Arquitetura de Computadores

Prof. Fábio M. Costa
Instituto de Informática – UFG
1S/2005

ISA – Parte II: Arquiteturas-Exemplo Simuladores e Máquinas Reais

Objetivos Gerais

- Demonstrar os conceitos genéricos de arquiteturas de computadores (nível ISA) através do uso de algumas arquiteturas-exemplo representativas
  - computadores hipotéticos: simuladores
  - computadores reais: Intel x86
- Introduzir o tópico de programação em baixo nível, como ferramenta para o entendimento de arquiteturas de computadores

Arquiteturas estudadas

- Computadores hipotéticos (simuladores com finalidade didática):
  - Neander
  - Ahmes
  - Ramses
  - Cesar

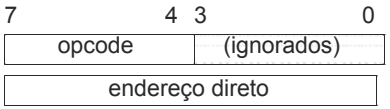
  Weber, Raul F. "**Fundamentos de Arquitetura de Computadores**". Série de Livros Didáticos do Instituto de Informática da UFRGS, Número 8, 2a. Edição. Sagra-Luzzatto, 2001
- Computadores reais:
  - IBM PC – família Intel x86

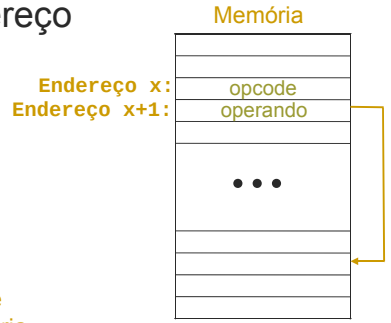
O Computador Neander

- Largura de dados e endereços: 8 bits
  - i.e., comprimento de palavra de 8 bits
- Dados representados em complemento de 2
- 1 acumulador de 8 bits (AC)
  - arquitetura de acumulador
- 1 apontador de programa de 8 bits (PC)
- 1 registrador de estado (flags) com 2 códigos de condição:
  - negativo (N) e zero (Z)

Neander: Formato das Instruções

- opcode: 8 bits
- operando: 8 bits seguintes
  - especifica um endereço
  - modo direto





Obs.: o segundo byte (operando) ocorre apenas em instruções de manipulação de dados, i.e., que fazem referência à memória

Neander: Organização da Memória - Convenções

- 256 palavras de 8 bits: 256 bytes
- Primeiros 128 bytes (metade inferior):
  - Código do programa
- 128 bytes seguintes (metade superior):
  - Dados do programa
- Apenas convenções: dados e código podem ser localizados em qualquer lugar
  - Mas um não pode invadir a área do outro

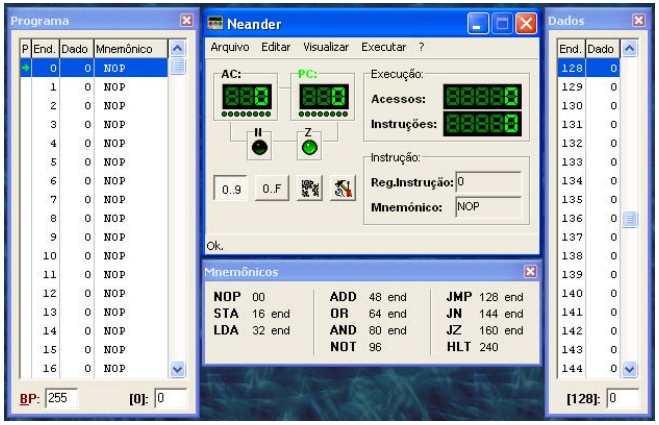
Neander: Conjunto de Instruções

Código	Instrução	Significado
0000	NOP	nenhuma operação
0001	STA end	MEM(end) ← AC
0010	LDA end	AC ← MEM(end)
0011	ADD end	AC ← MEM(end) + AC
0100	OR end	AC ← MEM(end) **OR** AC (“ou” bit-a-bit)
0101	AND end	AC ← MEM(end) **AND** AC (“e” bit-a-bit)
0110	NOT	AC ← **NOT** AC (complemento de 1)
1000	JMP end	PC ← end (desvio incondicional)
1001	JN end	IF N=1 THEN PC ← end
1010	JZ end	IF Z=1 THEN PC ← end
1111	HLT	término da execução (halt)

Neander: Códigos de Condição (Flags)

- Gerados pela Unidade Lógico-Aritimética após as seguintes operações:
  - ADD, NOT, AND, OR e LDA
- Testados pelas instruções JN e JZ
- N (negativo): indica o sinal do resultado
  - 1: resultado é negativo
  - 0: resultado é positivo
- Z (zero): indica se o resultado é igual a zero
  - 1: resultado é igual a zero
  - 0: resultado é diferente de zero

Neander: Simulador



O Computadore Ahmes

- Largura de dados e endereços: 8 bits
- Dados representados em complemento de 2
- 1 acumulador de 8 bits (AC)
- 1 apontador de programa de 8 bits (PC)
- 1 registrador de estado (flags) com 5 códigos de condição:
  - negativo (N) e zero (Z)
  - carry out (C), borrow out (B), overflow (V)
- Compatível com o Neander

Ahmes: Conjunto de Instruções

Código	Instrução	Significado
0000 xxxx	NOP	nenhuma operação
0001 xxxx	STA end	MEM(end) ← AC
0010 xxxx	LDA end	AC ← MEM(end)
0011 xxxx	ADD end	AC ← AC + MEM(end)
0100 xxxx	OR end	AC ← AC **OR** MEM(end) *("ou" bit-a-bit)*
0101 xxxx	AND end	AC ← AC **AND** MEM(end) *("e" bit-a-bit)*
0110 xxxx	NOT	AC ← **NOT** AC (complemento de 1)
0111 xxxx	SUB end	AC ← AC - MEM(end)
1000 xxxx	JMP end	PC ← end (desvio incondicional)
1001 00xx	JN end	IF N=1 THEN PC ← end
1001 01xx	JP end	IF N=0 THEN PC ← end
1001 10xx	JV end	IF V=1 THEN PC ← end
1001 11xx	JNV end	IF V=0 THEN PC ← end

Ahmes: Conjunto de Instruções (2)

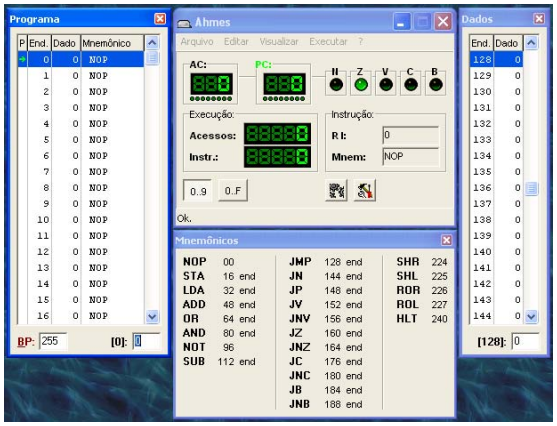
Código	Instrução	Significado
1010 00xx	JZ end	IF Z=1 THEN PC ← end
1010 01xx	JNZ end	IF Z=0 THEN PC ← end
1011 00xx	JC end	IF C=1 THEN PC ← end
1011 01xx	JNC end	IF Z=0 THEN PC ← end
1011 10xx	JB end	IF B=1 THEN PC ← end
1011 11xx	JNB end	IF Z=0 THEN PC ← end
1110 xx00	SHR	C ← AC(0); AC(i-1) ← AC(i); AC(7) ← 0
1110 xx01	SHL	C ← AC(7); AC(i) ← AC(i-1); AC(0) ← 0
1110 xx10	ROR	C ← AC(0); AC(i-1) ← AC(i); AC(7) ← C
1110 xx11	ROL	C ← AC(7); AC(i) ← AC(i-1); AC(0) ← C
1111 xxxx	HLT	término da execução (halt)

Instruções que afetam os Códigos de Condição

Instrução	Códigos alterados
NOP	nenhum
STA end	nenhum
LDA end	N, Z
ADD end	N, Z, V, C
SUB end	N, Z, V, B
OR end	N, Z
AND end	N, Z
NOT	N, Z

Instrução	Códigos alterados
JMP end	nenhum
Jxx end	nenhum
SHR	N, Z, C
SHL	N, Z, C
ROR	N, Z, C
ROL	N, Z, C
HLT	nenhum

Ahmes: Simulador



O Computadore Ramses

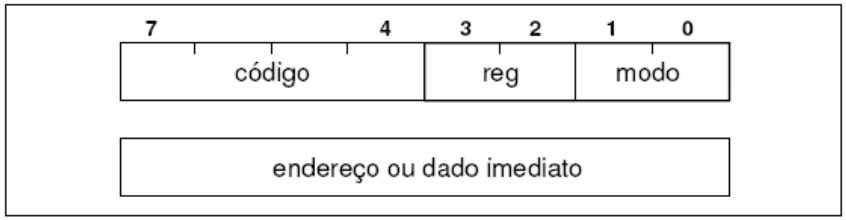
- Incorpora os recursos do NEANDER
- Acrescenta
  - Outros modos de endereçamento (o NEANDER só tem o direto)
  - Novos registradores
  - Novos bits no registrador de estado (códigos de condição)
  - Novas instruções (mais poderosas)
- É compatível com o código escrito para o NEANDER (é capaz de rodar os programas do NEANDER)

Ramses: Características

- Largura de dados e endereços = 8 bits
  - Tamanho da memória = 256 bytes
- Dados representados em complemento de 2
  - Isso tem efeito nos cálculos que envolvem a ULA
- Registradores (em negrito os adicionais ao Neander)
  - De uso geral: A e **B** (8 bits)
  - Registrador de índice: **X** (8 bits)
  - Apontador de programa (PC)
  - Registrador de estado
    - Códigos de condição N, Z e **C**

Ramses: Formato de Instruções

- Instruções são representadas por 1 ou 2 bytes
  - Semelhante ao Neander

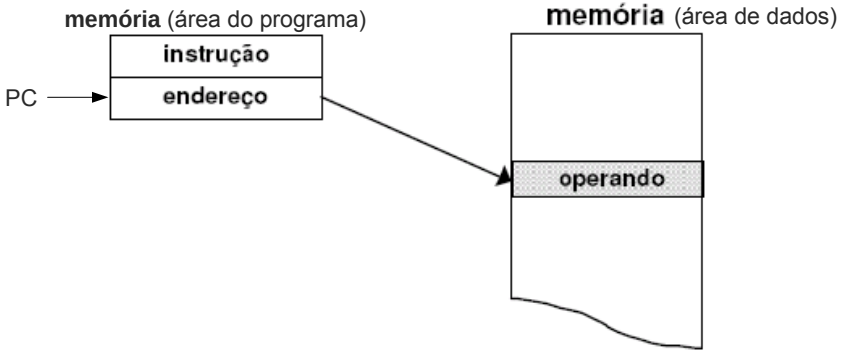


Ramses: Modos de Endereçamento

- **00 – Direto**: endereço do operando no segundo byte (como no Neander)
- **01 – Indireto**: endereço do endereço do operando no segundo byte
- **10 – Imediato**: o próprio operando no segundo byte
- **11 – Indexado**: o segundo byte contém um endereço (base), que é somado ao conteúdo do registrador RX (índice) para compor o endereço do operando

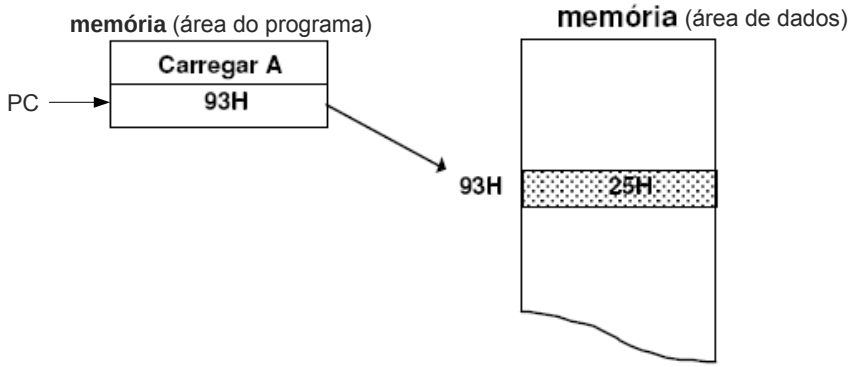
Ramses: Modo Direto

A = MEM(Palavra imediata) = MEM(MEM(PC))



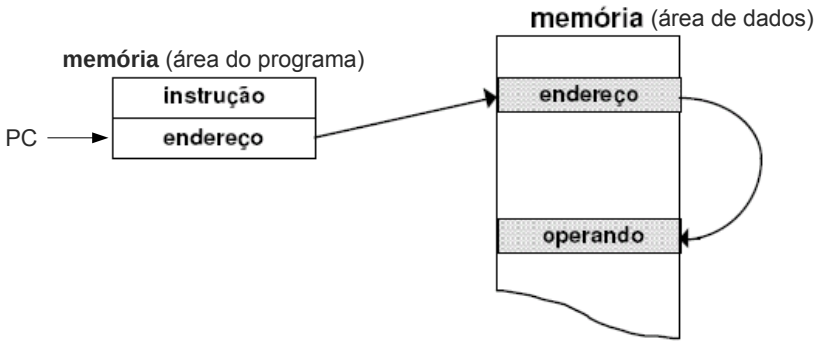
Ramses: Modo Direto - Exemplo

A = MEM((MEM(PC)) = MEM(93H) = 25H



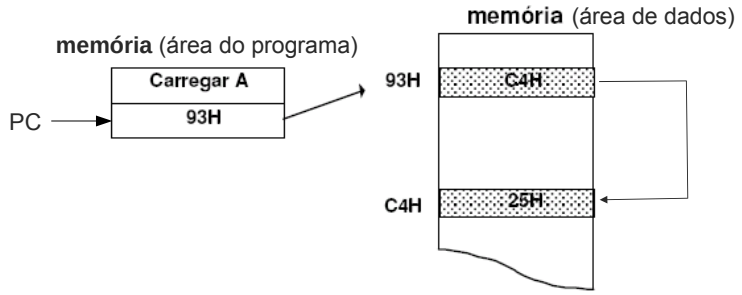
Ramses: Modo Indireto

A = MEM(MEM(Palavra imediata)) = MEM(MEM(MEM(PC)))



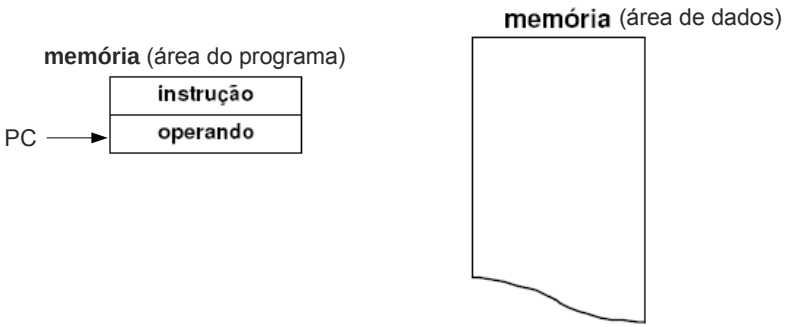
Ramses: Modo Indireto - Exemplo

A = MEM(MEM(MEM(PC))) = MEM(MEM(93H)) = MEM(C4H) = 25H



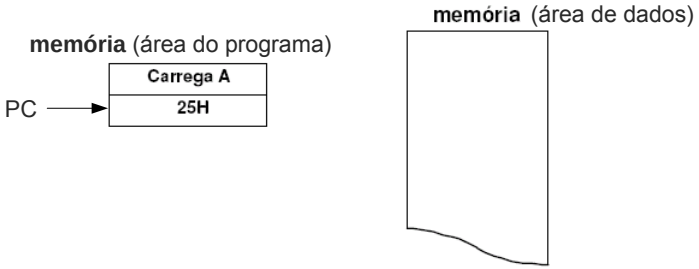
Ramses: Modo Imediato

A = Palavra imediata = MEM(PC)



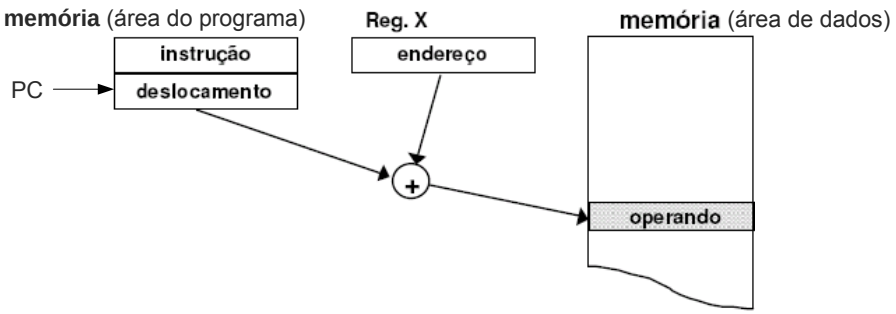
Ramses: Modo Imediato - Exemplo

A = MEM(PC) = 25H



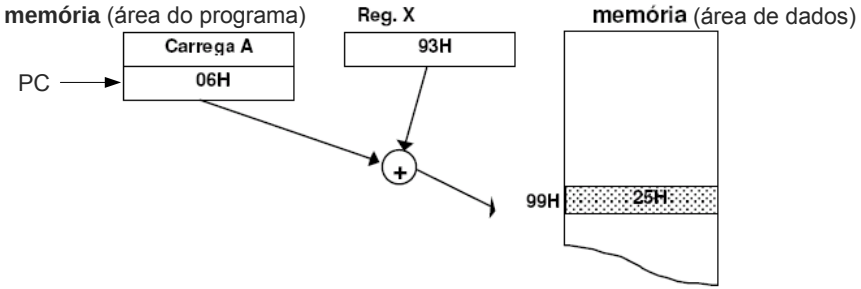
Ramses: Modo Indexado

A = MEM(X+Palavra imediata) = MEM (X + MEM(PC))



Ramses: Modo Indexado - Exemplo

A = MEM(X + MEM(PC) = MEM (93H + 06H) = MEM(99H) = 25H



Ramses: Códigos dos Registradores

- 00: Registrador A (RA)
- 01: Registrador B (RB)
- 10: Registrador de índice (RX)
- 11: indefinido

Ramses: Representação simbólica dos modos de endereçamento

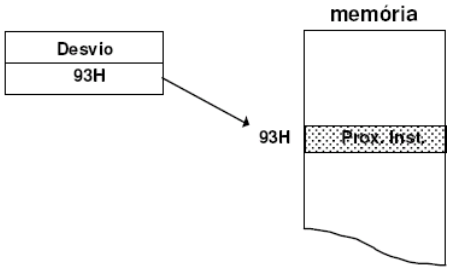
- Direto: end = n
  - Ex1: NOME
  - Ex2: 93H
- Indireto: end = n,I
  - Ex1: NOME,I
  - Ex2: 93H,I
- Imediato: end = #n
  - Ex1: #NOME
  - Ex2: #25H
- Indexado: end = n,X
  - Ex1: NOME,X
  - Ex2: 93H,X

Ramses: Desvios e Modos de Endereçamento

- Os seguintes modos de endereçamento podem ser usados para especificar o endereço alvo de desvios
  - direto
  - indireto
  - indexado
- Modo imediato: não é válido – seria um desvio para a própria instrução

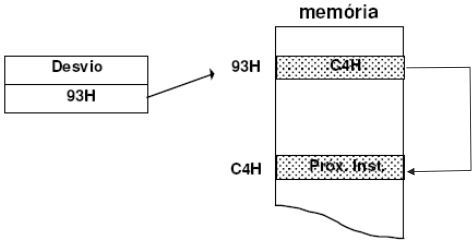
Ramses: Desvios no Modo Direto

- Endereço de desvio:
  - Palavra imediata = MEM(PC)
- O que é equivalente a:
  - Próxima instrução = MEM(MEM(PC))



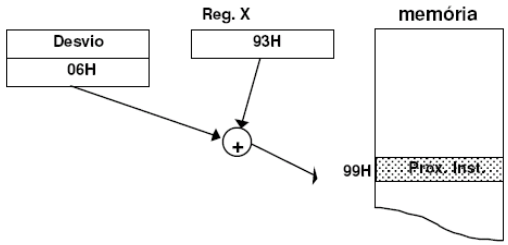
Ramses: Desvios no Modo Indireto

- Endereço de desvio
  - MEM(MEM(PC))
- Próxima instrução
  - MEM(MEM(MEM(PC)))



Ramses: Desvios no Modo Indexado

- Endereço de desvio
  - X + MEM(PC)
- Próxima Instrução
  - MEM(X + MEM(PC))



Ramses: Conjunto de Instruções (1)

- Acesso à memória
  - STR r end ; MEM(end) ← r
  - LDR r end ; r ← MEM(end)
- Aritiméticas e lógicas
  - ADD r end ; r ← MEM(end) + r
  - SUB r end
  - OR r end
  - AND r end
  - NOT r ; r ← NOT(r) – neg. bit-a-bit
  - NEG r ; r ← -r

Ramses: Conjunto de Instruções (2)

- Instruções de controle de fluxo
  - JMP end ; PC ← end
  - JN end ; IF N=1 THEN PC ← end
  - JZ end ; IF Z=1 THEN PC ← end
  - JC end ; IF C=1 THEN PC ← end
  - JSR end ; MEM(end) ← PC; PC ← end+1
    - Desvio para sub-rotina

Ramses: Conjunto de Instruções (3)

- Instrução de deslocamento de bits
  - SHR r ; r ← r/2
- Considerar as limitações impostas pelo formato de instrução adotado

Ramses: Códigos de Condição

Instrução	Códigos de Condição
0010 – LDR	N, Z
0011 – ADD	N, Z, C
0100 – OR	N, Z
0101 – AND	N, Z
0110 – NOT	N, Z
0111 – SUB	N, Z, C Obs.: carry=1: não houve borrow; carry=0: houve borrow
1101 – NEG	N, Z, C
1110 – SHR	N, Z, C Obs.: carry = bit menos significativo (deslocado para fora do registrador)

Exercício

- Ex. 3 (Weber):
  - Inicialização de uma área de memória com n posições
    - end. 128: número de posições
    - end. 129: posição inicial

O Montador Daedalus

- Uso de mnemônicos em lugar dos códigos de operação binários
  - evita ter que montar os códigos manualmente
- Permite o uso de rótulos em lugar de endereços
  - dados
  - instruções (para uso em desvios)

Ramses: Subrotinas (procedimentos)

- Chamada: JSR end
  - *Jump to SubRoutine*
- Retorno: JMP end,I
  - desvio com endereço indireto
- O endereço de retorno é armazenado na primeira palavra da subrotina
- Execução de JSR end

```
MEM(end) ← PC  ;armazena o endereço de retorno
PC ← end + 1       ;desvio para a primeira instrução
                   ;da subrotina
```

Exemplo:

- Uma subrotina para calcular o complemento de 2 de um número

```
End.  Instrução
60 NOP       ;aqui ficará o end. de retorno
61 NOT   B
62 ADD   B #1
64 JMP   60,I    ;retorno da subrotina
```

- Chamada da subrotina: JSR 60

Exemplos de chamadas

```
...
10 JSR 60    ;guarda end. 12 na pos. 60 da mem.
12   ...     ;instrução a executar após o retorno
...
16 JSR 60    ;guarda end. 18 na pos. 60 da mem.
18 ...       ; instrução a executar após o retorno
...
60 NOP   ;posição onde será guardado o end. retorno
61 xxx   ;posição onde fica a primeira instr. da rotina
```

- Em ambos os casos, após JSR, PC = 61

Ramses: Subrotinas - Limitações

- Não permite recursividade
  - Por que???
- Não permite reentrância
  - i.e., mesmo código compartilhado por vários programas
    - Por que???
- Mas permite chamadas aninhadas
  - uma subrotina que chama outra, que chama outra, ...

Ramses: Subrotinas – Passagem de Parâmetros

- Duas modalidades:
  - Por valor – passa-se o valor da variável
  - Por nome – passa-se o endereço da variável
- Mecanismos:
  - via registrador
  - via memória, em endereços pre-estabelecidos
  - via memória, nas posições seguintes à chamada
  - via memória, em endereço apontado por RX (modo indexado)

Passagem por Registrador

- **Programa principal:**

```
LDR A primeiro_operando
LDR B segundo_operando
JSR    multiplica
STR A resultado
```

- **Subrotina:**

```
multiplica: NOP
            STR A op1
            STR B op2
            <multiplicação>
            LDR A result
            JMP multiplica, I
```

Passagem por Memória

- **Programa principal:**

```
LDR A primeiro_operando
STR A param1
LDR A segundo_operando
STR A param2
JSR      multiplica
LDR A param3
STR A resultado
```

- **Subrotina**

```
NOP
LDR A param1
STR A op1
LDR A param2
STR A op2
<multiplicação>
LDR A result
STR A param3
JMP      multiplica,I
```

Passagem por Memória (na área do programa)

- **Programa principal:**

```
JSR multiplica
<valor do 1o. operando>
<valor do 2o. operando>
<endereço do resultado>
<<instrução seguinte>>
...
```

- **Subrotina:**

```
multiplica: NOP
            LDR A multiplica, I
            STR A param1
            LDR A multiplica
            ADD A #1
            STR A multiplica

            LDR A multiplica, I
            STR A param2
            LDR A multiplica
            ADD A #1
            STR A multiplica

            <multiplicação>

            LDR A multiplica, I
            STR A param3
            LDR B result
            STR B param3, I
            LDR A multiplica
            ADD A #1
            STR A multiplica

            JMP multiplica, I
```

Outra opção para acesso aos parâmetros: Modo Indexado

- **Subrotina**

```
multiplica: NOP
            LDR X multiplica ;endereço do primeiro operando
            LDR A 0,X        ;valor do primeiro operando
            STR A param1

            LDR A 1,X        ;valor do segundo operando
            STR A param2

            <multiplicação>

            STR X end_ret
            LDR X 2,X        ;endereço do terceiro op. (retorno)
            LDR A result     ;resultado da rotina
            STR A 0,X        ;salva result. no endereço do 3o. parâmetro

            LDR X end_ret
            ADD X #3     ;atualiza o endereço de retorno
            STR X multiplica

            JMP multiplica, I    ;retorna da subrotina
```

O Computador Hipotético Cesar: Visão geral da arquitetura

- Características gerais
- Organização de memória
- Modos de endereçamento
- Manipulação da Pilha
- Conjunto de instruções
- Entrada e Saída
- Subrotinas

O Computador Hipotético Cesar: Visão geral da arquitetura

- Modos de endereçamento: 8
- Registrador de estado com 4 códigos de condição:
  - N, Z, C, V
- Processamento de pilha
- Incompatível com o NEANDER e RAMSES
- Dados e endereços com 16 bits
- Representação dos dados em complemento de 2
- Instruções com 0, 1 ou 2 endereços (operandos)
- Registradores de uso geral
  - R0 até R5: sem função específica
  - R6: apontador de pilha (SP – Stack Pointer)
  - R7: apontador de programa (PC – Program Counter)

Cesar: Organização de memória

- Organizada em bytes
- Razões
  - Acomodar instruções com 1 byte
  - Usada em processadores comerciais
  - Em geral, dados mais largos possuem tamanhos múltiplos de 8 bits
- Formas de armazenamento
  - Big endian ou high/low: Motorola, CESAR
  - Little endian ou low/high: Intel

Cesar: Modos de Endereçamento

Código	Nome	Simbólico	Operação
000	Reg ou Registrador	Ri	Operando := Ri
001	Reg pós-incrementado	(Ri)+	Operando := MEM(Ri) Ri := Ri + 2
010	Reg pré-decrementado	-(Ri)	Ri := Ri – 2 Operando := MEM(Ri)
011	Indexado	ddd(Ri)	Operando := MEM(ddd+Ri) (soma em comp. de dois)
100	Reg **Indireto**	(Ri)	Operando := **MEM**(Ri)
101	Pós-incrementado **Indireto**	((Ri)+)	Operando := **MEM**(MEM(Ri)) Ri := Ri + 2
110	Pré-decrementado **Indireto**	(-(Ri))	Ri := Ri – 2 Operando := **MEM**(MEM(Ri))
111	Indexado **Indireto**	(ddd(Ri))	Operando := **MEM**(MEM(ddd+Ri)) (soma em comp. de dois)

- **Obs.:** todos os modos (exceto 000) podem ser usados tanto para acesso a dados quanto para indicar endereços de desvio
  - para endereço de desvio, o modo 000 (registrador) não é válido

Cesar: Modos de endereçamento usando o registrador R7 (PC)

- O processador permite livre acesso ao PC (contador de programa).
- Alguns modos de endereçamento comuns podem ser obtidos desta forma:
  - imediato
  - absoluto (ou direto)

Cesar: Modo Imediato

- Uso do registrador R7 (PC) no modo registrador com pós-incremento
- A palavra (2 bytes) seguinte à instrução atual contém o operando imediato
  - Após buscar a instrução, PC aponta para o primeiro byte desta palavra
    - operando: MEM(PC)
  - Após executar o acesso (ao operando imediato)
    - PC ← PC + 2
      - isto é, o endereço da próxima instrução efetiva

Cesar: Modo Direto

- Uso do registrador R7 (PC) no modo pós-incremento indireto
- A palavra seguinte à instrução atual contém o endereço do operando
  - Após a busca da instrução, PC aponta para o endereço do operando
    - operando: MEM(MEM(PC))
  - Após executar o acesso ao operando
    - PC ← PC + 2

Cesar: Manipulação da Pilha do Sistema

- Apontador de topo da pilha (*SP-Stack Pointer*):
  - registrador R6
- Inicializado com 0
- Empilhamento:

```
SP := SP – 2
MEM(SP) := dado
```

- Desempilhamento

```
Dado := MEM(SP)
SP := SP + 2
```

- Cresce em direção aos endereços menores
  - Não existe controle de colisão com o programa ou dados

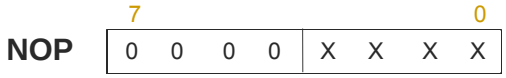
Obs.: não há instruções específicas para manipulação da pilha – usar os modos de registrador pós- e pré-incremento com o registrador R6

Cesar: Conjunto de Instruções

Código (4 bits mais significativos do 1o. byte da instrução)	Categoria de Instruções
0000	NOP
0001 e 0010	Instruções sobre códigos de condição
0011	Instruções de desvio condicional
0100	Instrução de desvio incondicional (JMP)
0101	Instrução de controle de laço (SOB)
0110	Instr. de desvio para subrotina (JSR)
0111	Instr. de retorno de subrotina (RTS)
1000	Instruções de um operando
1001 a 1110	Instruções de dois operandos
1111	Instrução de parada (HLT)

Cesar: Instrução NOP

- Ocupa apenas 1 byte



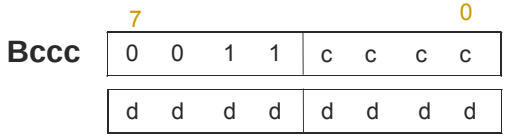
  - os bits 0 a 3 podem assumir qualquer valor

Cesar: Instruções para manipular códigos de condição

- Ocupam apenas 1 byte
- Formato:
  - CCC: 0 0 0 1 | n z v c
    - desliga (zera) os códigos de condição correspondentes aos bits selecionados (n, z, v, c)
  - SCC: 0 0 1 0 | n z v c
    - liga os códigos de condição selecionados

Cesar: Instruções de desvio Condicional

- Ocupam 2 bytes
- Formato geral:



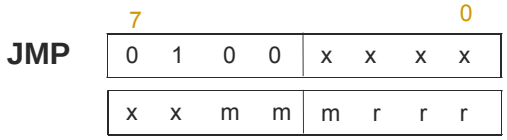
  - **cccc**: condição a ser testada
  - segundo byte: deslocamento relativo ao PC
    - em complemento de 2: faixa PC-128 até PC+127
    - valor do PC utilizado: endereço da próxima instrução (isto é, após o byte de deslocamento)

Cesar: Desvio condicional: Códigos de condição e mnemônicos

Cccc	*mnemônico*	*Condição de desvio*
0000	**BR** (always)	sempre verdadeira
0001	**BNE** (Not Equal)	Z = 0
0010	**BEQ** (EQual)	Z = 1
0011	**BPL** (Plus)	N = 0
0100	**BMI** (Minus)	N = 1
0101	**BVC** (oVerflow Clear)	V = 0
0110	**BVS** (oVerflow Set)	V = 1
0111	**BCC** (Carry Clear)	C = 0
1000	**BCS** (Carry Set)	C = 1
1001	**BGE** (Greater or Equal)	N = V
1010	**BLT** (Less Than)	N < > V
1011	**BGT** (GreaTer)	N = V and Z = 0
1100	**BLE** (Less or Equal)	N ≠ V or Z = 1
1101	**BHI** (HIgher)	C = 0 and Z = 0
1110	**BLS** (Lower or Same)	C = 1 or Z = 1

Cesar: Instrução de Desvio Incondicional

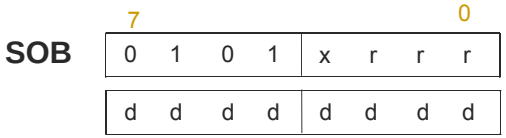
- Ocupa 2 bytes
- Formato:



  - bits x: podem assumir qualquer valor
  - Especificação do endereço de desvio:
    - **campo mmm**: modo de endereçamento
    - **campo r r r**: registrador utilizado

Cesar: Instrução de Controle de Laço

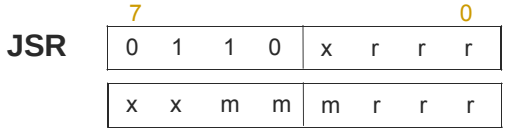
- Ocupa 2 bytes
- Significado:
  - *Subtract One and Branch if not zero*
  - utiliza um registrador como contador
- Formato:



  - **bits r r r**: registrador usado como contador
  - **bits d**: endereço para o desvio (início do laço), expresso como um deslocamento relativo ao PC

Cesar: Instrução de desvio para Subrotina

- Ocupa 2 bytes, com o seguinte formato:

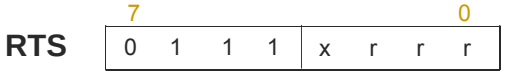


  - **bits r r r do 1o. byte**: registrador de retorno
    - pode ser o próprio PC! (i.e., R7)
  - **bits mmm e rrr (2o. byte)**: modo de endereçamento e registrador – usados para calcular o endereço da subrotina
- Execução:

  temporário ⟵ *endereço da subrotina*
  pilha ⟵ *registrador de retorno*
  registrador de retorno ⟵ *R7 (i.e., PC)*
  R7 ⟵ *temporário*

Cesar: Retorno de Subrotina

- Ocupa apenas 1 byte; formato:

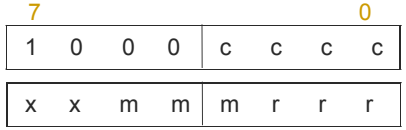


  - campo r r r: indica o registrador de retorno
    - contém o endereço de retorno
- Execução:

  R7 (PC) ⟵ *registrador de retorno*
  registrador de retorno ⟵ *topo da pilha*

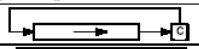
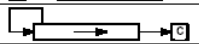
Cesar: Instruções de 1 Operando

- Formato geral:



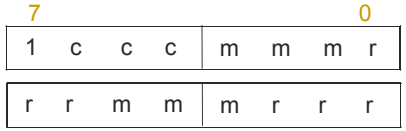
  - **campo cccc**: código específico da instrução
  - **campos mmm e rrr**: modo de endereçamento e registrador – usados para calcular o operando da instrução

Cesar: Instruções de 1 Operando

cccc	*instrução*	*Significado*	*N*	*Z*	*C*	*V*
0000	**CLR**	op ← 0	t	t	0	0
0001	**NOT**	op ← NOT op	t	t	1	0
0010	**INC**	op ← op + 1	t	t	t	t
0011	**DEC**	op ← op - 1	t	t	not(t)	t
0100	**NEG**	op ← - op	t	t	not(t)	t
0101	**TST**	op ← op	t	t	0	0
0110	**ROR**		t	t	lsb	xor
0111	**ROL**		t	t	msb	Xor
1000	**ASR**		t	t	lsb	Xor
1001	**ASL**		t	t	msb	Xor
1010	**ADC**	op ← op + c	t	t	t	T
1011	**SBC**	op ← op - c	t	t	t	T

Cesar: Instruções de 2 Operandos

- Formato geral:



  - **campo ccc**: código específico da instrução
  - **primeiro par mmm e rrr**: operando fonte
  - **segundo par mmm e rrr**: operando destino

Cesar: Instruções de 2 Operandos

ccc	*instrução*	*significado*	*N*	*Z*	*V*	*C*
001	**MOV**	dst ← src	t	t	0	-
010	**ADD**	dst ← dst + src	t	t	t	t
011	**SUB**	dst ← dst - src	t	t	t	not(t)
100	**CMP**	src - dst	t	t	t	not(t)
101	**AND**	dst ← dst AND src	t	t	0	-
110	**OR**	dst ← dst OR src	t	t	0	-

Cesar: Codificação Simbólica (com uso do montador)

- Modos de endereçamento

Modo	Símbolo
Registrador	Rx
Reg. (indireto) com pós-decremento	(Rx)+
Reg. (indireto) com pré-decremento	-(Rx)
Indexado	ddd(Rx)
Registrador indireto	(Rx)
Reg. pós-incr. (duplamente) indireto	((Rx)+)
Reg. pré-decr. (duplamente) indireto	(-(Rx))
Indexado Indireto	(ddd(Rx))
Imediato	*#nnn* (onde: *nnn* = valor da constante)
Absoluto ou direto	*nnn* (onde: *nnn* = endereço de mem.)

Cesar: Codificação Simbólica (com uso do montador)

- Instruções (mnemônicos)

Categoria de Instruções	Codificação simbólica
Instruções sobre códigos de condição	**CCC [N] [Z] [V] [C]** **SCC [N] [Z] [V] [C]**
Instruções de desvio condicional	**Bccc *ddd*** (onde: *ddd* = deslocamento em relação ao R7. Ex.: BNE -16)
Instrução de desvio incondicional	**JMP *end*** (onde: *end* indica o endereço alvo do desvio, com uso de um modo de endereçamento apropriado)
Instrução de controle de laço	**SOB Rx, *ddd*** (onde: ddd = deslocamento)
Instruções para desvio e retorno de subrotina	**JSR Rx, end** (onde: *end* = endereço da subrotina) **RTS Rx** (e Rx = reg. de retorno)
Instruções de um operando	**XXX *op_end*** (onde: op_end = end. do operando) (Ex.: DEC op_end ➢ op ← op – 1)
Instruções de dois operandos	**XXX *end_fonte*, *end_dest*** (Ex.: MOV end_fonte, end_dest ➢ dest ← fonte)

Cesar: Entrada e Saída

- Visor alfanumérico com 36 posições
  - letras e dígitos
- Teclado
  - leitura de um caractere
  - teste se uma tecla foi digitada
  - ➢ Usa o teclado do computador hospedeiro
- E/S mapeada na memória, nos últimos 38 bytes
  - endereços 65500 a 65535: visor
  - 65499: último caractere digitado
  - 65498: estado do teclado (80h se um caractere foi digitado)
    - Este byte deve ser zerado toda vez que um caractere é lido (em preparação para a leitura do próximo caractere)

Cesar: Entrada e Saída Exemplo

```
MOV #65498, R3      ;status do teclado
MOV #65500, R1      ;1a. posição do visor
CLR (R3)            ;zera status do tecl.
TST (R3)            ;testa se tecla digit.
BEQ -4              ;senão, volta p/ teste
MOV 65499, (R1)     ;se sim, escreve caract.
                    ;digitado no visor
INC R1              ;próxima pos. do visor
BEQ -18             ;testa se fim do visor
BR -16              ;para ler próx. tecla
HLT
```

Cesar: Subrotinas

- JSR Rx, *end*
  - Rx: registrador de retorno (ou de ligação)
  - *end*: endereço da subrotina
  - Rx é salvo na pilha
  - Rx ← endereço de retorno (byte seguinte ao JSR)
  - R7 ← *end* (i.e., desvio para *end*)
- RTS Rx
  - R7 ← Rx
  - Rx ← topo da pilha

Observação:
Pode-se usar R7 como o próprio registrador de retorno (mais comum em arquiteturas atuais, mas menos flexível)

Exemplo

```
14 ...
16 MOV R0, #64
20 JSR R5, (R0)  ;chamada
22 ...
.. ...
.. ...
64 MOV R3, R4
.. ...       ;corpo da subrotina
.. ...
78 RTS R5    ;retorno
```

Exemplo com passagem de parâmetros

```
19  ...
20  JSR R5, 62      ;chamada da subrotina
24  param 1         ;vetor de parâmetros (3 bytes)
26  param 2
28  param 3
30  ...             ;posição para onde deve retornar
..  ...
..  ...
62  CLR R4          ;início da subrotina
64  MOV (R5)+, R1   ;obtém o primeiro parâmetro
66  MOV (R5)+, R2   ;obtém o segundo parâmetro
68  MOV (R5)+, R3   ;obtém o terceiro parâmetro
..  ...
84  RTS R5          ;retorno da subrotina
```

A Biblioteca BibCesar

- Limpar o visor – rotina iterativa
- Multiplicar dois inteiros positivos de 16 bits
- Limpar o visor – rotina mais rápida
- Dividir um número inteiro positivo de 32 bits por outro de 16 bits
- Escrever no visor
- Identificação da versão da biblioteca

Bibliografia

- Weber, Raul F. “**Fundamentos deArquitetura de Computadores**”. 2a. Edição. Sagra-Luzzatto, 2001
  - Capítulos 4, 5 e 11