Arquitetura e Organização de computadores

ENGENHARIA DA COMPUTAÇÃO – UFC/SOBRAL

Prof. Danilo Alves

danilo.alves@alu.ufc.br

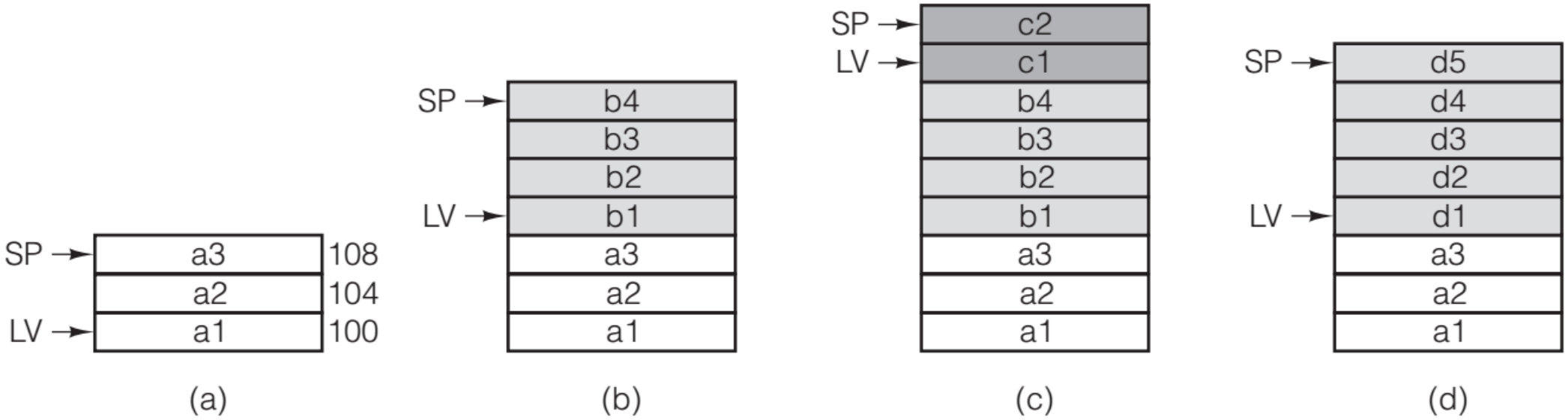
Exemplo de ISA: IJVM

- Introduzir um nível ISA, a ser interpretado pelo microprogarma que esteja rodando na microarquitetura
- Macroarquitetura (em contraste com microarquitetura), a arquitetura do conjunto de instruções do nível ISA

Pilhas

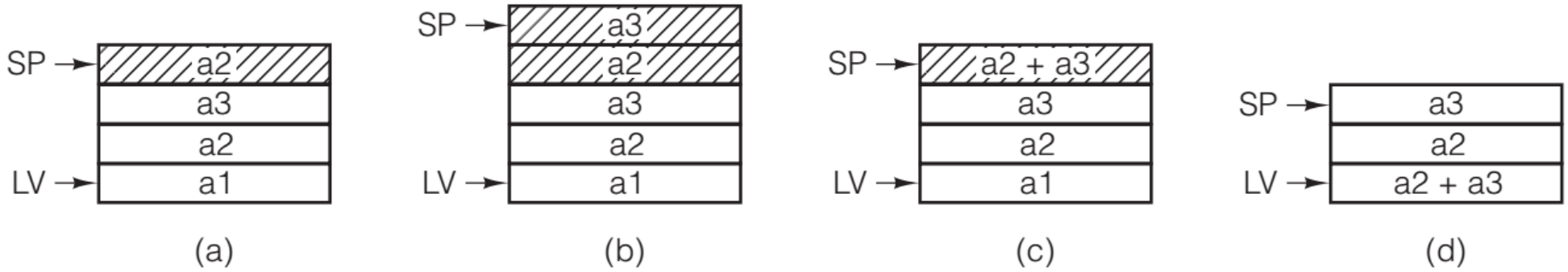
- O que são variáveis locais?
- Em qual lugar de memória devem ser mantidas?
- Problemas de endereços absolutos em recursão
- Área da memória denominada pilha é reservada para variáveis
- LV (Local Variable) para a base e SP (Stack Pointer) para a palavra mais alta
- Quadro de variáveis

Pilhas


SP
LV
a3
a2
a1
108
104
100
(a)
SP
LV
b4
b3
b2
b1
a3
a2
a1
(b)
SP
LV
c2
c1
b4
b3
b2
b1
a3
a2
a1
(c)
SP
LV
d5
d4
d3
d2
d1
a3
a2
a1
(d)

Pilhas

- Pilha de Operandos
- armazenamento de operandos durante a avaliação de uma expressão
- Cálculo de uma expressão: a1 = a2 + a3



Modelo de Memória IJVM

- Memória 4 GB e palavras de 4 bytes
- Áreas de memória
- O Pool de Constantes
  - Não pode ser escrita pelos programas IJVM
  - Contém constantes, strings e ponteiros para endereços de memória
  - O registrador CPP (Constant Pool Pointer) aponta para a primeira palavra do pool de constantes
- O Quadro de Variáveis Locais
  - No início guarda valores dos parâmetros (argumentos) do procedimento chamado
  - O registrador LV contém o endereço da primeira posição do quadro de variáveis locais

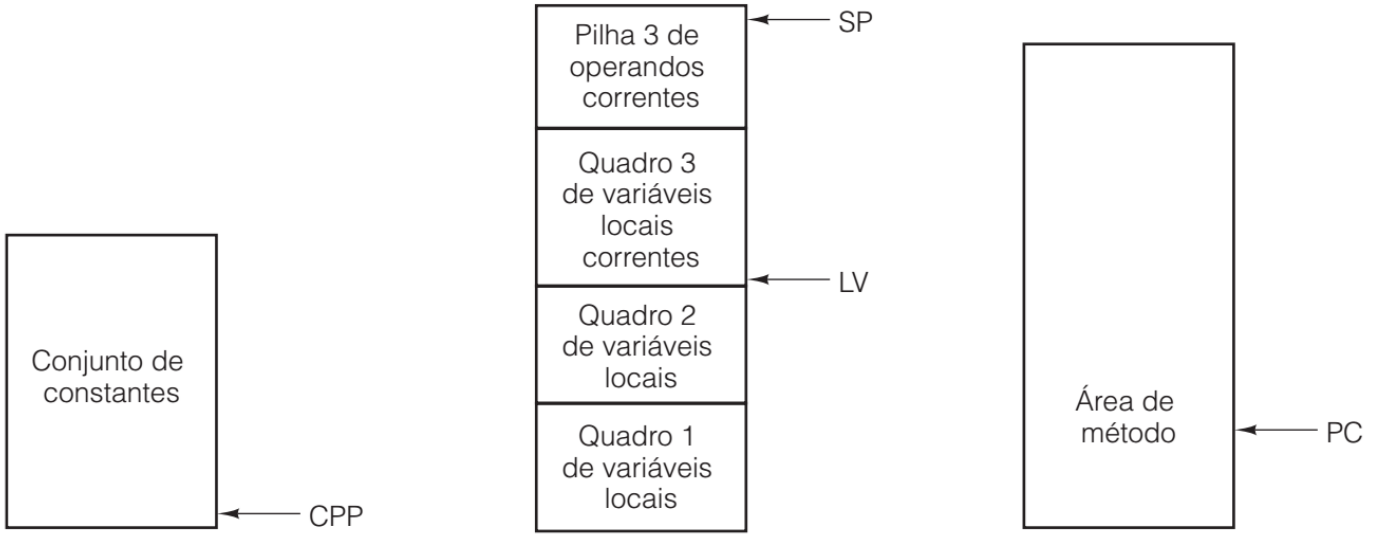
Modelo de Memória IJVM

- A Pilha de Operandos
  - Fica imediatamente acima do quadro de variáveis locais
  - Parte das variáveis locais
  - O registrador SP
    - Armazena o endereço do topo da pilha
    - Muda de valor durante a execução do procedimento

- A Área do Procedimento (método)
  - Região de memória que armazena o programa
  - O registrador PC (Program Counter) aponta para o endereço da instrução que deve ser buscada em seguida.

Modelo de Memória IJVM

- Observações:
  - CPP, LV e SP apontam para palavras (4bytes cada palavra)
  - PC aponta para bytes (cada instrução ocupa 1byte)
  - LV, LV+1 e LV+2 são as 3 primeiras palavras do quadro
  - LV, LV+4 e LV+8 são 3 palavras em intervalos de 4 bytes



Instruções da IJVM

- Mnemônicos em linguagem de montagem
- Constantes (LDC_W)
- Quadro de variáveis locais (ILOAD)
- Retirar valor da pilha e colocar em variável local (ISTORE)
- Operações aritméticas (IADD e ISUB) e lógica (IAND e IOR)
- Instruções de desvio
- Trocar palavras do topo uma pela outra (SWAP)

Instruções da IJVM

Hexa	Mnemônico	Significado
0x10	BIPUSH *byte*	Carregue o byte para a pilha
0x59	DUP	Copie a palavra do topo da pilha e passe-a para a pilha
0xA7	GOTO *offset*	Desvio incondicional
0x60	IADD	Retire duas palavras da pilha; carregue sua soma
0x7E	IAND	Retire duas palavras da pilha; carregue AND booleano
0x99	IFEQ *offset*	Retire palavra da pilha e desvie se for zero
0x9B	IFLT *offset*	Retire palavra da pilha e desvie se for menor do que zero
0x9F	IF_ICMPEQ *offset*	Retire duas palavras da pilha; desvie se iguais
0x84	IINC *varnum const*	Some uma constante a uma variável local
0x15	ILOAD *varnum*	Carregue variável local para pilha
0xB6	INVOKEVIRTUAL *disp*	Invoque um método
0x80	IOR	Retire duas palavras da pilha; carregue OR booleano

Instruções da IJVM

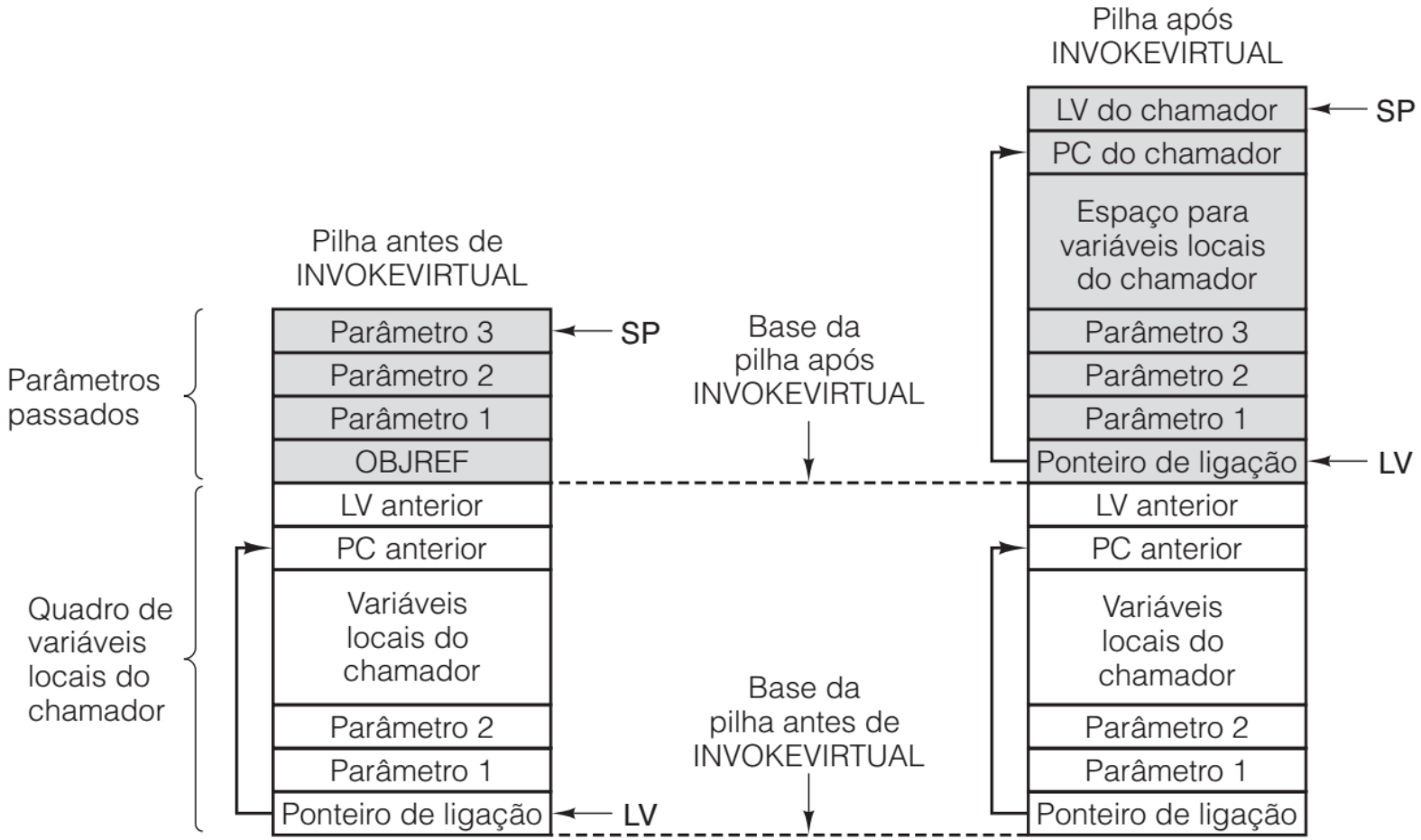
0xAC	IRETURN	Retorne do método com valor inteiro
0x36	ISTORE *varnum*	Retire palavra da pilha e armazene em variável local
0x64	ISUB	Retire duas palavras da pilha; carregue sua diferença
0x13	LDC_W *index*	Carregue constante do conjunto de constantes para pilha
0x00	NOP	Não faça nada
0x57	POP	Apague palavra no topo da pilha
0x5F	SWAP	Troque as duas palavras do topo da pilha uma pela outra
0xC4	WIDE	Instrução prefixada; instrução seguinte tem um índice de 16 bits

Instruções da IJVM

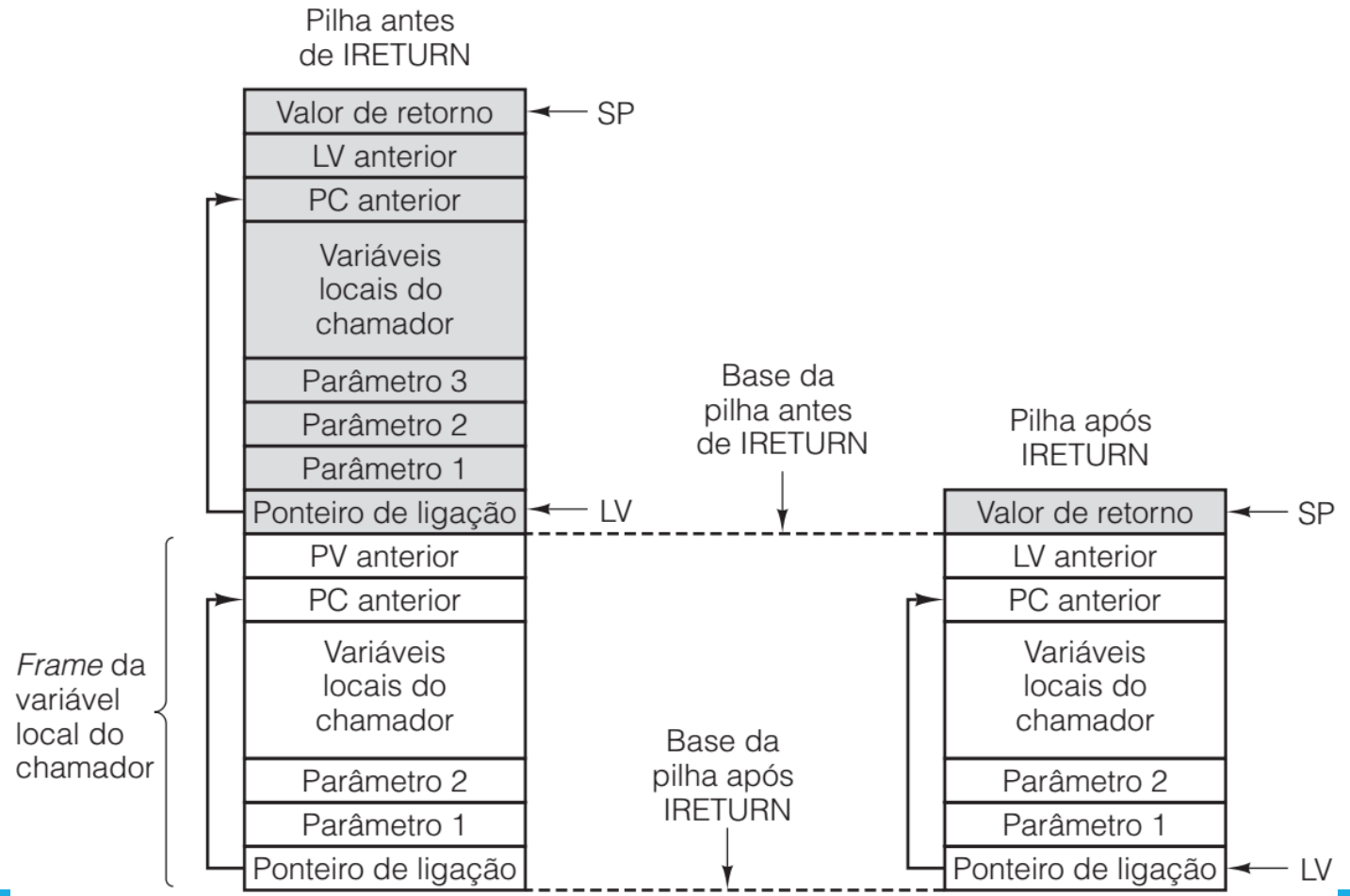
Mecanismo usado para implementar uma chamada a procedimento (INVOKEVIRTUAL)

- Chamador coloca na pilha uma referência (ponteiro) para o objeto a ser chamado, identificado por OBJREF
- Coloca parâmetros (Parâmetro 1, Parâmetro 2, ...) do procedimento na pilha
- A instrução INVOKEVIRTUAL é, então, executada
- As variáveis locais do procedimento são colocadas na pilha, sobre os parâmetros
- Os ponteiros PC e LV do chamador são salvos no topo da pilha
- A execução da instrução IRETURN reverte as operações realizadas pela INVOKEVIRTUAL
- A pilha retorna ao seu estado anterior, sem os parâmetros do procedimento, e os valores de LV e PC anteriores são recuperados
- O valor de retorno do procedimento é colocado no topo da pilha

Instruções da IJVM



Instruções da IJVM



Compilando Java para a IJVM

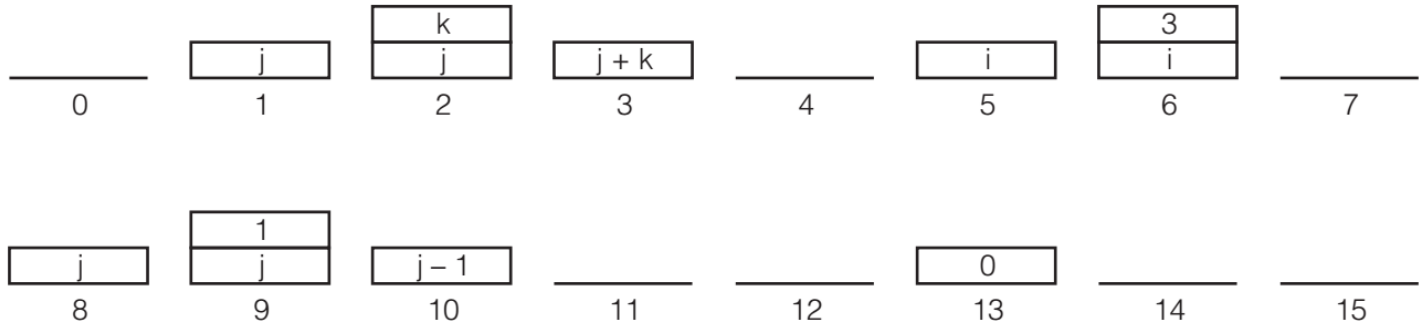
- Compilador Java produz código em linguagem de montagem
- Assembler Java traduz para binário
- Variáveis locais: i, j e k

```
i = j + k;
if (i == 3)
    k = 0;
else
   j = j – 1;
```

```
1        ILOAD  j          // i = j + k        0x15  0x02
2        ILOAD  k                              0x15  0x03
3        IADD                                  0x60
4        ISTORE i                              0x36  0x01
5        ILOAD  i          // if (i == 3)      0x15  0x01
6        BIPUSH 3                              0x10  0x03
7        IF_ICMPEQ L1                          0x9F  0x00  0x0D
8        ILOAD  j          // j = j – 1        0x15  0x02
9        BIPUSH 1                              0x10  0x01
10       ISUB                                  0x64
11       ISTORE j                              0x36  0x02
12       GOTO   L2                             0xA7  0x00  0x07
13  L1:  BIPUSH  0         // k = 0            0x10  0x00
14       ISTORE  k                             0x36  0x03
15  L2:
```

Compilando Java para a IJVM

```
 1          ILOAD  j          // i = j + k
 2          ILOAD  k
 3          IADD
 4          ISTORE i
 5          ILOAD  i          // if (i == 3)
 6          BIPUSH 3
 7          IF_ICMPEQ L1
 8          ILOAD  j          // j = j – 1
 9          BIPUSH 1
10          ISUB
11          ISTORE j
12          GOTO   L2
13    L1:   BIPUSH  0         // k = 0
14          ISTORE  k
15    L2:
```



Microinstruções

- Vários sinais podem ser ativados em cada ciclo
- Ex: Incrementar o valor de SP e leitura da próxima instrução

```
ReadRegister = SP, ULA = INC, WSP, Read, NEXT_ADDRESS = 122
```

- SP = SP + 1; rd
- Micro Assembly Language (MAL)
- Cada ciclo somente um Registrador pode ser escrito
- Lado A da ULA: +1, 0, -1
- Baseada em Java:
- MDR = SP; MDR = H + SP

Microinstruções

- Operações permitidas
  - SOURCE = Entrada no Barramento B
  - DEST = Saída para Barramento C

  - MDR = SP + MDR
  - MDR = H - MDR

DEST = H
DEST = SOURCE
DEST = $\overline{H}$
DEST = $\overline{SOURCE}$
DEST = H + SOURCE
DEST = H + SOURCE + 1
DEST = H + 1
DEST = SOURCE + 1
DEST = SOURCE – H
DEST = SOURCE – 1
DEST = –H
DEST = H AND SOURCE
DEST = H OR SOURCE
DEST = 0
DEST = 1
DEST = –1

Microinstruções

- Múltiplas atribuições:
  - SP = MDR = SP + 1
- Formas de acessar a memória:
  - MAR/MDR por rd e wr -> Palavras de dados de 4 bytes
  - PC/MBR por fetch -> instrução de 1 byte da sequência de instruções
- Registrador não pode receber valor da memória e do caminho de dados simultaneamente
  - MAR = SP, rd
  - MDR = H

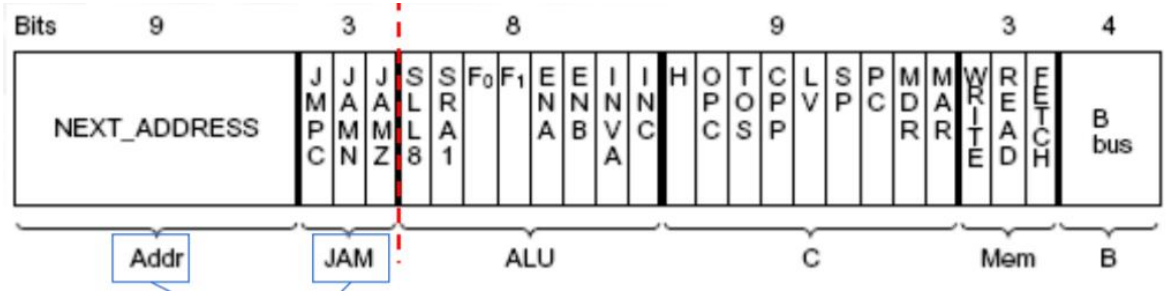
Microinstruções

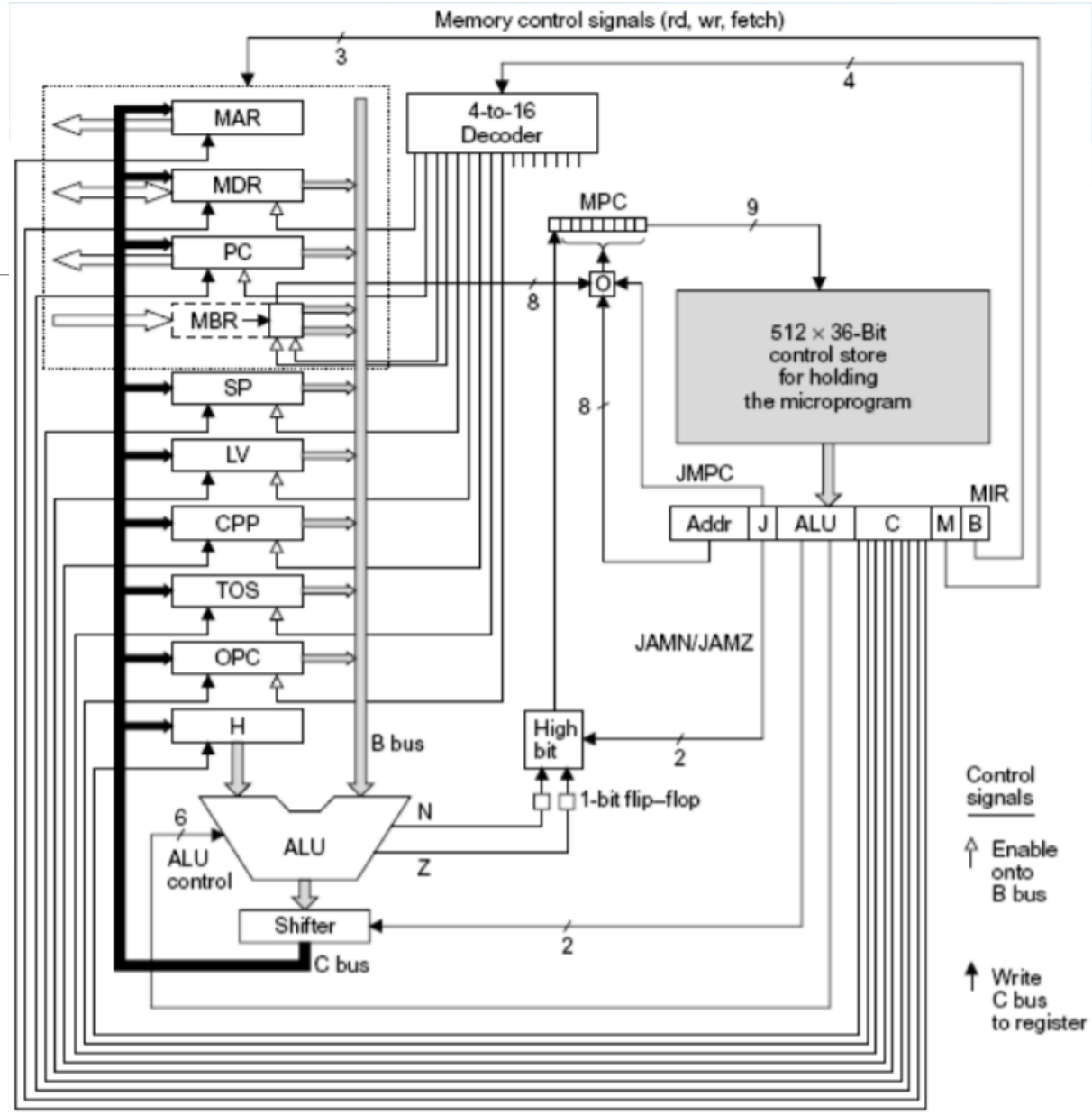
- Próxima Microinstrução em NEXT_ADDRESS
  - Goto label
  - Goto Main1

- Desvios condicionais
  - TOS = TOS
  - Z = TOS
  - Ajustar todos os bits do campo C da microinstrução para 0
  - Z = TOS; if(Z) goto L1; else goto L2

Microinstruções

- Z = TOS; if(Z) goto L1; else goto L2
  - Z é carregado e passar por um OR com o bit de alta ordem do MPC
  - Força a modificar o endereço para L1 ou L2
  - Após isso MPC pode ser carregado





Referências

- Andrew S. Tanenbaum, Organização Estruturada de Computadores, 5ª edição, Prentice-Hall do Brasil, 2007.