CS 2211 - Winter 2024 - Assignment 5 Marking Scheme

Part 1 - Makefile (10%)	Student Mark	Out of	Comments
Default target			
Builds successfully	2	2	
Links to ADT library	3	3	
minijvm.o target			
Builds successfully	2	2	
Adds include directory to GCC's search path	3	3	
clean target			
Deletes executable	2	2	
Deletes object files	2	2	
testN targets (test1 through test8)			
testN target present and runs ./mjava testfiles/testN	2	2	
test target			
Runs test1 through test8	2	2	
Continues running tests when a test encounters an error	2	2	
Part 1 - Deductions	**Student Mark**	**Out of**	**Comments**
Had to rename the file to Makefile		-2	
Makefile contained syntax errors and required modifications to get it to work		-10	
Makefile otherwise required modifications to get it to work		-5	
One or more invocations of GCC compiled header (.h) files		-2	
Makefile generates wrong executable name		-1	
Total	**20.00**	**20.00**	
Out of 100	**100.00**	**100.00**	

Part 2 (90%) - IF CODE DID NOT COMPILE	Student Mark	Out of	Comments
Maximum grade of 45% based on a quick review of the code		100	
Total	**0.00**	**100.00**	

Part 2 (90%) - IF CODE COMPILED	Student Mark	Out of	Comments
Code was compiled with a unit test suite containing 29 tests that tested functionality per the assignment spec. If your code passed a test, you received full marks. If it did not, you received 45% of the test weight, as a means of assigning part marks and being somewhat gentle in marking. Note that it is possible that you passed all eight of the provided .mclass files but still failed tests here. In most cases, this is due to the fact that you missed a particular edge case. Note that I threw out the tests that tested functions like jvm_init and jvm_read because too many people did not follow instructions, making it difficult to test these functions.			
Unit Tests			
Test 1 - return stops execution (i.e., no infinite loop)	1	1	
Test 2 - return sets JVM return value to 0	1	1	
Test 3 - bipush correctly pushes positive operand to the stack	1	1	
Test 4 - bipush correctly pushes negative operand to the stack	1	1	
Test 5 - iconst0 correctly pushes zero to the stack	1	1	
Test 6 - pop correctly pops the stack	1	1	
Test 7 - dup correctly duplicates the top value of the stack	1	1	
Test 8 - iadd correctly pops the top two operands, adds them, and pushes the result to the stack	1	1	
Test 9 - isub correctly pops the top two operands, subtracts (SecondFromTop - Top), and pushes the result to the stack	1	1	
Test 10 - imul correctly pops the top two operands, multiplies them, and pushes the result to the stack	1	1	
Test 11 - idiv correctly pops the top two operands, divides (SecondFromTop / Top), and pushes the quotient to the stack	1	1	
Test 12 - idiv correctly handles division by zero, printing an error and exiting to the operating system with a status of 1	1	1	
Test 13 - irem correctly pops the top two operands, computes (SecondFromTop % Top), and pushes the result to the stack	1	1	
Test 14 - irem correctly handles division by zero, printing an error and exiting to the operating system with a status of 1	1	1	
Test 15 - ishr correctly pops the top two operands, computes (SecondFromTop >> Top), and pushes the result to the stack	1	1	
Test 16 - iprint correctly prints the top of the stack with the value and a newline (and nothing else)	1	1	
Test 17 - iprint does not pop the top of the stack, but merely prints its top value	1	1	
Test 18 - iload correctly loads local variables 0 through 9 onto the stack when the values of the variables are positive	1	1	
Test 19 - iload correctly loads local variables 0 through 9 onto the stack when the values of the variables are negative	1	1	
Test 20 - istore correctly pops the top of the stack and stores it into local variables 0 through 9 when the stack top is positive	1	1	
Test 21 - istore correctly pops the top of the stack and stores it into local variables 0 through 9 when the stack top is negative	1	1	
Test 22 - iinc correctly increments local variables 0 through 9 by a positive constant	1	1	
Test 23 - iinc correclty decrements local variables 0 through 9 by a negative constant	1	1	
Test 24 - ifeq correctly pops the stack and branches to a positive offset when the popped value is zero	1	1	
Test 25 - ifeq correctly pops the stack and skips over a positive offset to the next instruction when the popped value is not zero	1	1	
Test 26 - ifeq correctly pops the stack and branches to a negative offset when the popped value is zero	1	1	
Test 27 - ifeq correctly pops the stack and skips over a negative offset to the next instruction when the popped value is not zero	1	1	
Test 28 - goto correctly branches to a positive offset	1	1	
Test 29 - goto correctly branches to a negative offset	1	1	
Part 2 - Deductions	**Student Mark**	**Out of**	**Comments**
Code required modifications to get it to compile and/or run properly with the unit test suite		-4	
Total	**29.00**	**29.00**	
Out of 100	**100.00**	**100.00**	

Summary	Student Mark	Out of	Comments
Part 1 (10%)	10.00	10	
Part 2 (90%)	90.00	90	
Assignment Total	**100.00**	**100.00**	