SWEN90006
SOFTWARE TESTING AND RELIABILITY
ASSIGNMENT 1

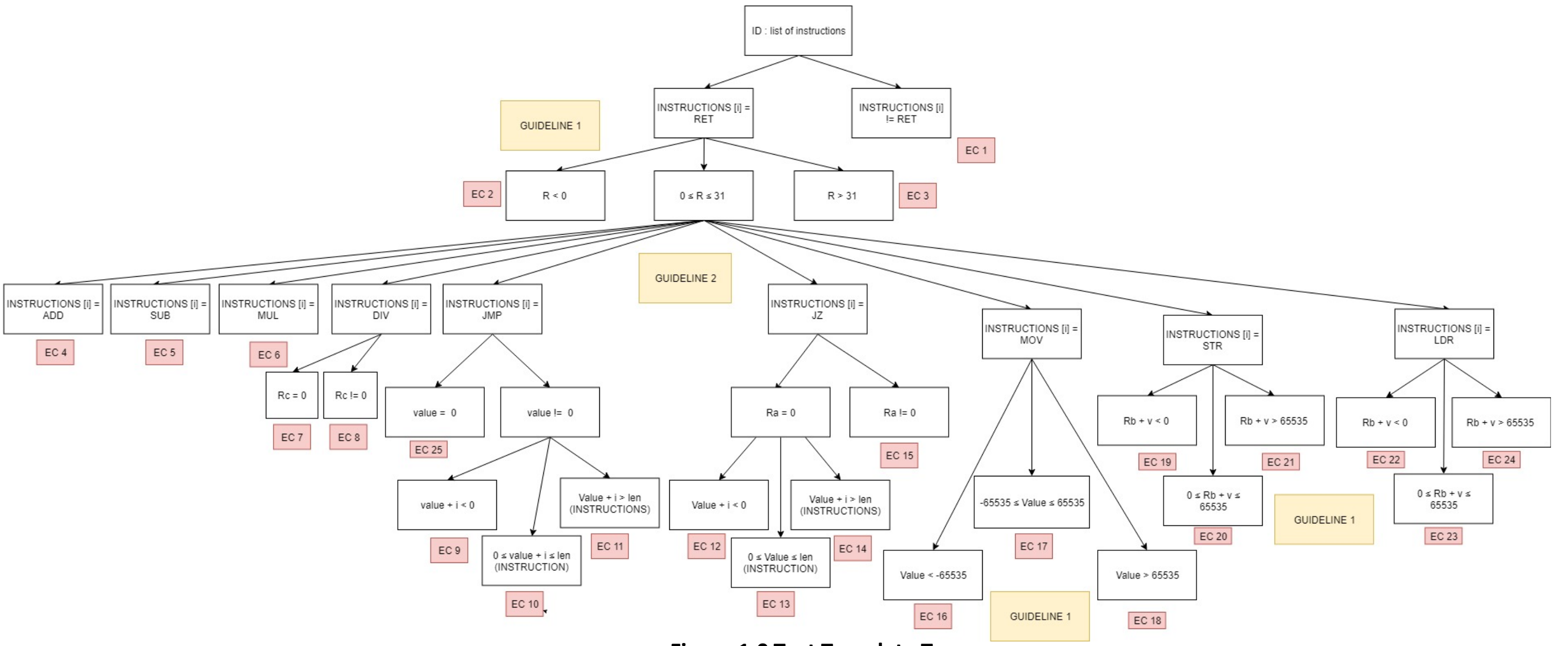

Figure 1.0 Test Template Tree

1. Based on the specification requirement, the input program in Machine class on **execute** method is a list of instruction strings. Each line or each string consists of an instruction which has specific rules for different type of command.

   Assuming that all syntax are correct, the format of the instructions (e.g grammar, order) will not be checked.

   First, related to a condition that in the last of instructions must have 'RET' command. The invalid class will be a list of instructions with no 'RET' command in the last instruction

   EC1invalid = { INSTRUCTIONS | INSTRUCTIONS [i] != RET}

   For each <<REGISTER>> element, as the input condition needs to be specified within a range, which is 0-31. Based on Guideline 1, some equivalence classes which consider that specification are built. This condition for R range will be applied into other instructions, but it will not be duplicated to the other equivalence classes to prevent redundant equivalence classes and tree explosion. Hence, there are two equivalence classes as the invalid inputs of R that generate InvalidInstructionException in the program. The valid R will be applied in the other equivalence classes.

EC2invalid = { INSTRUCTIONS | R < 0 }
EC3invalid = { INSTRUCTIONS | R > 31 }

Based on Guideline 2, each of specific instructions is considered in a different equivalence class.

EC4valid = { INSTRUCTIONS | INSTRUCTIONS [i] = ADD}
EC5valid = { INSTRUCTIONS | INSTRUCTIONS [i] = SUB}
EC6valid = { INSTRUCTIONS | INSTRUCTIONS [i] = MUL}

In 'DIV' operation, the value of denominator, which is Rc, resulting in different behavior in the program, which is no-op.

EC7valid = { INSTRUCTIONS | INSTRUCTIONS [i] = DIV ∩ Rc = 0}
EC8valid = { INSTRUCTIONS | INSTRUCTIONS [i] = DIV ∩ Rc != 0}

In 'JMP' and 'JZ' operations, the sum of the input value and the line instruction (i / pc) should not less than 0, and should not more than the number of line instructions as it generates NoReturnValueException in the program. Guideline 1 is applied in this case for splitting the value range.

EC9invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = JMP ∩ value != 0 ∩ value + i < 0}
EC10valid = { INSTRUCTIONS | INSTRUCTIONS [i] = JMP ∩ value != 0 ∩ 0 ≤ value + i ≤ len (INSTRUCTIONS)}
EC11invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = JMP ∩ value + i > len (INSTRUCTIONS)}
EC25valid = { INSTRUCTIONS | INSTRUCTIONS [i] = JMP ∩ value = 0}

EC12invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = JZ ∩ Ra = 0 ∩ value + i < 0}
EC13valid = { INSTRUCTIONS | INSTRUCTIONS [i] = JZ ∩ Ra = 0 ∩ 0 ≤ value + i ≤ len (INSTRUCTIONS)}
EC14 invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = JZ ∩ Ra = 0 ∩ value + i > len (INSTRUCTIONS)}
EC15valid = { INSTRUCTIONS | INSTRUCTIONS [i] = JZ ∩ Ra != 0}

For each <<VALUE>> element, to satisfy the input condition, which is an integer value between -65535 and 65535, some equivalence classes which consider that specification are built based on Guideline 1. Similar with instruction R, in order to avoid the redundancy, it is also assumed that this value condition is applied in all instructions.

EC16invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = MOV ∩ value < -65535}
EC17valid = { INSTRUCTIONS | INSTRUCTIONS [i] = MOV ∩ -65535 ≤ value ≤ 65535}
EC18invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = MOV ∩ value > 65535}

In order to handle LDR and STR operation, which contribute to an address value, the sum of Rb and the value should be in the range of 0 and 65535. Otherwise, it will do nothing (no-op). Guideline 1 is applied for these equivalence classes.

EC19invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = STR $\cap$ Rb + v < 0}
EC20valid = { INSTRUCTIONS | INSTRUCTIONS [i] = STR $\cap$ 0 ≤ Rb + v ≤ 65535}
EC21invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = STR $\cap$ Rb + v > 65535}
EC22invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = LDR $\cap$ Rb + v < 0}
EC23valid = { INSTRUCTIONS | INSTRUCTIONS [i] = LDR $\cap$ 0 ≤ Rb + v ≤ 65535}
EC24invalid = { INSTRUCTIONS | INSTRUCTIONS [i] = LDR $\cap$ Rb + v > 65535}

With the assumption of syntax correct, as the set of equivalences classes are built based on the specifications of the input conditions, all of those equivalence classes already cover all the input space. For example, it can be seen in the case of value condition that should not be smaller than -65535 and no larger than 65535. Some equivalence classes are created to cover all possibilities inputs whether it is valid or invalid. Hence, there are three possible input range that may occur in that case and all of them are included in the equivalence classes. Those possibilities are a value smaller than -65535 and larger than 65535 which are considered as the invalid input, then the valid input is between -65535 and 65535.

3. The equivalence classes which has no boundary will use the original test case from equivalence partitioning. Otherwise, on point and off point values are picked in each equivalence classes which has boundary. If more than one equivalence class produce the same on point/off point, there will be only one chosen because that condition could generate the same test case.

EC	**Boundaries**	**On Point**	**Off Point**
EC2	{ INSTRUCTIONS | R < 0 }	R0	R-1
EC3	{ INSTRUCTIONS | R > 31 }	R31	R32
EC7	{ INSTRUCTIONS | INSTRUCTIONS [i] = DIV $\cap$ Rc = 0}	Rc = 0	Rc = -1 Rc = 1
EC9	{ INSTRUCTIONS | INSTRUCTIONS [i] = JMP $\cap$ value + i < 0}	value + 1 = 0	value + 1 = -1
EC 10	{ INSTRUCTIONS | INSTRUCTIONS [i] = JMP $\cap$ 0 ≤ value + i ≤ len (INSTRUCTIONS)} Assuming len(INSTRUCTION) = 5	value + i = 5	
EC11	{ INSTRUCTIONS | INSTRUCTIONS [i] = JMP $\cap$ value + i > len (INSTRUCTIONS)} Assuming len(INSTRUCTION) = 5		value + i = 6
EC12	{ INSTRUCTIONS | INSTRUCTIONS [i] = JZ $\cap$ Ra = 0 $\cap$ value + i < 0}	value + i = 0	value + i = -1

EC13	{ INSTRUCTIONS | INSTRUCTIONS [i] = JZ ∩ Ra = 0 ∩ 0 ≤ value + i ≤ len (INSTRUCTIONS)} Assuming len(INSTRUCTION) = 5	value + i = 5	
EC14	{ INSTRUCTIONS | INSTRUCTIONS [i] = JZ ∩ Ra = 0 ∩ value + i > len (INSTRUCTIONS)} Assuming len(INSTRUCTION) = 5		value + i = 6
EC16 =	{ INSTRUCTIONS | INSTRUCTIONS [i] = MOV ∩ value < -65535}	value = -65535	value = -65536
EC17	{ INSTRUCTIONS | INSTRUCTIONS [i] = MOV ∩ -65535 ≤ value ≤ 65535}	value = 65535	
EC18	{ INSTRUCTIONS | INSTRUCTIONS [i] = MOV ∩ value > 65535}		value = 65536
EC19	{ INSTRUCTIONS | INSTRUCTIONS [i] = STR ∩ Rb + v < 0}	Rb + v = 0	
EC20	{ INSTRUCTIONS | INSTRUCTIONS [i] = STR ∩ 0 ≤ Rb + v ≤ 65535}	Rb + v = 65535	Rb + v = -1
EC21	{ INSTRUCTIONS | INSTRUCTIONS [i] = STR ∩ Rb + v > 65535}		Rb + v = 65536
EC22	{ INSTRUCTIONS | INSTRUCTIONS [i] = LDR ∩ Rb + v < 0}		Rb + v = -1
EC23	{ INSTRUCTIONS | INSTRUCTIONS [i] = LDR ∩ 0 ≤ Rb + v ≤ 65535}	Rb + v = 0	Rb + v = 65536
EC24	{ INSTRUCTIONS | INSTRUCTIONS [i] = LDR ∩ Rb + v > 65535}	Rb + v = 65535	
EC25	{ INSTRUCTIONS | INSTRUCTIONS [i] = JMP ∩ value = 0}	value = 0	value = -1 value = 1

Table 1.0 Boundary Analysis

5. Multiple-conditions coverage

In **execute** method, there are :

- 12 if statements containing a single condition = 12×2^1
- 1 if statements containing two conditions = 1×2^2
- 1 switch statement with 11 cases = 8

Total : $12 + 4 + 8 = \mathbf{39}$ possibilities

	Condition	Possible Outputs	Objective
C1	if (pc < 0 || pc >= length(instructions))	true false	1
		false false	2
		false true	3
		true true	4
C2	if instruction.equals(“”)	true	5
		false	6
C3	if (length(token) < 2)	true	7
		false	8
C4	switch (instructions[i])	ADD	9
		SUB	10
		MUL	11
		DIV	12
		MOV	13
		JZ	14
		JMP	15
		STR	16
		RET	17
		LDR	18
		INVALID	19
C5	ADD - if (length(token) != 4)	true	20
		false	21
C6	SUB - if (length(token) != 4)	true	22
		false	23
C7	MUL - if (length(token) != 4	true	24
		false	25
C8	DIV - if (length(token) != 4	true	26
		false	27
C9	LDR - if (length(token) != 4	true	28
		false	29
C10	STR - if (length(token) != 4	true	30
		false	31
C11	MOV - if (length(token) !=3	true	32

		false	33
C12	JMP - if (length(token) != 2	true	34
		false	35
C13	JZ - if (length(token) != 3	true	36
		false	37
C14	JZ - if (Ra == 0)	true	38
		false	39

Table 2.0 Multiple-condition for the program

- **Multiple-condition coverage in equivalence partitioning**

	Test Case	Meet with Objective
EC1	ADD R1 R2 R3	2, 6, 8, 9, 21
EC2	ADD R1-32 R2 R3 RET R1	2, 6, 8, 9,17, 21
EC3	ADD R100 R2 R3 RET R1	2, 6, 8, 9,17, 21
EC4	MOV R1 4 MOV R2 5 ADD R3 R1 R2 RET R3	2, 5, 6, 8, 13, 17, 21, 33
EC5	MOV R1 10 MOV R2 15 SUB R3 R1 R2 RET R3	2, 6, 8, 10, 13, 17, 23, 33
EC6	MOV R1 100 MOV R2 0 MUL R3 R1 R2 RET R3	2, 6, 8, 11, 13, 25, 33
EC7	MOV R1 100 MOV R2 0 DIV R3 R1 R2 RET R3	2, 6, 8, 13, 12, 17, 27, 33
EC8	MOV R1 100 MOV R2 20 DIV R3 R1 R2 RET R3	2, 6, 8,13,12, 17, 27, 33
EC9	MOV R1 100 MOV R2 20	2, 6, 8,13, 15, 17, 33, 35

	JMP -3 RET R3	
EC10	MOV R1 100 MOV R2 20 JMP 2 MOV R3 30 RET R2	2, 6, 8, 13, 15, 17, 33, 35
EC11	MOV R1 100 MOV R2 20 JMP 2 RET R3	2, 3, 6, 8, 13, 15, 17, 33, 35
EC12	MOV R1 100 MOV R2 0 JZ R2 -4 RET R1	2, 6, 8, 13,14, 33, 37, 38
EC13	MOV R1 100 MOV R2 0 JZ R2 1 RET R1	2, 6, 8, 13, 14, 17, 33, 37 , 38
EC14	MOV R1 100 MOV R2 0 JZ R2 10 RET R1	2, 3, 6, 8, 13,17, 14, 33, 37, 38
EC15	MOV R1 100 MOV R2 5 JZ R2 -4 RET R1	1, 2, 6, 8, 13, 17, 14, 33, 37, 39
EC16	MOV R1 -655888 MOV R2 5 RET R1	2, 6, 8, 13, 17, 33
EC17	MOV R1 100 MOV R2 5 RET R1	2, 6, 8, 13, 17, 33
EC18	MOV R1 777777 MOV R2 5 RET R1	2, 6, 8, 13, 17, 33
EC19	MOV R1 2	2, 6, 8, 13, 16, 17, 18, 29, 31, 33

	MOV R2 100 STR R1 -90 R2 LDR R4 R1 -90 RET R4	
EC20	MOV R1 2 MOV R2 100 STR R1 90 R2 LDR R4 R1 90 RET R4	2, 6, 8, 13, 17, 18, 29, 31, 33
EC21	MOV R1 10 MOV R2 100 STR R1 65532 R2 LDR R4 R1 65532 RET R4	2, 6, 8, 13, 17, 18, 29, 31, 33
EC22	MOV R1 2 MOV R2 100 STR R1 -90 R2 LDR R4 R1 -90 RET R4	2, 6, 8, 13, 17, 18, 29, 31, 33
EC23	MOV R1 2 MOV R2 100 STR R1 90 R2 LDR R4 R1 90 RET R4	2, 6, 8, 13, 17,18, 29, 31, 33
EC24	MOV R1 10 MOV R2 100 STR R1 65532 R2 LDR R4 R1 65532 RET R4	2, 6, 8, 13, 17, 18, 29, 31, 33
EC25	MOV R1 100 MOV R2 20 JMP 0 RET R3	2, 6, 8, 13, 15, 17, 33, 35

Table 3.0 Multiple-condition coverage in equivalence partitioning

With equivalence partitioning, 12 conditions are not met. Objective 4 is logically not possible, and the rest conditions are not met because of syntax correct assumption. In fact, in the program code there are some conditions to specifically check whether the length of the instruction syntax is correct or not.

Multiple-condition coverage result for equivalence partitioning : $\frac{\mathbf{27}}{\mathbf{39}} = \mathbf{69}\ \%$

- **Multiple-condition coverage in boundary analysis**

EC	Test Case	Meet with Objective
EC1	ADD R1 R2 R3	2, 6, 8, 9, 21
EC2A	ADD R0 R2 R3 RET R1	2, 6 , 8, 9, 17, 21
EC2B	ADD R1-1 R2 R3 RET R1	2, 6 , 8, 9, 17, 21
EC3A	ADD R31 R2 R3 RET R31	2, 6 , 8, 9, 17, 21
EC3B	ADD R32 R2 R3 RET R1	2, 6 , 8, 9, 17, 21
EC4	MOV R1 4 MOV R2 5 ADD R3 R1 R2 RET R3	2, 5, 6 , 8, 9, 13, 17, 21, 33
EC5	MOV R1 10 MOV R2 15 SUB R3 R1 R2 RET R3	2, 6 , 8, 10, 13, 17, 23, 33
EC6	MOV R1 100 MOV R2 0 MUL R3 R1 R2 RET R3	2, 6 , 8, 10, 13, 17, 23, 33
EC7A	MOV R1 100 MOV R2 0 DIV R3 R1 R2 RET R3	2, 6 , 8, 12, 13, 17, 27, 33
EC7B	MOV R1 100 MOV R2 -1 DIV R3 R1 R2 RET R3	2, 6 , 8, 12, 13, 17, 27, 33
EC8	MOV R1 100 MOV R2 20 DIV R3 R1 R2	2, 6 , 8, 12, 13, 17, 27, 33

	RET R3	
EC9A	MOV R1 100 MOV R2 20 JMP -3 RET R3	2, 6, 8, 13, 15, 17, 33, 35
EC9B	MOV R1 100 MOV R2 20 JMP -4 RET R3	1, 2, 6, 8, 13, 15, 17, 33, 35
EC10	MOV R1 100 MOV R2 20 JMP 2 MOV R3 30 RET R2	2, 6, 8, 13, 15, 17, 33, 35
EC11	MOV R1 100 MOV R2 20 JMP 3 MOV R3 30 RET R2	3, 6, 8, 13, 15, 17, 33, 35
EC12A	MOV R1 100 MOV R2 0 JZ R2 -3 RET R1	2, 6, 8, 13, 14, 17, 33, 37, 38
EC12B	MOV R1 100 MOV R2 0 JZ R2 -4	1, 2, 6, 8, 13, 14, 17, 33, 37, 38
EC13	MOV R1 100 MOV R2 0 ADD R3 R2 R1 JZ R2 1 RET R3	2, 6, 8, 9, 13, 14, 17, 21, 33, 37, 38
EC14	MOV R1 100 MOV R2 0 ADD R3 R2 R1 JZ R2 2 RET R3	2, 3, 6, 8, 9, 13, 14, 17, 21, 33, 37, 38
EC15	MOV R1 100 MOV R2 5	2, 6, 8, 13, 14, 17, 33, 37, 39

	JZ R2 -4 RET R1	
EC16A	MOV R1 -65535 MOV R2 5 RET R1	2, 6, 8, 13, 17, 33
EC16B	MOV R1 -65536 MOV R2 5 RET R1	2, 6, 8, 13, 17, 33
EC17	MOV R1 65535 MOV R2 5 RET R1	2, 6, 8, 13, 17, 33
EC18	MOV R1 65536 MOV R2 5 RET R1	2, 6, 8, 13, 17, 33
EC19	MOV R1 2 MOV R2 100 STR R1 -2 R2 LDR R4 R1 -2 RET R4	2, 6, 8, 13, 16, 17, 18, 29, 31, 33
EC20A	MOV R1 88 MOV R2 100 STR R1 -89 R2 LDR R4 R1 -89 RET R4	2, 6, 8, 13, 16, 17, 18, 29, 31, 33
EC20B	MOV R1 5 MOV R2 100 STR R1 65530 R2 LDR R4 R1 65530 RET R4	2, 6, 8, 13, 16, 17, 18, 29, 31, 33
EC21	MOV R1 3 MOV R2 100 STR R1 65533 R2 LDR R4 R1 65533 RET R4	2, 6, 8, 13, 16, 17, 18, 29, 31, 33
EC22	MOV R1 -38 MOV R2 100 STR R1 37 R2 LDR R4 R1 37 RET R4	2, 6, 8, 13, 16, 17, 18, 29, 31, 33

EC23A	MOV R1 -90 MOV R2 100 STR R1 90 R2 LDR R4 R1 90 RET R4	2, 6, 8, 13, 16, 17, 18, 29, 31, 33
EC23B	MOV R1 65535 MOV R2 100 STR R1 1 R2 LDR R4 R1 1 RET R4	2, 6, 8, 13, 16, 17, 18, 29, 31, 33
EC24	MOV R1 3 MOV R2 100 STR R1 65532 R2 LDR R4 R1 65532 RET R4	2, 6, 8, 13, 16, 17, 18, 29, 31, 33
EC25A	MOV R1 100 MOV R2 20 JMP 0 RET R3	2, 6, 8, 13, 15, 17, 33, 35
EC25B	MOV R1 100 MOV R2 20 JMP -1 RET R3	2, 6, 8, 13, 15, 17, 33, 35
EC25C	MOV R1 100 MOV R2 20 JMP 1 RET R1	2, 6, 8, 13, 15, 17, 33, 35

Table 4.0 Multiple-condition coverage in boundary analysis

With boundary analysis, 12 conditions are not met. Same with equivalence partitioning, objective 4 is logically not possible, and the rest conditions are not met because of syntax correct assumption. In fact, the program code has some if conditions to specifically check whether the length of the instruction syntax is correct or not. The result of multiple-condition coverage in boundary analysis is similar with multiple-condition coverage in equivalence partitioning because this coverage is only calculated within execute method which not cover other if condition that some test cases in boundary analysis could cover more (e.g validate offset, validate regs).

Multiple-condition coverage result for boundary analysis : $\frac{\mathbf{27}}{\mathbf{39}} = \mathbf{69}\ \%$

7. When it comes to check a program which specify a condition that has particular range like this program, it is found that boundary analysis is more effective than equivalence partitioning. Even though those two methods result in the same value for multiple-condition coverage as this coverage is only calculated in one particular function, more mutants can be killed with the test cases from boundary analysis instead of equivalence partitioning. Each mutant also can be killed with more test cases in boundary analysis rather than the test cases from from equivalence portioning. It is also supported with the fact that input considered in equivalence partitioning could be more broaden, but may not cover the off point and on point in the boundary analysis that could discover more faulty in the program.