

**Universidade de São Paulo**  
**Instituto de Ciências Matemáticas e de Computação**  
**Departamento de Sistemas de Computação**

**Processador ICMC**

**Guia de Referência**

por

**Paulo Sérgio Lopes de Souza**

**Eduardo do Valle Simões**

**Versão 1.0 – r2**  
**Março de 2013**



## 5. Registradores

| Nome           | Qtde | Finalidade                         |
|----------------|------|------------------------------------|
| R <sub>n</sub> | 0-7  | Registradores de propósito geral   |
| FR             | 1    | Flag register *                    |
| SP             | 1    | Ponteiro da pilha                  |
| PC             | 1    | Contador de programa               |
| IR (interno)   | 1    | Registrador de instruções          |
| MAR (interno)  | 1    | Registrador de endereço de memória |

\* Disposição dos bits no Flag Register (FR) - Bits menos significativos do registrador de 16 bits:  
<...|stackunderflow|stackoverflow|DivByZero|ArithmeticOverflow|carry|zero|equal|lesser|greater>

## 6. Formatos de Instrução

Todas as instruções têm 16 ou 32 bits.

Nomes dos campos:

opcode = código da operação (6 bits)

rx, ry, rz = registradores (3 bits para cada registrador)

c = uso do bit de carry (1 bit)

extended opcode (extopc)= 1, 3 ou 4 bits

Formatos de instrução

Instruções Lógicas e Aritméticas

Instruções de Entrada/Saída

|        |        |        |        |       |
|--------|--------|--------|--------|-------|
| 6 bits | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode | rx     | ry     | rz     | c     |

Instruções de Carga e Armazenamento de Dados (direto e imediato)

|                               |        |        |
|-------------------------------|--------|--------|
| 6 bits                        | 3 bits | 7 bits |
| opcode                        | rx     |        |
| Endereço (END) ou Número (NR) |        |        |

Instruções de Carga e Armazenado (indexado via registrador)

Instruções de Deslocamento e Rotação

Instruções de Movimentação de Dados

|        |        |           |             |
|--------|--------|-----------|-------------|
| 6 bits | 3 bits | 3 bits    | 4 bits      |
| opcode | rx     | ry/extopc | Núm./extopc |

Instruções de Controle de Desvio

Instruções para Chamadas de Procedimentos

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode         | extopc |        |
| Endereço (END) |        |        |

Instruções para Manipular Pilha  
Instruções para Incremento/Decremento de Registradores

|        |        |             |        |
|--------|--------|-------------|--------|
| 6 bits | 3 bits | 1 bit       | 6 bits |
| opcode | rx     | rx/fr/++/-- |        |

Instruções de Controle

|        |           |        |
|--------|-----------|--------|
| 6 bits | 1 bit     | 9 bits |
| opcode | set/clear |        |

## 7. Modos de Endereçamento

Direto – operando Endereço (END) permite acessar diretamente a posição de memória.

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 3 bits | 7 bits |
| opcode         | rx     |        |
| Endereço (END) |        |        |

Imediato – operando Número (NR) permite acessar imediatamente a constante especificada na própria instrução.

|             |        |        |
|-------------|--------|--------|
| 6 bits      | 3 bits | 7 bits |
| opcode      | rx     |        |
| Número (NR) |        |        |

Indexado via Registrador – operando é o número de um registrador (ry) que permite acessar a memória via conteúdo do registrador.

|        |        |        |        |
|--------|--------|--------|--------|
| 6 bits | 3 bits | 3 bits | 4 bits |
| opcode | rx     | ry     |        |

Registrador – operando é o número de um registrador (rx ou ry) que permite o acesso ao conteúdo do mesmo.

|        |        |        |        |
|--------|--------|--------|--------|
| 6 bits | 3 bits | 3 bits | 4 bits |
| opcode | rx     | ry     |        |

## 8. Instruções

### Instruções Aritméticas e Lógicas

Soma o conteúdo de ry com o conteúdo de rz e armazena o resultado em rx.

add rx, ry, rz                      rx <= ry+rz                      100000 | rx | ry | rz | 0

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x20) | rx     | ry     | rz     | c (0) |

Soma o conteúdo de ry com o conteúdo de rz e com 1 do bit de carry, armazenando o resultado em rx.

addc rx, ry, rz                      rx <= ry+rz+c                      100000 | rx | ry | rz | 1

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x20) | rx     | ry     | rz     | c (1) |

Subtrai de ry do conteúdo de rz e armazena o resultado em rx.

sub rx, ry, rz

rx <= ry-rz

100001 | rx | ry | rz | 0

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x21) | rx     | ry     | rz     | c (0) |

Subtrai de ry o conteúdo de rz somado com 1 do bit de carry, armazenando o resultado em rx.

subc rx, ry, rz

rx <= ry-rz+c

100001 | rx | ry | rz | 1

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x21) | rx     | ry     | rz     | c (1) |

Multiplica o conteúdo de ry com rz e armazena o resultado em rx.

mul rx, ry, rz

rx <= ry\*rz

100010 | rx | ry | rz | 0

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x22) | rx     | ry     | rz     | c (0) |

Multiplica o conteúdo de ry com rz, soma 1 do bit de carry, armazenando o resultado em rx.

mulc rx, ry, rz

rx <= ry\*rz+c

100010 | rx | ry | rz | 1

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x22) | rx     | ry     | rz     | c (1) |

Divide o conteúdo de ry por rz e armazena o resultado em rx.

div rx, ry, rz

rx <= ry/rz

100011 | rx | ry | rz | 0

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x23) | rx     | ry     | rz     | c (0) |

Divide o conteúdo de ry por rz, soma 1 do bit de carry, armazenando o resultado em rx.

divc rx, ry, rz

rx <= ry/rz+c

100011 | rx | ry | rz | 1

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x23) | rx     | ry     | rz     | c (1) |

Incrementa o conteúdo de rx, armazenando o resultado em rx.

inc rx

rx <= rx + 1

100100 | rx | 0 | xxxxxx

|               |        |        |        |
|---------------|--------|--------|--------|
| 6 bits        | 3 bits | 1 bit  | 6 bits |
| opcode (0x24) | rx     | ++ (0) |        |

Decrementa o conteúdo de rx, armazenando o resultado em rx.

dec rx

rx <= rx -1

100100 | RX | 1 | xxxxxx

|               |        |        |        |
|---------------|--------|--------|--------|
| 6 bits        | 3 bits | 1 bit  | 6 bits |
| opcode (0x24) | rx     | -- (1) |        |

Calcula o resto da divisão do conteúdo de ry por rz, atribuindo o resultado para rx.

mod rx, ry, rz

rx <= ry mod rz

100101 | rx | ry | rz | x

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x25) | rx     | ry     | rz     |       |

Faz a rotação à esquerda de rx, em n posições, armazenando o resultado em rx.

rotl rx, n

rotate to left

010000 | rx | 10x | nnnn

|               |        |        |        |
|---------------|--------|--------|--------|
| 6 bits        | 3 bits | 3 bits | 4 bits |
| opcode (0x10) | rx     | 10x    | nnnn   |

Faz a rotação à direita de rx, em n posições, armazenando o resultado em rx.

rotr rx, n

rotate to right

010000 | rx | 11x | nnnn

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x10) | rx     | 11x    | nnnn   |

Faz o deslocamento à esquerda de rx, em n posições, armazenando o resultado em rx.

Preenche os bits com 0 (zero).

shifto rx, n

shift to left (fill 0)

010000 | rx | 000 | nnnn

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x10) | rx     | 000    | nnnn   |

Faz o deslocamento à esquerda de rx, em n posições, armazenando o resultado em rx.

Preenche os bits com 1 (um).

shifto1 rx, n

shift to left (fill 1)

010000 | rx | 001 | nnnn

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x10) | rx     | 001    | nnnn   |

Faz o deslocamento à direita de rx, em n posições, armazenando o resultado em rx. Preenche os bits com 0 (zero).

shiftr0 rx, n

shift to right (fill 0)

010000 | rx | 010 | nnnn

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x10) | rx     | 010    | nnnn   |

Faz o deslocamento à direita de rx, em n posições, armazenando o resultado em rx. Preenche os bits com 1 (zero).

shiftr1 rx, n

shift to right (fill 1)

010000 | rx | 011 | nnnn

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x10) | rx     | 011    | nnnn   |

Faz a operação lógica AND sobre cada bit de ry com rz, armazenando o resultado em rx.

and rx, ry, rz

rx <= ry and rz

010010 | rx | ry | rz | x

| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
|---------------|--------|--------|--------|-------|
| opcode (0x12) | rx     | ry     | rz     |       |

Faz a operação lógica OR sobre cada bit de ry com rz, armazenando o resultado em rx.

or rx, ry, rz

rx <= ry or rz

010011 | rx | ry | rz | x

| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
|---------------|--------|--------|--------|-------|
| opcode (0x13) | rx     | ry     | rz     |       |

Faz a operação lógica XOR sobre cada bit de ry com rz, armazenando o resultado em rx.

xor rx, ry, rz

rx <= ry xor rz

010100 | rx | ry | rz | x

| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
|---------------|--------|--------|--------|-------|
| opcode (0x14) | rx     | ry     | rz     |       |

Faz a operação lógica NOT sobre cada bit de ry, armazenando o resultado em rx.

not rx, ry

rx <= not(ry)

010101 | rx | ry | xxx | x

| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
|---------------|--------|--------|--------|-------|
| opcode (0x15) | rx     | ry     |        |       |

Compara o conteúdo de rx com o conteúdo de ry, setando apropriadamente o bit do registrador fr (Flag Register). Os bits do fr seguem o padrão descrito na seção Registradores deste Guia de Referência.

cmp rx, ry                      fr <= fr or condição                      010110 | rx | ry | xxx | x

| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
|---------------|--------|--------|--------|-------|
| opcode (0x16) | rx     | ry     |        |       |

### **Instruções de Acesso à Memória**

Armazena o conteúdo de rx na memória apontada por END.

store END, rx                      mem[END] <= rx                      110001 | rx | xxx | xxx | x                      END

| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
|---------------|--------|--------|--------|-------|
| opcode (0x31) | rx     |        |        |       |
| END           |        |        |        |       |

Carrega em rx o conteúdo da memória apontada por END.

load rx, END                      rx <= Mem[END]                      110000 | rx | xxx | xxx | x                      END

| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
|---------------|--------|--------|--------|-------|
| opcode (0x30) | rx     |        |        |       |
| END           |        |        |        |       |

Armazena o conteúdo de ry na memória apontada pelo conteúdo de rx.

storei rx, ry                      mem[rx] <= ry                      111101 | rx | ry | xxxx

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x3d) | rx     | ry     |        |

Carrega em rx o conteúdo da memória apontada pelo conteúdo de ry.

loadi rx, ry                      rx <= mem[ry]                      111100 | rx | ry | xxxx

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x3c) | rx     | ry     |        |

### **Movimentação de Dados**

Carrega em rx um valor imediato presente na própria instrução.

loadn rx, #NR                      rx <= NR                      111000 | rx | xxx | xxx | 0                      NR

| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
|---------------|--------|--------|--------|-------|
| opcode (0x38) | rx     |        |        | 0     |
| NR            |        |        |        |       |

Copia em rx o conteúdo de ry.

mov rx, ry                      rx <= ry                      110011 | rx | ry | xxx0

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x33) | rx     | ry     | xxx0   |

Copia em rx o conteúdo de SP.

mov rx, sp                      rx <= sp                      110011 | rx | xxx | xx01

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x33) | rx     |        | xx01   |

Copia em sp o conteúdo de rx.

mov sp, rx

sp <= rx

110011 | rx | xxx | xx11

| 6 bits        | 3 bits | 3 bits | 4 bits |
|---------------|--------|--------|--------|
| opcode (0x33) | rx     |        | xx11   |

### Instruções de Desvio

Se satisfizerem a condição, todas as instruções desviam o fluxo de execução para o endereço apontado por END. A instrução **jmp END** é incondicional. Os seguintes passos são efetuados:

**Se** (condição é satisfeita) **então** // exceção: **jmp END** que é incondicional  
pc <= END

Desvia incondicionalmente para a instrução em END.

jmp END pc <= END (unconditional) 000010 | 0000 | xxxxxx END

| 6 bits         | 4 bits | 6 bits |
|----------------|--------|--------|
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última comparação de rx e ry tenha resultado igualdade.

jeq END pc <= END (Equal) 000010 | 0001 | xxxxxx END

| 6 bits         | 4 bits | 6 bits |
|----------------|--------|--------|
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última comparação resultou que os números não eram iguais.

jne END pc <= END (NotEqual) 000010 | 0010 | xxxxxx END

| 6 bits         | 4 bits | 6 bits |
|----------------|--------|--------|
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última operação aritmética/lógica resultou o número 0 (zero).

jz END pc <= END (Zero) 000010 | 0011 | xxxxxx END

| 6 bits         | 4 bits | 6 bits |
|----------------|--------|--------|
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última operação aritmética/lógica não resultou o número 0 (zero).

jnz END pc <= END (NotZero) 000010 | 0100 | xxxxxx END

| 6 bits         | 4 bits | 6 bits |
|----------------|--------|--------|
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última operação aritmética teve um bit de carry em relação ao par de bits mais significativos dos operandos.

jc END pc <= END (Carry) 000010 | 0101 | xxxxxx END

| 6 bits         | 4 bits | 6 bits |
|----------------|--------|--------|
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |



Desvia para a instrução em END se a última operação aritmética não teve um bit de carry em relação ao par de bits mais significativos dos operandos.

jnc END      pc <= END (NotCarry)      000010 | 0110 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última comparação resultou em rx > ry.

jgr END      pc <= END (Greater)      000010 | 0111 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última comparação resultou em rx < ry.

jle END      pc <= END (Lesser)      000010 | 1000 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última comparação resultou em rx >= ry.

jeg END      pc <= END (EqualorGreater)      000010 | 1001 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última comparação resultou em rx <= ry.

jel END      pc <= END (EqualorLesser)      000010 | 1010 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última operação aritmética resultou um overflow.

jov END      pc <= END (Overflow ULA)      000010 | 1011 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última operação aritmética não resultou um overflow.

jno END      pc <= END (NotOverflow)      000010 | 1100 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | Cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última operação aritmética resultou um número negativo.

jn END      pc <= END (Negative ULA)      000010 | 1101 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

Desvia para a instrução em END se a última operação aritmética resultou em uma divisão por zero.

jdz END      pc <= END (DivbyZero)      000010 | 1110 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x02)  | cond   |        |
| Endereço (END) |        |        |

### Instruções de Chamadas a Procedimentos

Se satisfizerem a condição, todas as instruções guardam o conteúdo de PC na pilha e então desviam o fluxo de execução para o endereço apontado por END. A instrução **call END** é incondicional. Os seguintes passos são efetuados:

**Se** (condição é satisfeita) **então** // exceção: **call END** que é incondicional  
    memória(sp) <= pc  
    pc <= END  
    sp <= sp -1

Desvia fluxo de execução incondicionalmente para END. Salva na pilha o conteúdo de PC.

call END      (unconditional)      000011 | 0000 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última comparação de rx e ry tenha resultado igualdade. Salva na pilha o conteúdo de PC.

ceq END      ( Equal)      000011 | 0001 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última comparação de rx e ry não tenha resultado igualdade. Salva na pilha o conteúdo de PC.

cne END      (NotEqual)      000011 | 0010 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última operação aritmética tenha resultado 0 (zero) . Salva na pilha o conteúdo de PC.

cz END      ( Zero)      000011 | 0011 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última operação aritmética não tenha resultado 0 (zero). Salva na pilha o conteúdo de PC.

cnz END      (NotZero)      000011 | 0100 | xxxxxx      END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última operação aritmética tenha apresentado um bit de carry, considerando os bits mais significativos dos operandos. Salva na pilha o conteúdo de PC.

cc END (Carry) 000011 | 0101 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última operação aritmética não tenha apresentado um bit de carry, considerando os bits mais significativos dos operandos. Salva na pilha o conteúdo de PC.

cnc END (NotCarry) 000011 | 0110 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última comparação entre rx e ry tenha resultado "maior". Salva na pilha o conteúdo de PC.

cgr END (Greater) 000011 | 0111 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última comparação entre rx e ry tenha resultado "menor". Salva na pilha o conteúdo de PC.

cle END (Lesser) 000011 | 1000 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última comparação entre rx e ry tenha resultado "maior ou igual". Salva na pilha o conteúdo de PC.

ceg END (EqualorGreater) 000011 | 1001 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última comparação entre rx e ry tenha resultado "menor ou igual". Salva na pilha o conteúdo de PC.

cel END (EqualorLesser) 000011 | 1010 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última operação aritmética tenha resultado um overflow. Salva na pilha o conteúdo de PC.

cov END (Overflow ULA) 000011 | 1011 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última operação aritmética não tenha resultado um overflow. Salva na pilha o conteúdo de PC.

cno END (NotOverflow) 000011 | 1100 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode         | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última operação aritmética tenha resultado um número negativo. Salva na pilha o conteúdo de PC.

cn END (Negative ULA) 000011 | 1101 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Desvia fluxo de execução para END caso a última operação aritmética tenha resultado uma divisão por zero. Salva na pilha o conteúdo de PC.

cdz END (DivbyZero) 000011 | 1110 | xxxxxx END

|                |        |        |
|----------------|--------|--------|
| 6 bits         | 4 bits | 6 bits |
| opcode (0x03)  | cond   |        |
| Endereço (END) |        |        |

Retorna o fluxo de execução, ao final de um procedimento. Recupera o conteúdo de PC da pilha, permitindo o retorno no final da execução de um procedimento. Ao usar esta instrução, deve-se sempre incrementar pc depois, pois o mesmo estará apontando para o END da rotina que fez a chamada (call). Os passos executados são:

rts  
 $sp \leq sp + 1$   
 $pc \leq \text{memória}(sp)$   
 $pc \leq pc + 1$ 
000100 | xxxxxxxxxx

|               |        |        |        |
|---------------|--------|--------|--------|
| 6 bits        | 3 bits | 3 bits | 4 bits |
| opcode (0x04) |        |        |        |

### Instruções de Entrada e Saída

Permite a entrada de um caracter vindo do teclado<sup>1</sup>

inchar rx  $rx \leq "00000000" \&key$  110101 | rx | xxx | xxx | x

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x35) | rx     |        |        |       |

Permite escrever um caracter existente em rx (código ASCII do caracter está em rx) na posição ry do vídeo.

outchar rx, ry  $\text{vídeo}(ry) \leq \text{ASCII}(rx)$  110010 | rx | ry | xxx | x

|               |        |        |        |       |
|---------------|--------|--------|--------|-------|
| 6 bits        | 3 bits | 3 bits | 3 bits | 1 bit |
| opcode (0x32) | rx     | ry     |        |       |

<sup>1</sup> Para utilizar a instrução *inchar*, implemente uma espera ocupada (*loop*) verificando se o valor do registrador rx é diferente de 255. Exemplo:

```
loadn r1, #255           ;Carrega valor 255 em r1 para futura comparacao

ler_tecla:               ;Inicio do loop
    inchar r0             ;Ler entrada do teclado
    cmp r0, r1            ;Compara a entrada com 255
    jeq ler_tecla         ;Se o valor continuar sendo 255 volta ao inicio do loop
    outchar r0, r7        ;Se o valor da entrada for diferente de 255 imprime na tela
```

### Instruções para Manipulação da Pilha

Insere o conteúdo de rx na pilha. Decrementa sp após inserção.

push rx                      memória(sp) <= rx                      000101 | rx | 0 | xxxxxx  
sp <= sp - 1

|               |        |       |        |
|---------------|--------|-------|--------|
| 6 bits        | 3 bits | 1 bit | 6 bits |
| opcode (0x05) | rx     | 0     |        |

Insere o conteúdo de fr na pilha. Decrementa sp após inserção.

push fr                      memória(sp) <= fr                      000101 | xxx | 1 | xxxxxx  
sp <= sp - 1

|               |        |       |        |
|---------------|--------|-------|--------|
| 6 bits        | 3 bits | 1 bit | 6 bits |
| opcode (0x05) | rx     | 1     |        |

Retira o conteúdo do topo da pilha e o atribui para rx. Incrementa sp antes da operação.

pop rx                      sp <= sp + 1                      000110 | rx | 0 | xxxxxx  
memória(sp) => rx

|               |        |       |        |
|---------------|--------|-------|--------|
| 6 bits        | 3 bits | 1 bit | 6 bits |
| opcode (0x06) | rx     | 0     |        |

Retira o conteúdo do topo da pilha e o atribui para fr. Incrementa sp antes da operação.

pop fr                      sp <= sp + 1                      000110 | xxx | 1 | xxxxxx  
memória(sp) => fr

|               |        |       |        |
|---------------|--------|-------|--------|
| 6 bits        | 3 bits | 1 bit | 6 bits |
| opcode (0x06) | rx     | 1     |        |

### Instruções de Controle

Atribui 0 (zero) ao bit de carry em fr.

clearc                      c<-0                      001000 | 0 | xxxxxxxxx

|               |       |        |
|---------------|-------|--------|
| 6 bits        | 1 bit | 9 bits |
| opcode (0x08) | 0     |        |

Atribui 1 (um) ao bit de carry em fr.

setc                      c<-1                      001000 | 1 | xxxxxxxxx

|               |       |        |
|---------------|-------|--------|
| 6 bits        | 1 bit | 9 bits |
| opcode (0x08) | 1     |        |

Termina a execução do programa.

halt                      stop execution                      001111 | x | xxxxxxxxx

|               |       |        |
|---------------|-------|--------|
| 6 bits        | 1 bit | 9 bits |
| opcode (0x0f) |       |        |

Troca o modo de execução para clock manual, i.e., insere um *break-point* no código.

breakp                      Insert break point                      001110 | xxxxxxxxx

|               |       |        |
|---------------|-------|--------|
| 6 bits        | 1 bit | 9 bits |
| opcode (0x0e) |       |        |

Operação nula.

nop                      no operation                      000000 | x | xxxxxxxxx

|               |       |        |
|---------------|-------|--------|
| 6 bits        | 1 bit | 9 bits |
| opcode (0x00) |       |        |

## 9. Diretivas para o Montador

O montador não é *case sensitive*.

***;* (ponto e vírgula)**

Inicia um comentário na linha.

Exemplo de uso:

***;* este é um comentário no código fonte.**

***label:***

Rótulo que indica um endereço na memória. Pode indicar uma posição para o início de um dado ou de uma instrução.

Exemplos de uso:

***inicio\_loop:***

***main:***

***var #quantidade de posições***

Reserva <quantidade de posições> na memória. Está associada a um rótulo, normalmente. Cada posição tem 16 bits.

Exemplo de uso:

***word: var #20*** ; reserva 20 posições de 16 bits na memória sendo que o rótulo word aponta para a primeira posição.

***static <label> + #<pos>, #<num>***

Preenche a memória na posição apontada por <label> mais <pos> deslocamentos, com o número <num>, o qual deve ser um inteiro sem sinal. As posições de memória já devem ter sido alocadas previamente. Cada posição de memória tem 16 bits.

Exemplo de uso:

***static word + #0, #5*** ; preenche a primeira posição apontada pelo rótulo word com o número 5.

***static word + #1, #8*** ; preenche a segunda posição apontada pelo rótulo word com o número 8.

***string "caracteres"***

Aloca a quantidade necessária de posições de memória para armazenar a string em "***caracteres***" e mais uma posição para o '\0'. Preenche as posições de memória alocadas com os códigos ASCII da string. Cada posição de memória tem 16 bits.

Exemplo de uso:

***mensagem: string "Organizacao"*** ; aloca 12 posições de memória ao todo: 11 para todas as letras e mais 1 para \0 no final. O rótulo mensagem aponta para o primeiro caracter da string.