Space	Instruction	Oopcode	Type	Syntax	Description	Psuedocode	Operand	Binary Visual	working
0	NOP	0b00000	system	< NOP >	no operation	-	0 0 0 0 0 0 0 0 0 0 0 0	⬛⬛⬛⬛⬛▨▨▨▨▨▨▨▨▨▨▨▨	
1	HLT	0b00001	system	< HLT >	halt	-	0 0 0 0 0 0 0 0 0 0 0 0	⬛⬛⬛⬛⬜▨▨▨▨▨▨▨▨▨▨▨▨	
2	ADD	0b00010	arithmetic	< ADD > < R1 > < R2 > < R3 >	addition	A + B => C	R1 R2 R3	⬛⬛⬛⬜⬛🟥🟥🟥🟥🟨🟨🟨🟨🟧🟧🟧🟧	
3	SUB	0b00011	arithmetic	< SUB > < R1 > < R2 > < R3 >	subtraction	A - B => C	R1 R2 R3	⬛⬛⬛⬜⬜🟥🟥🟥🟥🟨🟨🟨🟨🟧🟧🟧🟧	
4	XOR	0b00100	arithmetic	< XOR > < R1 > < R2 > < R3 >	bitwise xor	A ^ B => C	R1 R2 R3	⬛⬛⬜⬛⬛🟥🟥🟥🟥🟨🟨🟨🟨🟧🟧🟧🟧	
5	OR	0b00101	arithmetic	< OR > < R1 > < R2 > < R3 >	bitwise or	A | B => C	R1 R2 R3	⬛⬛⬜⬛⬜🟥🟥🟥🟥🟨🟨🟨🟨🟧🟧🟧🟧	
6	AND	0b00110	arithmetic	< AND > < R1 > < R2 > < R3 >	bitwise and	A & B => C	R1 R2 R3	⬛⬛⬜⬜⬛🟥🟥🟥🟥🟨🟨🟨🟨🟧🟧🟧🟧	
7	MUL	0b00111	arithmetic	< MUL > < R1 > < R2 > < R3 >	multiplication	A * B => C	R1 R2 R3	⬛⬛⬜⬜⬜🟥🟥🟥🟥🟨🟨🟨🟨🟧🟧🟧🟧	
8	S_OPR	0b01000	arithmetic	< S_OPR > < opr > < R2 > < R3 >	single opreator single var	f(B) => C	opr R2 R3	⬛⬜⬛⬛⬛▩▩▩▩🟨🟨🟨🟨🟧🟧🟧🟧	
9	CMP	0b01001	boolean	< CMP > < type > < R1 > < R2 >	compare type return bool	type(A - B)	type R1 R2	⬛⬜⬛⬛⬜■■■■🟥🟥🟥🟥🟨🟨🟨🟨	
10	LDI	0b01010	data	< LDI > < R1 > < 0bx8 >	load immediate	<0bx8> => A	R1 < 0bx8 >	⬛⬜⬛⬜⬛🟧🟧🟧🟧🟦🟦🟦🟦🟦🟦🟦🟦	
11	LD_M2R	0b01011	data	< LD_M2R > < R1 > < mem_add>	load mem to reg	mem => data => A	R2 memory address	⬛⬜⬛⬜⬜🟧🟧🟧🟧🟫🟫🟫🟫🟫🟫🟫🟫	
12	LD_R2M	0b01100	data	< LD_R2M > < R1 > < mem_add>	load reg to mem	A => data => mem	R3 memory address	⬛⬜⬜⬛⬛🟨🟨🟨🟨🟫🟫🟫🟫🟫🟫🟫🟫	
13	JMP	0b01101	address	< JMP > < ins_add >	jump to address	addr => ins	0 0 ins address	⬛⬜⬜⬛⬜⬛⬛🟪🟪🟪🟪🟪🟪🟪🟪🟪🟪	
14	JMP_IF	0b01101	address	< JMP_IF > < ins_add >	jump to address if true	addr => ins(if true)	0 1 ins address	⬛⬜⬜⬛⬜⬛⬜🟪🟪🟪🟪🟪🟪🟪🟪🟪🟪	
15	JMP_IFNOT	0b01101	address	< JMP_IFnot > < ins_add >	jump to address if not true	addr => ins(if false)	1 0 ins address	⬛⬜⬜⬛⬜⬜⬛🟪🟪🟪🟪🟪🟪🟪🟪🟪🟪	
16	CALL	0b10000	address	< CALL >	call	push ins addr	0 0 ins address	⬜⬛⬛⬛⬛⬛⬛🟪🟪🟪🟪🟪🟪🟪🟪🟪🟪	
17	RET	0b10000	address	< RET >	return	pop ins addr	0 1 ins address	⬜⬛⬛⬛⬛⬛⬜🟪🟪🟪🟪🟪🟪🟪🟪🟪🟪	
18	MOV	0b10010	data	< MOV > < R1 > < R2 >	mov data accross reg	A => B	R1 R2 0 0 0 0	⬜⬛⬛⬜⬛🟥🟥🟥🟥🟨🟨🟨🟨▨▨▨▨	no
19	WAIT	0b10011	time	< WAIT >	wait until	-	0 0 0 0 0 0 0 0 0 0 0 0	⬜⬛⬛⬜⬜▨▨▨▨▨▨▨▨▨▨▨▨	no
20	DELAY	0b10100	time	< DELAY > < setting_addr >	delay as per set	-	delay_address 0 0 0 0 0 0 0 0	⬜⬛⬜⬛⬛🔹🔹🔹🔹▨▨▨▨▨▨▨▨	no
21		0b10101						⬜⬛⬜⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
22		0b10110						⬜⬛⬜⬜⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
23		0b10111						⬜⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
24		0b11000						⬜⬜⬛⬛⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
25		0b11001						⬜⬜⬛⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
26		0b11010						⬜⬜⬛⬜⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
27		0b11011						⬜⬜⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
28		0b11100						⬜⬜⬜⬛⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
29		0b11101						⬜⬜⬜⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
30		0b11110						⬜⬜⬜⬜⬛⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	
31		0b11111						⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜	

Space	Sopr_instruction	S-Oopcode	Type	Syntax	Description	Psuedocode	Operand	Binary Visual	
0	SET_0	0b0000	single_opr	< S_OPR > < SET0 > < R2 > < R3 >	set to 0	A => 0	0 0 0 0 R2 R3	⬛⬜⬛⬛⬛⬛⬛⬛⬛🟨🟨🟨🟨🟧🟧🟧🟧	
1	RSH	0b0001	single_opr	< S_OPR > < RSH > < R2 > < R3 >	right shift	A >> 1 => B	0 0 0 1 R2 R3	⬛⬜⬛⬛⬛⬛⬛⬛⬜🟨🟨🟨🟨🟧🟧🟧🟧	
2	LSH	0b0010	single_opr	< S_OPR > < LSH > < R2 > < R3 >	left shift	A << 1 => B	0 0 1 0 R2 R3	⬛⬜⬛⬛⬛⬛⬛⬜⬛🟨🟨🟨🟨🟧🟧🟧🟧	
3	RCL	0b0011	single_opr	< S_OPR > < RCL > < R2 > < R3 >	right shift clock	A clockwise => B	0 0 1 1 R2 R3	⬛⬜⬛⬛⬛⬛⬛⬜⬜🟨🟨🟨🟨🟧🟧🟧🟧	
4	LCL	0b0100	single_opr	< S_OPR > < LCL > < R2 > < R3 >	left shift clock	A antic. => B	0 1 0 0 R2 R3	⬛⬜⬛⬛⬛⬛⬜⬛⬛🟨🟨🟨🟨🟧🟧🟧🟧	
5	NOT	0b0101	single_opr	< S_OPR > < NOT > < R2 > < R3 >	bitwise not	A ~ => B	0 1 0 1 R2 R3	⬛⬜⬛⬛⬛⬛⬜⬛⬜🟨🟨🟨🟨🟧🟧🟧🟧	
6	MUL_8-15	0b0110	single_opr	< S_OPR > < MUL > < R2 > < R3 >	remaining 8-15th bit	pre mul data => B	0 1 1 0 R2 R3	⬛⬜⬛⬛⬛⬛⬜⬜⬛🟨🟨🟨🟨🟧🟧🟧🟧	
7	R++	0b0111	single_opr	< S_OPR > < R++ > < R2 > < R3 >	increment by 1	A +1 => B	0 1 1 1 R2 R3	⬛⬜⬛⬛⬛⬛⬜⬜⬜🟨🟨🟨🟨🟧🟧🟧🟧	
8	R--	0b1000	single_opr	< S_OPR > < R-- > < R2 > < R3 >	decreemtn by 1	A - 1 => B	1 0 0 0 R2 R3	⬛⬜⬛⬛⬛⬜⬛⬛⬛🟨🟨🟨🟨🟧🟧🟧🟧	
9	NEG	0b1001	single_opr	< S_OPR > < NEG > < R2 > < R3 >	negative	-A => B	1 0 0 1 R2 R3	⬛⬜⬛⬛⬛⬜⬛⬛⬜🟨🟨🟨🟨🟧🟧🟧🟧	no
10	ODD	0b1010	single_opr	< S_OPR > < ODD > < R2 > < R3 >	returns 1 if odd else 0	bin(A) & 0b1 => B	1 0 1 0 R2 R3	⬛⬜⬛⬛⬛⬜⬛⬜⬛🟨🟨🟨🟨🟧🟧🟧🟧	
11	MSB	0b1011	single_opr	< S_OPR > < MSB > < R2 > < R3 >	most signf. bit	msb(A) => B	1 0 1 1 R2 R3	⬛⬜⬛⬛⬛⬜⬛⬜⬜🟨🟨🟨🟨🟧🟧🟧🟧	
12	LSB	0b1100	single_opr	< S_OPR > < LSB > < R2 > < R3 >	least signf. bit	lsb(A) => B	1 1 0 0 R2 R3	⬛⬜⬛⬛⬛⬜⬜⬛⬛🟨🟨🟨🟨🟧🟧🟧🟧	
13	LEN	0b1101	single_opr	< S_OPR > < LEN > < R2 > < R3 >	len of binary	len(A) => B	1 1 0 1 R2 R3	⬛⬜⬛⬛⬛⬜⬜⬛⬜🟨🟨🟨🟨🟧🟧🟧🟧	
14		0b1110	single_opr	< S_OPR > < > < R2 > < R3 >			1 1 1 0 R2 R3	⬛⬜⬛⬛⬛⬜⬜⬜⬛🟨🟨🟨🟨🟧🟧🟧🟧	
15		0b1111	single_opr	< S_OPR > < > < R2 > < R3 >			1 1 1 1 R2 R3	⬛⬜⬛⬛⬛⬜⬜⬜⬜🟨🟨🟨🟨🟧🟧🟧🟧	

Space	Type_instruction	T-Oopcode	Type	Syntax	Description	Psuedocode	Operand	Binary Visual	
0	FLAG	0b0000	cmp	< CMP > < FLAG > < R1 > < R2 >	returns {=,!=,...,even} as data	A- B	0 0 0 0 R1 R2	⬛⬜⬛⬛⬜⬛⬛⬛⬛🟥🟥🟥🟥🟨🟨🟨🟨	no
1	=	0b0001	cmp	< CMP > < = > < R1 > < R2 >	equal	A- B	0 0 0 1 R1 R2	⬛⬜⬛⬛⬜⬛⬛⬛⬜🟥🟥🟥🟥🟨🟨🟨🟨	
2	!=	0b0010	cmp	< CMP > < != > < R1 > < R2 >	not equal	A- B	0 0 1 0 R1 R2	⬛⬜⬛⬛⬜⬛⬛⬜⬛🟥🟥🟥🟥🟨🟨🟨🟨	
3	<	0b0011	cmp	< CMP > < < > < R1 > < R2 >	only -ve	A- B	0 0 1 1 R1 R2	⬛⬜⬛⬛⬜⬛⬛⬜⬜🟥🟥🟥🟥🟨🟨🟨🟨	
4	=>	0b0100	cmp	< CMP > < => > < R1 > < R2 >	not negative	A- B	0 1 0 0 R1 R2	⬛⬜⬛⬛⬜⬛⬜⬛⬛🟥🟥🟥🟥🟨🟨🟨🟨	
5	>	0b0101	cmp	< CMP > < > > < R1 > < R2 >	only +ve	A- B	0 1 0 1 R1 R2	⬛⬜⬛⬛⬜⬛⬜⬛⬜🟥🟥🟥🟥🟨🟨🟨🟨	
6	=<	0b0110	cmp	< CMP > < =< > < R1 > < R2 >	not positive	A- B	0 1 1 0 R1 R2	⬛⬜⬛⬛⬜⬛⬜⬜⬛🟥🟥🟥🟥🟨🟨🟨🟨	
7	odd	0b0111	cmp	< CMP > < ODD > < R1 > < R2 >	odd	A & 0b1	0 1 1 1 R1 0 0 0 0	⬛⬜⬛⬛⬜⬛⬜⬜⬜🟥🟥🟥🟥🟨🟨🟨🟨	
8	even	0b1000	cmp	< CMP > < EVEN > < R1 > < R2 >	even	A & 0b1	1 0 0 0 R1 0 0 0 0	⬛⬜⬛⬛⬜⬜⬛⬛⬛🟥🟥🟥🟥🟨🟨🟨🟨	
9		0b1001	cmp	< CMP > < > < R1 > < R2 >			1 0 0 1 R1 R2	⬛⬜⬛⬛⬜⬜⬛⬛⬜🟥🟥🟥🟥🟨🟨🟨🟨	
10		0b1010	cmp	< CMP > < > < R1 > < R2 >			1 0 1 0 R1 R2	⬛⬜⬛⬛⬜⬜⬛⬜⬛🟥🟥🟥🟥🟨🟨🟨🟨	
11		0b1011	cmp	< CMP > < > < R1 > < R2 >			1 0 1 1 R1 R2	⬛⬜⬛⬛⬜⬜⬛⬜⬜🟥🟥🟥🟥🟨🟨🟨🟨	
12		0b1100	cmp	< CMP > < > < R1 > < R2 >			1 1 0 0 R1 R2	⬛⬜⬛⬛⬜⬜⬜⬛⬛🟥🟥🟥🟥🟨🟨🟨🟨	
13		0b1101	cmp	< CMP > < > < R1 > < R2 >			1 1 0 1 R1 R2	⬛⬜⬛⬛⬜⬜⬜⬛⬜🟥🟥🟥🟥🟨🟨🟨🟨	
14		0b1110	cmp	< CMP > < > < R1 > < R2 >			1 1 1 0 R1 R2	⬛⬜⬛⬛⬜⬜⬜⬜⬛🟥🟥🟥🟥🟨🟨🟨🟨	
15		0b1111	cmp	< CMP > < > < R1 > < R2 >			1 1 1 1 R1 R2	⬛⬜⬛⬛⬜⬜⬜⬜⬜🟥🟥🟥🟥🟨🟨🟨🟨	