CH552 Instruction Set
Quick Reference Card

This card lists all CH552 instructions, with CH552 specific cycles.

Key to Tables

<dest8>/<src8>	argument specifying an 8-bit value (see notes for valid arguments)	addr16	16-bit absolute memory address
<dest1>/<src1>	argument specifying a single bit (see notes for valid arguments)	addr11	11-bit address that is within the same 2K memory block as the origin address
direct	byte memory address or label	rel	signed 8-bit address relative to the next instruction
bit	bit memory address or specifier	@Ri	absolute 16-bit memory address contained in a register, where i is 0 or 1.
#data8	8-bit immediate value	#data16	16-bit immediate value

Operation		Assembler	Flags	Action	§	Notes
Move	Move Byte	MOV <dest8>, <src8>		<dest8> = <src8>	1-3	Args: (A, {Rn, direct, @Ri, #data}), (Rn, {A, direct, #data}), (direct, {A, Rn, direct, @Ri, #data}), (@Ri, {A, direct, #data})
	Move Bit	MOV <dest1>, <src1>		<dest1> = <src1>	2	Args: (C, bit), (bit, C)
	Load DPTR 16-bit Const	MOV DPTR, #data16		DPTR = #data16	3	Only instruction that moves 16-bits of data at once
	Move Code Byte	MOVC A, @A+<base-reg>		A = @(A + <base-reg>)	5*	Args: (A, @A + DPTR), (A, @A + PC); For PC, counts from next instruction
	Move External	MOVX <dest8>, <src8>		<dest8> = <src8>	1	Args: (A, @Ri), (A, @DPTR), (@Ri, A), (@DPTR, A)
	XRAM Fast Copy	DB 0A5H		@DPTR1 = A	1	XBUS_AUX determines which DPTR is active for other instructions
Add Subtract	Add	ADD A, <src8>	C AC OV	A = A + <src8>	1-2	Args: (A, Rn), (A, direct), (A, @Ri), (A, #data8)
	Add with Carry	ADDC A, <src8>	C AC OV	A = A + C + <src8>	1-2	Args: (A, Rn), (A, direct), (A, @Ri), (A, #data8)
	Subtract with Borrow	SUBB A, <src8>	C AC OV	A = A - C - <src8>	1-2	Args: (A, Rn), (A, direct), (A, @Ri), (A, #data8); CLR C first for no borrow
Multiply Divide	Multiply	MUL AB	C OV	A = (A * B)[7:0]; B = (A * B)[15:8]	1	Args: (AB); OV set if product > 255, else cleared; Carry always cleared
	Divide	DIV AB	C OV	A = A / B; B = A % B	4	Args: (AB); OV set on divide by zero, else cleared; Carry always cleared
Logical	Logical-AND for Byte	ANL <dest8>, <src8>		<dest8> = <dest8> AND <src8>	1-3	Args: (A, Rn), (A, direct), (A, @Ri), (A, #data8), (direct, A), (direct, #data8)
	Logical-AND for Bit	ANL C,[/]<src1>	C	C = C AND [NOT]<src1>	2	Args: (C, bit), (C, /bit)
	Logical-OR for Byte	ORL <dest8>, <src8>		<dest8> = <dest8> OR <src8>	1-3	Args: (A, Rn), (A, direct), (A, @Ri), (A, #data8), (direct, A), (direct, #data8)
	Logical-OR for Bit	ORL C, [/]<src1>	C	C = C OR [NOT]<src-bit>	2	Args: (C, bit), (C, /bit)
	Logical-XOR for Byte	XRL <dest8>, <src8>		<dest8> = <dest8> XOR <src8>	1-3	Args: (A, Rn), (A, direct), (A, @Ri), (A, #data8), (direct, A), (direct, #data8)
	Clear Accumulator	CLR A		A = 0	1	
	Clear Bit	CLR bit	C	bit = 0	1-2	Args: (C), (bit); Only affects carry flag when arg is C
	Complement Accumulator	CPL A		A = NOT A	1	
	Complement Bit	CPL bit	C	bit = NOT bit	1-2	Args: (C), (bit); Only affects carry flag when arg is C
	Set Bit	SETB bit	C	bit = 1	1-2	Args: (C), (bit); Only affects carry flag when arg is C
Rotate	Rotate Accumulator Left	RL A		A = (A[6:0] << 1) OR A[7]	1	
	Rotate Acc Left thru C	RLC A	C	A = (A[6:0] << 1) OR C; C = A[7]	1	
	Rotate Accumulator Right	RR A		A = (A[0] << 7) OR A[7:1]	1	
	Rotate Acc Right thru C	RRC A	C	A = (C << 7) OR A[7:1]; C = A[0]	1	

Binary-Coded Decimal	BCD Adjust Acc for ADD	DA A	C	if ((A[0:3] > 9) OR (AC == 1))
				then A[3:0] = A[3:0] + 6
				if ((A[4:7] > 9) OR (C == 1))
				then A[7:4] = A[7:4] + 6
					1	After using ADD or ADDC to add a pair of BCD formatted values, this will restore the result to BCD format.

	Exchange Digit	XCHD A, @Ri		A[3:0] = (@Ri)[3:0] (@Ri)[3:0] = A[3:0]	1	Args: (A, @Ri); Exchanges the low order digit in BCD or hexadecimal value
	Swap Nibbles in Acc	SWAP A		A = (A[3:0] << 4) OR A[7:4]	1	Swaps upper and lower digit in BCD or hexadecimal value

CH552 Instruction Set
Quick Reference Card

Operation		Assembler	§	Action	Notes
Push/Pop	Push onto Stack	`PUSH direct`	2	SP = SP + 1, @SP = direct	
	Pop from Stack	`POP direct`	2	direct = @SP, SP = SP - 1	
Reverse	Exchange Accumulator with Byte	`XCH A, <byte>`	1-2	A = <byte>, <byte> = A	Args: (A, Rn), (A, direct), (A, @Ri)
Increment Decrement	Increment	`INC <byte>`	1-2	<byte> = <byte> + 1	Args: (A), (Rn), direct, @Ri
	Increment	`INC DPTR`	1	DPTR = DPTR + 1	Only 16-bit register that can be incremented
	Decrement	`DEC <byte>`	1-2	<byte> = <byte> - 1	Args: (A), (Rn), direct, @Ri
Branch	Absolute Jump	`AJMP addr11`	4*	PC = PC + 2, PC[10:0] = addr11	
	Long Jump	`LJMP addr16`	5*	PC = addr16	
	Short Jump	`SJMP rel`	4*	PC = PC + 2, PC = PC + rel	
	Jump Indirect	`JMP @A+DPTR`	3*	PC = @(A + DPTR)	

Conditional Branch	Compare and Jump if Not Equal	`CJNE <dest8>, <src8>, rel`	3, 5*	PC = PC + 3
				if (<dest8> != <src8>) then PC = PC + rel
				if (<dest8> < <src8>) then C = 1 else C = 0;
				Args: (A, direct, rel), (A, #data8, rel), (Rn, #data8, rel), (@Ri, #data8, rel)	

	Decrement and Jump if Not Zero	`DJNZ <byte>, rel`	2,4*-3,5*	PC = PC + 2 <byte> = <byte> - 1 if (<byte> != 0) then PC = PC + rel Args: (Rn, rel), (direct, rel)	
	Jump if Bit Set	`JB bit, rel`	3,5*	PC = PC + 3, if (bit == 1) then PC = PC + rel	
	Jump if Bit Not Set	`JNB bit, rel`	3,5*	PC = PC + 3, if (bit == 0) then PC = PC + rel	
	Jump if Bit Set and Clear Bit	`JBC bit, rel`	3,5*	PC = PC + 3, if (bit == 1) then (bit = 0, PC = PC + rel)	
	Jump if Carry Set	`JC rel`	2,4*	PC = PC + 2, if (C == 1) then PC = PC + rel	
	Jump if Carry Not Set	`JNC rel`	2,4*	PC = PC + 2, if (C == 0) then PC = PC + rel	
	Jump if Accumulator Not Zero	`JNZ rel`	2,4*	PC = PC + 2, if (A != 0) then PC = PC + rel	
	Jump if Accumulator Zero	`JZ rel`	2,4*	PC = PC + 2, if (A == 0) then PC = PC + rel	
Subroutine Call / Return	Absolute Call	`ACALL addr11`	4*	PC =: PC + 2, SP = SP + 2, (SP) = PC, PC[10:0] = addr11	
	Long Call	`LCALL addr16`	5*	PC = PC + 3, SP = SP + 2, (SP) = PC, PC = addr16	
	Return from Subroutine	`RET`	4*	PC = (@SP)[15:0], SP = SP - 2	
	Return from Interrupt	`RETI`	4*	PC = (@SP)[15:0], SP = SP - 2; Reenables interrupts of equal or lower priority	
Nop	No Operation	`NOP`	1	PC = PC + 1	Does nothing for one cycle

Instruction Cycles

Range cycle lengths: Some arguments increase the instruction size. Each additional byte adds 1 cycle. (See Instruction Size chart.)

* Starred lengths follow these rules:
- For conditional jumps, if no jump occurs, the number of cycles is the number of bytes in the instruction (the first, non-starred value shown).
- For all jumps, if the target address (return point for RET/RETI) is odd, add 1 cycle.
- For MOVC/JMP @A+DPTR/JB/JBN/JBC/CJNE A, direct, rel/DJNZ, if the address of this instruction is odd, add 1 cycle.
- For MOVC, if the address of the next instruction is odd, add 1 cycle. (Because MOVC is always 1 byte, either this instruction or the next (but not both) will be odd, which means it should always be 6 cycles, but the datasheet does not mention this.)

Instruction Size

Instruction size depends entirely on arguments. All instructions use 1 byte for the opcode. The arguments listed below add one or two additional bytes for each appearance in the argument list used.

1 Byte Args	2 Byte Args
addr11 direct #data8 rel bit	addr16 #data16

Notes

Some Actions are simplified, with multiple steps reduced to a single one. For example, LCALL and ACALL store the 16-bit return address by incrementing the SP and pushing one byte of it twice, rather than incrementing once by two and then writing 16 bytes all at once. This should make no difference to functionality.

The information used here comes from WCH's CH552 ISA document, CH55X 汇编指令说明, that can be found here: https://www.wch.cn/uploads/file/20210617/1623896732811060.pdf The information in this document is provided explicitly for educational use, under fair-use doctrine, and may include information copyrighted by WCH. This document was not created or published by WCH and is not endorsed by WCH.