The following table summarizes the SAYAC instructions implemented in the current version of compiler

Sl. No.	SAYAC Instruction	Assembly name	Shadow bit	Implemented (Yes/No)
1	LDR	ldr		Yes
2	LIR	lir		No
3	LDB	ldb		Yes
4	LIB	lib		No
5	STR	str		Yes
6	SIR	sir		No
7	STB	stb		Yes
8	SIB	sib		No
9	MSI	msi		Yes
10	MSI-shadow		R15[14]	No
11	MHI	mhi		Yes
12	CMR	cmr		Yes
13	CMRS		R15[10]	Yes
14	CMI	cmi		Yes
15	CMIS		R15[10]	Yes
16	JMR (save PC bit = 0)	jmr		Yes
17	JMRS (save PC bit = 1)	jmrs		Yes
18	JMB	jmb		No
19	JMI	jmi		No
20	BRC	brc		Yes
21	BRR	brr		No
22	ADR	adr		Yes
23	ADI	adi		Yes
24	ADIR		R15[15]	No
25	SUB	sub		Yes
26	SUI	sui		Yes
27	SUIR		R15[15]	No
28	AND	and		Yes
29	ANI	ani		Yes
30	ANIR		R15[15]	No
31	MUL	mul		No
32	MULS		R15[12]	No
33	MULSG		R15[13]	Yes
34	DIV	div		No
35	DIVS		R15[12]	No
36	DIVQ		R15[11]	Yes
37	NTR (1/2C bit = 0)	ntr1		Yes
38	NTR (1/2C bit = 1)	ntr2		No
39	NTD (1/2C bit = 0)	ntd1		Yes
40	NTD (1/2C bit = 1)	ntd2		No
41	SLR	slr		Yes
42	SAR	sar		Yes
43	SHI (LA bit = 0)	shil		Yes
44	SHI (LA bit = 1)	shia		Yes

Assumptions

1. Since shadow instruction and their corresponding normal instruction have the same Opcode, we have kept single assembly name for shadow and corresponding normal instruction. This was done to ensure one-to-one mapping between assembly language code and binary code. For example: CMR and CMRS both have `cmr` as their assembly name. The instruction to execute can be decided based on shadow bits' state(ON/OFF).

2. Initially shadow bit for compare instruction (R15[10]) is assumed to be ON as most of the time signed comparison is performed. Whenever unsigned comparison needs to be performed, the code is written to perform following task in order
   - R15[10] is turned OFF,
   - then `cmr` instruction is executed,
   - and then R15[10] is turned ON again.

3. Currently, only one variation of MUL instruction (MULSG) is implemented. So the corresponding shadow bit R15[13] is assumed to be ON. Similarly for DIV instruction shadow bit R15[11] is assumed to be ON.

msym Instruction

We have defined a special instruction "`msym`" which is similar to "`msi`", but instead of immediate value it loads address of symbol to the register.

Ex: `msym a1 swap -`

`swap` might be a function or a location in the program, the given command would find that location as an address number and load it into register a1.

This was required as actual program locations are unknown until they are loaded into the memory and we have to deal with symbols till then.

Logical OR, Logical XOR instruction

Logical OR , Logical XOR instruction are not directly supported by SAYAC microprocessor. But the compiler supports these instruction by expanding these instructions using a sequence of SAYAC supported instructions.

For example: `(a|b) = ~((~a) & (~b))`