Instrctions set decoding table (8bits instruction): [$B_7B_6B_5B_4B_3B_2B_1B_0$]

ALU Op : 3bits [$B_7B_6B_5$] Access Mode : 3bits [$B_4B_3B_2$] Bus Access : 2bits [B_1B_0]

Instructions (ALU Op)	Bus Access	Access Mode							
		000 [D],AC	**001** [X],AC	**010** [Y,D],AC	**011** [Y,X],AC	**100** [D],X	**101** [D],Y	**110** [D],OUT	**111** [Y,X++],OUT
LD[000]	00(D)	MV AC,$dd	MV AC,$dd	MV AC,$dd	MV AC,$dd	MV X,$dd	MV Y,$dd	MV OUT,$dd	MV OUTxx,$dd
	01(mem)	LD AC,[$dd]	LD AC,[X]	LD AC,[Y,$dd]	LD AC,[Y,X]	LD X,[$dd]	LD Y,[$dd]	LD OUT,[$dd]	LD OUTxx,[Y,X]
	10(AC)	NOP	NOP	NOP	NOP	MV X,AC	MV Y,AC	MV OUT,AC	MV OUTxx,AC
	11(IN)	INPUT AC	INPUT AC	INPUT AC	INPUT AC	INPUT X	INPUT Y	INPUT OUT	INPUT OUTxx
AND[001]	00(D)	AND AC,$dd	AND AC,$dd	AND AC,$dd	AND AC,$dd	AND X,$dd	AND Y,$dd	AND OUT,$dd	AND OUTxx,$dd
	01(mem)	AND AC,[$dd]	AND AC,[X]	AND AC,[Y,$dd]	AND AC,[Y,X]	AND X,[$dd]	AND Y,[$dd]	AND OUT,[$dd]	AND OUTxx,[Y,X]
	10(AC)	NOP	NOP	NOP	NOP	MV X,AC	MV Y,AC	MV OUT,AC	MV OUTxx,AC
	11(IN)	AND AC,IN	AND AC,IN	AND AC,IN	AND AC,IN	AND X,IN	AND Y,IN	AND OUT,IN	AND OUTxx,IN
OR[010]	00(D)	OR AC,$dd	OR AC,$dd	OR AC,$dd	OR AC,$dd	OR X,$dd	OR Y,$dd	OR OUT,$dd	OR OUTxx,$dd
	01(mem)	OR AC,[$dd]	OR AC,[X]	OR AC,[Y,$dd]	OR AC,[Y,X]	OR X,[$dd]	OR Y,[$dd]	OR OUT,[$dd]	OR OUTxx,[Y,X]
	10(AC)	NOP	NOP	NOP	NOP	MV X,AC	MV Y,AC	MV OUT,AC	MV OUTxx,AC
	11(IN)	OR AC,IN	OR AC,IN	OR AC,IN	OR AC,IN	OR X,IN	OR Y,IN	OR OUT,IN	OR OUTxx,IN
XOR[011]	00(D)	XOR AC,$dd	XOR AC,$dd	XOR AC,$dd	XOR AC,$dd	XOR X,$dd	XOR Y,$dd	XOR OUT,$dd	XOR OUTxx,$dd
	01(mem)	XOR AC,[$dd]	XOR AC,[X]	XOR AC,[Y,$dd]	XOR AC,[Y,X]	XOR X,[$dd]	XOR Y,[$dd]	XOR OUT,[$dd]	XOR OUTxx,[Y,X]
	10(AC)	CLR AC	CLR AC	CLR AC	CLR AC	CLR X	CLR Y	CLR OUT	CLR OUTxx
	11(IN)	XOR AC,IN	XOR AC,IN	XOR AC,IN	XOR AC,IN	XOR X,IN	XOR Y,IN	XOR OUT,IN	XOR OUTxx,IN
ADD[100]	00(D)	ADD AC,$dd	ADD AC,$dd	ADD AC,$dd	ADD AC,$dd	ADD X,$dd	ADD Y,$dd	ADD OUT,$dd	ADD OUTxx,$dd
	01(mem)	ADD AC,[$dd]	ADD AC,[X]	ADD AC,[Y,$dd]	ADD AC,[Y,X]	ADD X,[$dd]	ADD Y,[$dd]	ADD OUT,[$dd]	ADD OUTxx,[Y,X]
	10(AC)	SL AC	SL AC	SL AC	SL AC	SL X	SL Y	SL OUT	SL OUTxx
	11(IN)	ADD AC,IN	ADD AC,IN	ADD AC,IN	ADD AC,IN	ADD X,IN	ADD Y,IN	ADD OUT,IN	ADD OUTxx,IN
SUB[101]	00(D)	SUB AC,$dd	SUB AC,$dd	SUB AC,$dd	SUB AC,$dd	SUB X,$dd	SUB Y,$dd	SUB OUT,$dd	SUB OUTxx,$dd
	01(mem)	SUB AC,[$dd]	SUB AC,[X]	SUB AC,[Y,$dd]	SUB AC,[Y,X]	SUB X,[$dd]	SUB Y,[$dd]	SUB OUT,[$dd]	SUB OUTxx,[Y,X]
	10(AC)	CLR AC	CLR AC	CLR AC	CLR AC	CLR X	CLR Y	CLR OUT	CLR OUTxx
	11(IN)	SUB AC,IN	SUB AC,IN	SUB AC,IN	SUB AC,IN	SUB X,IN	SUB Y,IN	SUB OUT,IN	SUB OUTxx,IN
ST[110]	00(D)	ST [$dd],$dd	ST [X],$dd	ST [Y,$dd],$dd	ST [Y,X],$dd	-*	-*	ST [$dd],$dd	ST [Y,X++],$dd***
	01(mem)	IO [$dd]**	IO [X]**	IO [Y,$dd]**	IO [Y,X]**	IO $dd**	CE**	IO AC**	IO IN**
	10(AC)	ST [$dd],AC	ST [X],AC	ST [Y,$dd],AC	ST [Y,X],AC	-*	-*	ST [$dd],AC	ST [Y,X++],AC***
	11(IN)	ST [$dd],IN	ST [X],IN	ST [Y,$dd],IN	ST [Y,X],IN	-*	-*	ST [$dd],IN	ST [Y,X++],IN***
Bcc[111]	00(D)	JMP $dd	BGT $dd	BLT $dd	BNE $dd	BEQ $dd	BGE $dd	BLE $dd	B $dd
	01(mem)	JMP [$dd]	BGT [$dd]	BLT [$dd]	BNE [$dd]	BEQ [$dd]	BGE [$dd]	BLE [$dd]	B [$dd]
	10(AC)	JMP AC	BGT AC	BLT AC	BNE AC	BEQ AC	BGE AC	BLE AC	B AC
	11(IN)	JMP IN	BGT IN	BLT IN	BNE IN	BEQ IN	BGE IN	BLE IN	B IN

* those instructions can be executed on Gigatron but they are not relevant.
** those instructions cannot be executed on Gigatron, so they are salvaged on Megatron for input/output extended mechanism.
*** those 3 instructions are not simulated correctly, should not be used on simulation.

Instructions set:

Instruction Set	Syntax bar character (|) means or	Description bar character (|) means or
MV	MV reg_{dst},reg_{src}|$dd	move : reg_{dst} ← reg_{src}|$dd
LD	LD reg_{dst},mem_{src}	load : reg_{dst} ← mem_{src}
ST	ST mem_{dst},reg_{src}|$dd	store : mem_{dst} ← reg_{src}|$dd
INPUT	INPUT reg_{dst}	input : reg_{dst} ← reg_{IN}
AND	AND reg_{dst},reg_{src}|mem_{src}|$dd	and : reg_{dst} ← reg_{AC} and (reg_{src}|mem_{src}|$dd)
OR	OR reg_{dst},reg_{src}|mem_{src}|$dd	or : reg_{dst} ← reg_{AC} or (reg_{src}|mem_{src}|$dd)
XOR	XOR reg_{dst},reg_{src}|mem_{src}|$dd	xor : reg_{dst} ← reg_{AC} xor (reg_{src}|mem_{src}|$dd)
ADD	ADD reg_{dst},reg_{src}|mem_{src}|$dd	addition : reg_{dst} ← reg_{AC} + (reg_{src}|mem_{src}|$dd)
SUB	SUB reg_{dst},reg_{src}|mem_{src}|$dd	substraction : reg_{dst} ← reg_{AC} - (reg_{src}|mem_{src}|$dd)
SL	SL reg_{dst}	Shift Left : reg_{dst} ← AC << 1 (1 bit shift)
CLR	CLR reg_{dst}	clear : reg_{dst} ← 0
JMP	JMP reg_{src}|$mem_{$dd}$\|$dd	jump : unconditionnal jump to 16 bits address with segment Y and offset reg_{src}|$mem_{$dd}$\|$dd
B	B reg_{src}|$mem_{$dd}$\|$dd	branch : unconditionnal jump to 8 bits address reg_{src}|$mem_{$dd}$| $dd within the same segment
BEQ	BEQ reg_{src}|$mem_{$dd}$\|$dd	branch if equal : conditionnal jump to 8 bits address reg_{src}| $mem_{$dd}$\|$dd within the same segment if A == B
BNE	BNE reg_{src}|$mem_{$dd}$\|$dd	branch if not equal : conditionnal jump to 8 bits address reg_{src}|$mem_{$dd}$\|$dd within the same segment if A != B
BGT	BGT reg_{src}|$mem_{$dd}$\|$dd	branch if greater than : conditionnal jump to 8 bits address reg_{src}|$mem_{$dd}$\|$dd within the same segment if A > B
BGE	BGE reg_{src}|$mem_{$dd}$\|$dd	branch if greater than or equal : conditionnal jump to 8 bits address reg_{src}|$mem_{$dd}$\|$dd within the same segment if A >= B
BLT	BLT reg_{src}|$mem_{$dd}$\|$dd	branch if less than : conditionnal jump to 8 bits address reg_{src}| $mem_{$dd}$\|$dd within the same segment if A < B
BLE	BLE reg_{src}|$mem_{$dd}$\|$dd	branch if less than or equal : conditionnal jump to 8 bits address reg_{src}|$mem_{$dd}$\|$dd within the same segment if A <= B
NOP	NOP	do nothing
IO*	IO reg_{src}|$mem_{$dd}$\|$dd	Load to Input/Output register : reg_{ioc} ← reg_{src}|$mem_{$dd}$\|$dd
CE*	CE	Control Signal Enable : bring out control signals from reg_{ioc}

* specific to Megatron.

Megatron:

Megatron is a variant of Gigatron, it was designed to be compatible with Gigatron, and to be more compact containing less chips. It uses a specific input/output unit to extend the input/output capabilities of Gigatron, thus a keyboard or other devices can be easily added. The programmable register IOC (Input/Output Control) is responsible for controlling whitch device must be used, it's fourth lower nibbles (4 bits) are used to choose thich input to read from in read instructions, and the uppers which to write at in output instructions. This register is also used by CE instruction to emit devices control signals like hsync and vsync for VGA, this register must be programmed depending on how devices are connected to Megatron.

Usage rules and conditions:

#	Title	Description
1	user registers	4 user registers : AC, X, Y, OUT (IN register can't be modified)
2	AC register	the accumulator, used for general purpose, it is implicitly the 1st operand in every arithmetic/logic operation
3	X register	used like offset within memory addressing modes, can't be used like operand in arithmetic/logic operations
4	Y register	used like segment within memory addressing modes, can't be used like operand in arithmetic/logic operations
5	IN register	used for input operations
6	OUT register	used for output operations, can't be used like operand in arithmetic/logic operations
7	OUTxx register	is OUT register executing X++
8	$dd	immediate or literal value ex : MV AC,5 (5 is $dd)
9	reg_{dst}	element of { AC,X,Y,OUT,OUTxx} (IN can't be destination register)
10	reg_{src}	element of { AC,IN} (X,Y,OUT can't be source register)
11	memory cell	8 bits memory cells are used to store values
12	accessing memory	the syntax [address] is used to access memory cells
13	memory address	address can be on 8 bits or 16 bits (segment,offset), multiple addressing modes exist
14	mem_{src}	memory cell used as a source, addressing modes allowed depends on reg_{dst} on the same instruction : 4 possible addressing modes if reg_{dst} = AC : [$dd],[X],[Y,$dd],[Y,X] 1 possible addressing mode if reg_{dst} = X or Y or OUT : [$dd] 1 possible addressing mode if reg_{dst} = OUTxx : [Y,X]
15	mem_{dst}	used only within LD, addressing modes are : [$dd],[X],[Y,$dd],[Y,X],[Y,X++]
16	$mem_{\$dd}$	memory source used with the addressing mode [$dd]
17	arithmetic/logic operation instructions	include AND, OR, XOR, ADD, SUB, SL, CLR. AC is implicitly the first operand, thus it is not inserted in the instruction syntax.
18	SL	Shift Left : it's the left shift instruction of the AC by 1 bit. AC is implicit in the instruction, thus only the destination is indicated in the instruction
19	bitwise operations	**and**,**or**,**xor** are bitwise logical operations (bit by bit)
20	JMP instruction	jump instructions use 16 bits of address to perform a long jump, Y register is implicitly added to the address like segment part
21	branch instructions	include B, BEQ, BNE, BGT, BGE, BLT, BTE, use 8 bits of address to perform short jumps within the same segment of 256 bytes
22	comparative branches	include BEQ, BNE, BGT, BGE, BLT, BTE, they need subtraction instruction A-B before their execution, the result of the subtraction determines the result of the condition (A is AC and B depend on the subtract instruction)
23	NOP instruction	this instruction does nothing, but must always be added after a jump or a branch instruction to avoid the pipeline hazard. Not mandatory on Megatron.
24	Megatron*	it is an instruction retro compatible version of Gigatron with extended input/output mechanism.
25	reg_{ioc} register*	used for 2 purposes, used for holding multiple input/output register write enable signals. And to emit 8 control signals for different devices.
26	IO and CE instruction*	used with reg_{ioc} on Megatron Input/Output Unit (IOU) to deal with multiple devices.

* specific to Megatron.