL(op & c)	t	t	msb	xor
1000	ASR	op ← SHR(msb & op)	t	t	lsb	xor
1001	ASL	op ← SHL(op & 0)	t	t	msb	xor

Manipulação de NZCV

CCC e SCC

Manipulação de códigos de condição

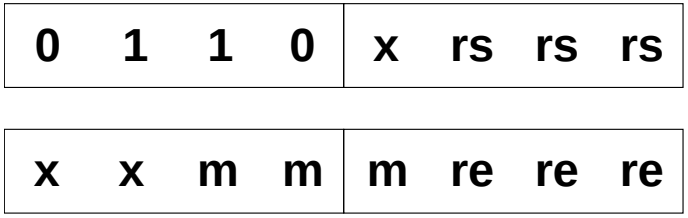


- Ocupam um byte
- São usadas para ligar ou desligar qualquer dos códigos de condição: N, Z, V e C
- Pode-se ligar ou desligar mais de um dos códigos de condição com uma única instrução
- **CCC [N] [Z] [V] [C]**
  - Usada para "zerar" (*clear*) o código de condição **N**, **Z**, **V** ou **C**
- **SCC [N] [Z] [V] [C]**
  - Usada para "ligar" (*set*) o código de condição **N**, **Z**, **V** ou **C**

Chamada de Subrotinas

JSR e RTS

Chamada de Subrotina



- **JSR Rs,Endereço** ; desvio para subrotina
  - Rs Registrador a ser salvo na pilha
  - mmm,Re Indica o endereço da subrotina
- Operação
  - Rs → Pilha
  - R7 → Rs
  - Endereço da subrotina → R7

Execução de JSR (Completa)

- Procedimento
  - Rs → Pilha
  - R7 → Rs
  - Endereço da subrotina → R7
- É equivalente a
  - Endereço da subrotina → R7(PC) → Rs → Pilha

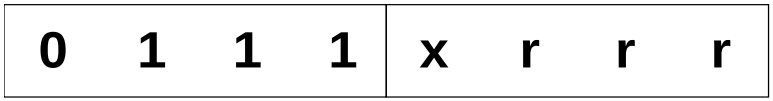
Execução de JSR (Quando Rs=R7)

- Procedimento
  - R7 → Pilha
  - R7 → R7
  - Endereço da subrotina → R7
- É equivalente a
  - Endereço da subrotina → R7 → R7 → Pilha
- Ou seja
  - Endereço da subrotina → R7 → Pilha

Cuidados com o JSR

- Se usado com o modo 000 (registrador)
  - O resultado é o mesmo que um NOP
- Se usado com o modo imediato
  - O desvio é feito para o próprio R7
  - A constante é usada como uma instrução (???)

Retorno da Subrotina



- RTS Rs ; retorna de subrotina
  - Em geral, “Rs” é o mesmo registrador usado na chamada da subrotina
- Operação
  - Rs → R7(PC)
  - Pilha → Rs

Execução de RTS (Completa)

- Operação
  - Rs → R7(PC)
  - Pilha → Rs
- É equivalente a:
  - Pilha → Rs → R7(PC)

Execução de RTS (Quando Rs=R7)

- Operação
  - R7 → R7(PC)
  - Pilha → R7
- É equivalente a:
  - Pilha → R7 → R7(PC)
- Ou seja
  - Pilha → R7

Exercícios

Exercício – 1

- Motivação: Verificar o funcionamento do JSR e RTS
- Realizar o teste de mesa no código abaixo, indicando sempre que a pilha é alterada: JSR e RTS

```
        ORG     0
        MOV     #H8000, R6
        JSR     R7, Rotina
        HLT
Rotina:
        RTS     R7
```

5 min

Exercício – 2

- Motivação: Verificar a operação da pilha
- Realizar o teste de mesa no código abaixo, indicando sempre que a pilha é alterada (JSRs e RTSs)

```
        ORG     0
        MOV     #H8000, R6
        JSR     R7, Rotina1
        HLT
Rotina1:
        JSR     R7, Rotina2
Rotina2:
        JSR     R7, Rotina3
Rotina3:
        RTS     R7
```

5 min

Exercício – 3

- Motivação: Passagem do PC em um registrador, para uso na subrotina
- Realizar o teste de mesa no código abaixo
- Observar a chamada usando “R0”
  - Qual o efeito do “DEC” dentro da “Rotina1”?

```
        ORG     0
        MOV     #H8000, R6
        JSR     R0, Rotina1
        DW      5
        HLT
Rotina1:
        DEC     (R0)+
        RTS     R0
```

Exercício – 4

- Motivação: Passagem de parâmetros pela pilha e manipulação dentro da rotina
- Realizar o teste de mesa no código abaixo
- Observar as alterações da pilha: JSR, RTS e alterações de R6
  - Como está a pilha, quando executando dentro da “getValue”?

```
        ORG     0
        MOV     #H8000, R6
        MOV     #5, -(R6)
        JSR     R7, getValue        ; getValue (5)
        ADD     #2, R6
        HLT

getValue:                           ; short getValue( short param )
        MOV     2(R6), R0
        RTS     R7
```

Exercício – 5

- Motivação: Mapeamento de programas em “C” para assembler, usando passagem de parâmetros pela pilha
- Implementar, em *assembly* do CESAR, o programa “C” escrito abaixo
- Utilize passagem de parâmetros pela pilha
- O retorno da função deve ser feita pelo registrador R0

```
void main() {
    unsigned short x;
    x = soma(3,7);
}

unsigned short soma(unsigned short a, unsigned short b) {
    return (a+b);
}
```

7 min

Solução – parte 1
Chamada da função

- Os parâmetros serão colocados na pilha de maneira que o primeiro parâmetro fique no topo
- Observar o ajuste de R6 após retornar da função

```
        ORG     0
        MOV     #H8000, R6
main:
        MOV     #7, -(R6)
        MOV     #3, -(R6)
        JSR     R7, soma
        ADD     #4, R6
        HLT
```

Solução – parte 2 (Função)

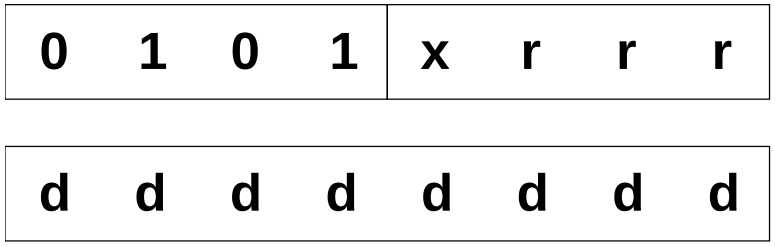
- Observar como está a pilha, quando a execução entra na função
  - 2(R6) = parâmetro “a”
  - 4(R6) = parâmetro “b”
  - 0(R6) = endereço de retorno da chamada

```
soma:
        MOV     2(R6), R0
        ADD     4(R6), R0
        RTS     R7
```

Controle de Laço

SOB

Controle de Laço



- **SOB Ri, ddd**
  - Subtract One and Branch if not zero
- Ocupa um byte mais o deslocamento
  - Código da instrução e o registrador a ser usado
  - Deslocamento para o caso de desvio (em complemento de 2)
- Operação
  - O registrador **Ri** é decrementado de uma unidade
  - Se o resultado não for zero, efetua o desvio
  - Se for zero, passa para a instrução seguinte

Controle de Laço (para quê serve?)

- Facilitar a implementação de laços controlados por um número inteiro
- Exemplo:

```
unsigned short n,k;
n = 5;
k = 0;
do {
        k += n;
} while(--n != 0);
```

```
k:      dw      0

        org     0
        mov     #5,r0
        clr     k
while:
        add     n,k
        sob     r0,while
        hlt
```

Exercício – 6

- Implementar, em *assembly* do CESAR, o programa “C” abaixo
- Utilizar a passagem de parâmetros pela pilha
- Utilizar a instrução SOB para o controle do laço DO-WHILE da função
- Sugestão: utilize a variável “total” como o “R0”

```
unsigned short vetor[5];

void main() {
    unsigned short x;
    x = soma(vetor, 5);
}

unsigned short soma(unsigned short *p, unsigned short n) {
    unsigned short total;
    total = 0;
    do {
        total += *p++;
    } while(--n != 0);
    return total;
}
```

CESAR

Conjunto de Instruções

Parte 2

Prof. Sérgio L. Cechin