Some JVM Instructions

These tables contain all instructions used in Chapters 5 and 6 and some other ones that can be useful as optimizations in Assignment 4. We use the dot (.) to separate values on the stack, and two-letter variables (*dd*,*ee*) to represent double values. The asterisk (*) in an explanation indicates that there is a longer explanation after the tables.

Jasmin	**args**	**stack**	**explanation**	**HEX**
`aload`	var i	$. \rightarrow .V(i)$	load ref from var i	19
`aload_i` (i=0..3)		$. \rightarrow .V(i)$		2A..2D
`areturn`		$.r \rightarrow$	return ref from method	B0
`astore`	var i	$.r \rightarrow .$	store ref in var i	3A
`astore_i` (i=0..3)		$.r \rightarrow .$	store ref in var i	4B..4E
`bipush`	byte i	$. \rightarrow .i$	push byte i as int	10
`d2i`		$.dd \rightarrow .i$	convert double to int	8E
`dadd`		$.dd.ee \rightarrow .dd + ee$	add double	63
`dcmpg`		$.dd.ee \rightarrow .i$	compare if >*	98
`dcmpl`		$.dd.ee \rightarrow .i$	compare if <*	97
`dconst_dd` (dd=0,1)		$. \rightarrow .dd$	push double	0E,0F
`ddiv`		$.dd.ee \rightarrow .dd/ee$	divide double	6F
`dload`	var i	$. \rightarrow .V(i)$	load double from var i	18
`dload_i` (i=0..3)		$. \rightarrow .V(i)$	load double from var i	26..29
`dmul`		$.dd.ee \rightarrow .dd * ee$	multiply double	6B
`dneg`		$.dd \rightarrow . - dd$	negate double	77
`dreturn`		$.dd \rightarrow$	return double from method	AF
`dstore`	byte i	$.dd \rightarrow .$	store double in var i	39
`dstore_i` (i=0..3)		$.dd \rightarrow .$	store double in var i	47..4A
`dsub`		$.dd.ee \rightarrow .dd - ee$	subtract double	67
`dup`		$.v \rightarrow .v.v$	duplicate top (for int)	59
`dup2`		$.dd \rightarrow .dd.dd$	duplicate top (for double)*	5C

Jasmin	**args**	**stack**	**explanation**	**HEX**
`goto`	label L		go to label L	A7
`i2d`		$.i \rightarrow .dd$	convert int to double	87
`iadd`		$.v.u \rightarrow .v + u$	add int	60
`iconst_m1`		$. \rightarrow . - 1$	push int constant -1	02
`iconst_i` (i=0..5)		$. \rightarrow .i$	push int constant	03..08
`idiv`		$.v.u \rightarrow .v/u$	divide int	6C
`if_icmpeq..le`	label L	$.v.u \rightarrow .$	compare ints on stack*	9F..A4
`ifeq..le`	label L	$.v \rightarrow .$	compare int with 0*	99..9E
`iinc`	var i, byte c		increment var i with c	84
`iload`	ref i	$. \rightarrow .V(i)$	load int from var i	15
`iload_i` (i=0..3)		$. \rightarrow .V(i)$	load int from var i	1A..1D
`imul`		$.v.u \rightarrow .v * u$	multiply int	68
`ineg`		$.v \rightarrow . - v$	negate int	74
`invokestatic`	method	$.v...w \rightarrow .$	call static method	B8
`invokevirtual`	method	$.v...w \rightarrow .$	call virtual method	B6
`irem`		$.v.u \rightarrow .v\%u$	remainder int	70
`ireturn`		$.v \rightarrow$	return int from method	AC
`istore`	ref i	$.v \rightarrow .$	store int in var i	36
`istore_i` (i=0..3)		$.v \rightarrow .$	store int in var i	3B..3E
`isub`		$.v.u \rightarrow .v - u$	subtract int	64
`ldc`	int v	$. \rightarrow .v$	push int constant v*	12
`ldc2_w`	double dd	$. \rightarrow .dd$	push double constant dd*	14
`nop`			do nothing	00
`pop`		$.v \rightarrow .$	pop int	57
`pop2`		$.dd \rightarrow .$	pop double*	58
`return`		$. \rightarrow$	return void from method	B1

More explanations:

- `dcmpg`, `dcmpl`: the value left on the stack is 1 if the inequality holds, 0 if the values are equal, and -1 otherwise.
- `dup2`: the instruction duplicates the topmost two words, which can be one double value or two integer values.
- `if_icmpeq`, `if_icmpne`, `if_icmplt`, `if_icmpge`, `if_icmpgt`, `if_icmple` corresponding to $=, \neq, <, \geq, >, \leq$ jump to the label if the comparison holds between the top-2 integer values on the stack.
- `ifeq`, `ifne`, `iflt`, `ifge`, `ifgt`, `ifle` corresponding to $=, \neq, <, \geq, >, \leq$ jump to the label if the comparison holds between the top integer value on the stack and 0.
- `ldc`, `ldc2_w`: the constants pushed are stored in the constant pool, and the actual bytecode argument (after assembly) is a reference to this pool.
- `pop2`: the instruction pops the topmost two words, which can be one double value or two integer values.