IJVM instruction set

Here's the bogey. Figure 4-11: The IJVM instruction set

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

This is a 0 address, stack-based ISA. IJVM is all integer. This is a 32 bit machine.
Memory limit is 4GB, or 1 giga-words. Word = 4 bytes. Most things are 4 byte (int) quantities.
Opcodes are 1 byte.

Implicit registers:

- Program counter (PC) - address of the instruction next to fetch
- Stack pointer (SP) - top of the stack, where local vars and method parameters reside
- Local variable frame (LV) - address of stack for local variables of current method
- Constant pool (CPP) - read-only area, all constants are an offset from this base address

Offsets are word offsets, so CPP+1 refers to the 2nd word in the constant pool, not the 2nd byte. PC values are bytes, so offsets there are in bytes.

Notice in Figure 4-14 - the translation from Java to IJVM assembly is complex. Each Java statement means multiple IJVM assembly statements.
The translation from assembly code to machine code, however, is pretty easy.

For fun, I sorted the IJVM instructions by functionality.

	Instruction	Description
Loads	`ILOAD varnum`	Push local var on stack
	`LDC_W index`	Push const from const pool on stack
Stores	`ISTORE varnum`	Pop word from stack and store in local var
Stack ops	`BIPUSH byte`	Push byte onto stack
	`DUP`	Copy top word on stack and push it
	`POP`	Pop top word off of stack
	`SWAP`	Swap the top words on the stack
Arithmetic	`IADD`	Pop two words from stack; push their sum
	`ISUB`	Pop two words from stach; push their difference
	`IINC varnum const`	Add const value to a local var
Logic	`IAND`	Pop two words from stack; push their Boolean AND
	`IOR`	Pop two words from stack; push their Boolean OR
Cmp/Branch	`GOTO offset`	Unconditional branch
	`IFEQ offset`	Pop word from stack, branch if it is 0
	`IFLT offset`	Pop word from stack, branch if it is < 0
	`IF_ICMPEQ offset`	Pop two words from stack, branch if equal
Methods	`INVOKEVIRTUAL disp`	Invoke/call a method/procedure
	`IRETURN`	Return from a method
etc	`NOP`	Do nothing
	`WIDE`	Prefix instruction, next instruction have 16 bit index

Comments:

- This is a stack-based architecture. `ILOAD` is push. `ISTORE` is pop. the other stack ops operate as expected.
- Normal Arithmetic/Logic operations are postfix. Add example: Push a value (using `ILOAD`), push another value, and the `IADD` pops 2 values, adds them and then pushes the sum onto the stack.
- Branching `offset` parameter is 2 bytes. It's added to the current instruction location (PC). Negative numbers are allowed, and a 2's complement representation is used.
- Note that the `varnum` parameter is a 1 byte quantity that indicates the word offset (not byte) from the local variable frame. So, variables are reference by their number: 1, 2, 3...
- The `index` parameter of `LDC_W` is again a word offset from the CPP register. So, constants are numbered as well.
- There's a lot of shenanigans surrounding `INVOKEVIRTUAL` (can you say "call"... jeez) and `IRETURN`. These instructions are just about implementing a stack pointer for method parameters and a local variable frame pointer for local variables to be placed on the same stack.