Ch 4 IJVM Example

Page 243: “Ideally, we would like to introduce this subject by explaining general principles of microarchitecture design. Unfortunately, there are no general principles; every ISA is a special case.”

So...let’s do the small example from Chapter 4 using IJVM, and Mic-1.

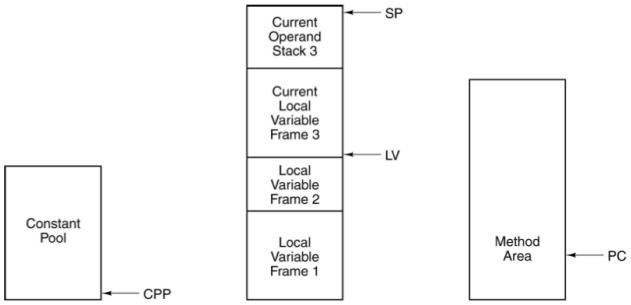
```
i = j + k;                1     ILOAD j          // i = j + k      0x15 0x02
if (i == 3)               2     ILOAD k                         0x15 0x03
    k = 0;                3     IADD                            0x60
else                      4     ISTORE i                        0x36 0x01
    j = j – 1;            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:
        (a)                          (b)                              (c)
```

Page 266, Figure 4-14: our example in a) Java, b) IJVM assembly, c) IJVM machine lang in hex

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

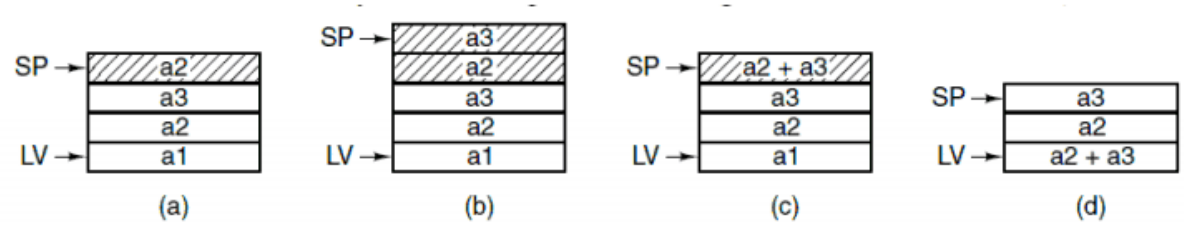
Page 262, Figure 4-11: The IJVM instruction set

Stacks are used for computation, local variables, and method calls



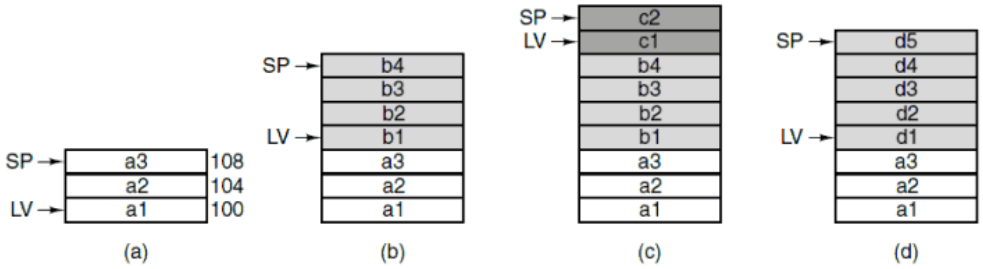
Page 261, Figure 4-10: IJVM memory

Computation example: a1 = a2 + a3 => (a) push a2, (b) push a3, (c) add, (d) store a1



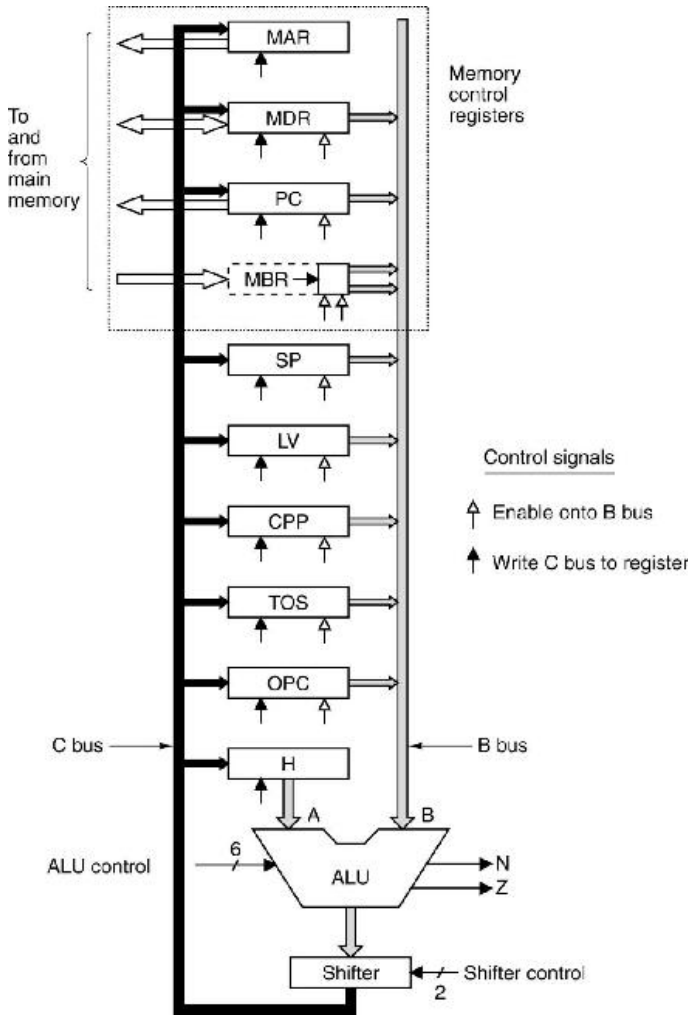
Page 259, Figure 4-9: Using a stack for computation

Local variable example: (a) 3 local vars in method A, (b) After A calls B, (c) After B calls C, and (d) After B and C return and A calls method D



Page 259, Figure 4-8: Using a stack for local variables

Datapath: The hardware on which IJVM will run:



Page 245, Figure 4-1: Datapath

ALU: Our guy from Ch 3/Program #2. **Shifter**: shifts bits left or right depending on control bits.

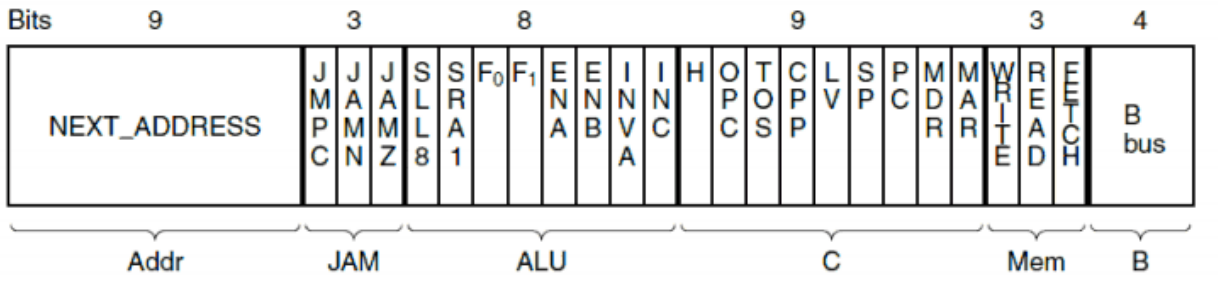
Registers are:

MAR - Memory Address Register	PC - Program Counter, mem addr of next instr
MDR - Memory Data Register	MBR - 1 byte reg, aka instr opcode
SP - stack pointer	CPP - constant pool pointer
LV - local variable frame pointer	TOS - value at the top of the stack

See page 254, Figure 4-6 for the full Mic-1 architecture.

The connection between IJVM assembly language and microcode is the hex opcode. The hex opcode for each IJVM assembly instruction is actually an address in the microcode ROM. That address is the location of the first micro-instruction for that IJVM assembly instruction.

The connection between microcode and your datapath is a microinstruction.



B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Microcode is the glue connecting IJVM opcodes to the Mic-1 microarchitecture. There are many micro-instructions for each opcode. The micro-instructions describe the ALU/Shifter/Register changes for each cycle.

See page 272-274 for the microcode for Mic-1.

So, follow along… it's Java -> IJVM assembly -> IJVM hex machine code -> Mic-1 microcode instructions -> datapath.