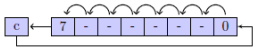
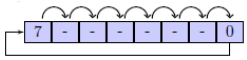
Chimera-2016-A Function Listing

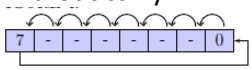
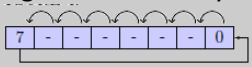
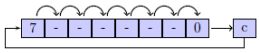
This is a help document for maintenance of the Chimera-2016-A emulator. Listed below is a table that contains the function names, a brief description, what the function returns, any parameters of the function, a list of opcode groups that use the function and the specific opcodes that use the function.

Function name	Description	Parameters	Returns	Opcode group using function	Opcodes using function directly
carry_test_ADC	Checks to see if the carry flag is set for ADC	inReg (WORD)	temp (WORD)	**Used by functions:** F_ADC	
carry_test_SBC	Checks to see if the carry flag is set for SBC	inReg (WORD)	temp (WORD)	**Used by functions:** F_SBC, F_SBI	
F_ADC	Adds registers with carry A + CF + R Same logic as F_ADD just adding the carry flag (if set) to the value.	reg1 (BYTE), reg2 (BYTE)	none (void)	ADC	0x31, 0x41, 0x51, 0x61, 0x71, 0x81
F_ADD	Adds registers A + R A 10101010 R 11001100 + 101110110. Because this is a 9-bit number it will set the carry flag and the value stored in A will be 01110110.	reg1 (BYTE), reg2 (BYTE)	none (void)	ADD	0x33, 0x43, 0x53, 0x63, 0x73, 0x83
F_AND	Bitwise ands registers A & R A 10101010 R 11001100 & 10001000	reg1 (BYTE), reg2 (BYTE)	none (void)	AND	0x37, 0x47, 0x57, 0x67, 0x77, 0x87
F_BIT	Register bit tested with accumulator Bitwise ands registers to compare	reg1 (BYTE), reg2 (BYTE)	none (void)	BIT	0x39, 0x49, 0x59, 0x69, 0x79,

	them, however doesn't set the value of the register.				0x89
F_CALL	For use with call opcodes. It puts the program counter into the stack pointer.	address (WORD)	none (void)	CCC, CCS, CNE, CEQ, CMI, CPL, CHI, CLE	0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29
F_CMP	Compares registers using subtraction, doesn't set any registers.	reg1 (BYTE), reg2 (BYTE)	none (void)	CMP	0x35, 0x45, 0x55, 0x65, 0x75, 0x85
F_COM_M	Negate memory This is the same as bitwise XOR with 0xFF. R 10011101 ~ 01100010	address (WORD)	none (void)	COM	0xA7, 0xB7, 0xC7
F_COM_R	Negate register This is the same as bitwise XOR with 0xFF. R 10011101 ~ 01100010	reg1(BYTE)	none (void)	COMA, COMB	0xD7, 0xE7
F_CPI	Data compared to accumulator. A - M	reg1 (BYTE)	none (void)	CPIA, CPIB	0x95, 0x96
F_DEC_I	Decrements the index register. Index register = Index register – 1.	reg1 (BYTE)	none (void)	DEY, DECX	0x03, 0x01
F_DEC_M	Decrements memory. Memory = Memory – 1.	address (WORD)	none (void)	DEC	0xA1, 0xB1, 0xC1
F_DEC_R	Decrements register. Register = Register – 1.	reg1 (BYTE)	none (void)	DECA, DECB	0xD1, 0xE1
F_INC_I	Increments the index register.	reg1 (BYTE)	none (void)	INCY, INCX	0x02, 0x04

	Index register = Index register + 1.				
F_INC_M	Increments memory. Memory = Memory + 1.	address (WORD)	none (void)	INC	0xA0, 0xB0, 0xC0
F_INC_R	Increments the register. Register = Register + 1.	reg1 (BYTE)	none (void)	INCA, INCB	0xD0, 0xE0
F_LOAD	Loads memory into register. Makes destination register equal to the memory address location.	reg (BYTE), address (WORD)	none (void)	LDAA, LDAB	0x0A, 0x1A, 0x2A, 0x3A, 0x4A, 0x5A, 0x0B, 0x1B, 0x2B, 0x3B, 0x4B, 0x5B
F_LOAD_I	Loads memory into index register. Makes destination index register equal to the memory address location.	reg (BYTE), address (WORD)	none (void)	LDX, LDY	0x0E, 0x1E, 0x2E, 0x3E, 0x4E, 0x5E, 0x0F, 0x1F, 0x2F, 0x3F, 0x4F, 0x5F
F_LOAD_S	Loads memory into Stack Pointer. Makes stack pointer equal to the memory address location.	address (WORD)	none (void)	LODS	0x20, 0x30, 0x40, 0x50, 0x60, 0x70
F_LSR_M	Shift right memory $0 \rightarrow \boxed{7 \quad 0} \rightarrow C$	address (WORD)	none (void)	LSR	0xA6, 0xB6, 0xC6
F_LSR_R	Shift right register $0 \rightarrow \boxed{7 \quad 0} \rightarrow C$	reg1 (BYTE)	none (void)	LSRA, LSRB	0xD6, 0xE6
F_MVI	Loads memory into register.	reg (BYTE)	none (void)	MVI	0x1C, 0x1D
F_OR	Bitwise or registers A | R	reg1 (BYTE), reg2 (BYTE)	none (void)	OR	0x36, 0x46, 0x56,

	A 10101010 R 11001100 | 11101110				0x66, 0x76, 0x86
F_ORI	Data bitwise inclusive or with accumulator A | M Same logic as F_OR just with data.	reg1 (BYTE)	none (void)	ORIA, ORIB	0x97, 0x98
F_POP	Pops the top of the stack into register.	reg1(BYTE)	none (void)	POP	0xBF, 0xCF, 0xEF, 0xFF
F_POP_FL	Pops the top of the stack into flags.	none (void)	none (void)	POP, Flags	0xDF
F_PUSH	Pushes register onto the stack.	reg1 (BYTE)	none (void)	PUSH	0xBE, 0xCE, 0xEE, 0xFE
F_PUSH_FL	Pushes the flags onto the stack.	none (void)	none (void)	PUSH, Flags	0xDE
F_RLC_M	Rotate memory left with carry  Bits that fall off the left end go through the carry, so the status of the carry flag gets put onto the other side.	address (WORD)	none (void)	RLC	0xA3, 0xB3, 0xC3
F_RLC_R	Rotate register left with carry  Bits that fall off the left end go through the carry, so the status of the carry flag gets put onto the other side.	reg1 (BYTE)	none (void)	RLCA, RLCB	0xD3, 0xE3
F_ROL_M	Rotate memory left without carry  Bits that fall off the left end return to the other side.	address (WORD)	none (void)	ROL	0xA8, 0xB8, 0xC8
F_ROL_R	Rotate accumulator left without carry 	reg1 (BYTE)	none (void)	ROLA, ROLB	0xD8, 0xE8

	Bits that fall off the left end return to the other side.				
F_RR_M	Rotate memory right without carry  Bits that fall off the right end return to the other side.	address (WORD)	none (void)	RR	0xA9, 0xB9, 0xC9
F_RR_R	Rotate accumulator right without carry  Bits that fall off the right end return to the other side.	reg1 (BYTE)	none (void)	RRA, RRB	0xD9, 0xE9
F_RRC_M	Rotate memory right with carry  Bits that fall off the right end go through the carry, so the status of the carry flag gets put onto the other side.	address (WORD)	none (void)	RRC	0xA2, 0xB2, 0xC2
F_RRC_R	Rotate register right with carry  Bits that fall off the right end go through the carry, so the status of the carry flag gets put onto the other side.	reg1 (BYTE)	none (void)	RRCA, RRCB	0xD2, 0xE2
F_SBC	Subtracts registers with carry A - CF - R Same logic as F_SUB just subtracting the carry flag (if set) the value.	reg1 (BYTE), reg2 (BYTE)	none (void)	SBC	0x32, 0x42, 0x52, 0x62, 0x72, 0x82
F_SBI	Data subtracted to accumulator with carry. A - CF - M Same logic as F_SBC just with data.	reg1 (BYTE)	none (void)	SBI	0x93, 0x94

F_SL_M	Arithmetic shift left memory $C \leftarrow \boxed{7 \quad 0} \leftarrow 0$ The bit that falls off the left sets the carry flag. The left is padded with zeros.	address (WORD)	none (void)	SAL	0xA4, 0xB4, 0xC4
F_SL_R	Arithmetic shift left accumulator $C \leftarrow \boxed{7 \quad 0} \leftarrow 0$ The bit that falls off the left sets the carry flag. The left is padded with zeros.	reg1 (BYTE)	none (void)	SALA, SALB	0xD4, 0xE4
F_SR_M	Arithmetic shift right memory $N \rightarrow \boxed{7 \quad 0} \rightarrow C$ The bit that falls off the right sets the carry flag. The status of the flag is put into the left side.	address (WORD)	none (void)	SAR	0xA5, 0xB5, 0xC5
F_SR_R	Arithmetic shift right accumulator $N \rightarrow \boxed{7 \quad 0} \rightarrow C$ The bit that falls off the right sets the carry flag. The status of the flag is put into the left side.	reg1 (BYTE)	none (void)	SARA, SARB	0xD5, 0xE5
F_STORE	Stores registers into memory.	reg (BYTE), address (WORD)	none (void)	STORA, STORB	0xBA, 0xCA, 0xDA, 0xEA, 0xFA, 0xBB, 0xCB, 0xDB, 0xEB, 0xFB
F_STORE_I	Stores index registers into memory	reg (BYTE), address (WORD)	none (void)	STOX, STOY	0xBC, 0xCC, 0xDC, 0xEC, 0xFC, 0xBD, 0xCD, 0xDD, 0xED, 0xFD

F_STORE_S	Stores Stack Pointer into memory	address(WORD)	none (void)	STOS	0x6A, 0x7A, 0x8A, 0x9A, 0xAA
F_SUB	Subtracts registers A 10101010 R 00100100 - 10000110	reg1 (BYTE), reg2 (BYTE)	none (void)	SBA, SAB	0xF4, 0xF6
F_XOR	Bitwise XOR registers A 10101010 R 11001100 ^ 01100110	reg1 (BYTE), reg2 (BYTE)	none (void)	XOR	0x38, 0x48, 0x58, 0x68, 0x78, 0x88
get_address	Sets value for type of address to be used the next function	address_type (integer) *This integer is from the variables declared above the function.*	address (WORD)	LDAA, LDAB, STORA, STORB, LDX, LDY, STOX,	0x1A, 0x2A, 0x3A, 0x4A, 0x5A, 0x1B, 0x2B, 0x3B, 0x4B, 0x5B, 0xBA, 0xCA, 0xDA, 0xEA, 0xFA, 0xBB, 0xCB, 0xDB, 0xEB, 0xFB, 0x0E, 0x1E, 0x2E, 0x3E, 0x4E, 0x5E, 0x0F, 0x1F, 0x2F, 0x3F, 0x4F, 0x5F, 0xBC, 0xCC,

					0xDC, 0xEC, 0xFC,
				STOY,	0xBD 0xCD, 0xDD, 0xED, 0xFD,
				LODS,	0x20, 0x30, 0x40, 0x50, 0x60, 0x70,
				STOS,	0x6A, 0x7A, 0x8A, 0x9A, 0xAA,
				JMP, JSR, JCC, JCS, JNE, JEQ, JMI, JPL, JHI, JLE, CCC, CCS, CNE, CEQ, CMI, CPL, CHI, CLE, INC,	0x10, 0x21, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0xA0, 0xB0, 0xC0,
				DEC,	0xA1, 0xB1, 0xC1,
				COM,	0xA7, 0xB7, 0xC7,
				SAL,	0xA4, 0xB4, 0xC4,
				SAR,	0xA5, 0xB5,

				ROL, RR, RLC, RRC, LSR	0xC5, 0xA8, 0xB8, 0xC8, 0xA9, 0xB9, 0xC9, 0xA3, 0xB3, 0xC3, 0xA2, 0xB2, 0xC2, 0xA6, 0xB6, 0xC6
set_flag_c	Sets the carry flag if the answer is larger than 8 bits.	inReg (WORD)	none (void)	**Used by functions:** set_flag_cnz	
set_flag_n	Sets the negative/sign bit.	inReg (BYTE)	none (void)	**Used by functions:** F_AND, F_OR, F_XOR, F_BIT, F_INC_R, F_INC_M, F_DEC_R, F_DEC_M, F_CPI, F_COM_R, F_ROL_M, F_ROL_R, F_RR_M, F_RR_R, F_RLC_R, F_RLC_M, F_RRC_R, F_RRC_M, F_LSR_M, F_LSR_R Set_flag_cnz CAY	0xF0
set_flag_z	Sets the zero flag if value is zero.	inReg (BYTE)	none (void)	**Used by functions:** F_AND, F_OR, F_XOR, F_BIT,	

				F_INC_R, F_INC_I, F_DEC_R, F_DEC_I, F_INC_M, F_DEC_M, F_ORI, F_ROL_M, F_ROL_R, F_RR_M, F_RR_R, F_RLC_R, F_RLC_M, F_RRC_R, F_RRC_M, F_LSR_M, F_LSR_R set_flag_cnz	
set_flag_cnz	Sets c, n and z flags by calling the other functions.	None (void)	inReg (WORD)	**Used by functions:** F_ADD, F_SUB, F_ADC, F_CMP, F_SBC, F_SBI, F_CPI, F_COM_M, F_COM_R, F_SL_M, F_SL_R, F_SR_M, F_SR_R	