RISC8B Core MCU Instruction Set and Assembly Tool

Version: 2B

https://wch-ic.com

1. Overview

WCH-RISC8B is an 8-bit MCU core, and RISC8B utilizes some bit transfer instructions based on RISC8B. All instructions are 16 bits in width, and instructions consist of operation codes and operands. RISC8B has 66 instructions, which are divided into control class, byte-oriented operation class, constant operation class, bit-oriented operation class, and transfer class according to the operation objects.

2. Instruction Set

F represents SFR or RAM register, and the valid values are 0x00-0xFF;
F represents extended address SFR or RAM register, and the valid values are 0x000-0x1FF;
A is the working register, C is the carry flag and Z is the zero flag;
D represents the destination register, and the valid values are 0, 1, a, f, empty (meaning that there is no such operand);
When d=0 or d=A, the operation result is put into the A register; when d=1 or d=F or d is empty, the operation result is put into the F register;
B represents bit selection, and valid value is 0-7;
A represents independent bit selection, with valid values of 0-3, 0 corresponding to C, and 1-3 corresponding to self-defined independent bits;
K2 represents a 2-bit constant, and the valid value is 0-3;
K or k8 represents an 8-bit constant, and the valid value is 0x00-0xFF;
K7 represents a 7-bit constant with valid values of 0x00-0x7F;
K9 represents a 9-bit constant with valid values of 0x000-0x1FF;
K10 represents a 10-bit constant, and the valid value is 0x000-0x3FF;
K12 represents a 12-bit constant with the valid value of 0x000-0xFFF;
TOS represents the Top of the Stack and refers to the current stack unit;
1#ff and 2#ff refer to the self-defined 1# and 2# fast SFR registers respectively;
1#bf and 2#bf respectively refer to the self-defined 1# and 2# bit operation SFR registers;
{*, *} represents the set, f[b] represents the b bit of f, [@a] represents the register pointed by address a, and (k) represents the execution with parameter k.
Instructions shaded in gray are added or redefined for the RISC8B kernel, which may not be supported by the RISC8A core.

Binary instruction code	HEX	Instruction Description	Mnemonic	Operand	Perform operations	Influence state
Control class (18)						
00000000 000000xx	0000	Empty operation	NOP		None	None
00000000 000010xx	0008	Clear watchdog timer	CLRWDT		O→WatchDogTimer	None

00000000 00001100	000C	Sleep	SLEEP		clock stop	None
00000000 000011kk	000C	Enter the specified sleep mode	SLEEPX	k2	k2→SleepMode, clock stop (k2)	None
00000000 00010bbb	0010	The bit waiting for b selection is 1.	WAITB	b	wait bit[b]==1	None
00000000 00010000	0010	Waiting to read from the parallel port	WAITRD		wait SB_IF_READ==1	None
00000000 00010100	0014	Wait for writing from a parallel port or SPI interrupt.	WAITWR WAITSPI		wait SB_IF_WRITE==1 wait SB_IF_SPI==1	None
00000000 00011000	0018	Reading code from program space	RDCODE		ROM_CODE→{SFR, A}	None
00000000 000110kk	0018	Reading code from program space	RCODE	k2	ROM_CODE(k2)→{SFR, A}	None
00000000 00011100	001C	Write code into program space	WRCODE		{SFR}→ROM_CODE	Short pause
00000000 000111kk	001C	Custom actions with parameters	EXEC	k2	custom operation (k2)	Customize
00000000 001000xx	0020	Save state to stack	PUSHAS		{A, Z, C,..}→TOS	None
00000000 001001xx	0024	Restore the state from the stack	POPAS		TOS→{A, Z, C,..}	Z, C
00000000 001010xx	0028	Save indirect addressing register 2 and page bits to the stack.	PUSHA2		{SFR_INDIR_ADDR2, INDIR_RAM_PAGE}→TOS	None
00000000 001011xx	002C	Recover indirectly addressed register 2 and page bits from the stack.	POPA2		TOS→{INDIR_RAM_PAGE, SFR_INDIR_ADDR2}	INDIR_RAM_PAGE_BIT
00000000 001100xx	0030	Subroutine return	RET		TOS→PC	None
00000000 001101xx	0034	The subroutine returns to zero state	RETZ		TOS→PC,1→Z	Z
00000000 001110xx	0038	Interrupt return	RETIE		TOS→PC,1→IE_global	None
Byte-oriented operation class (16)						
00000000 000001xx	0004	A clearing	CLRA		0xO0→A,1→Z	Z
00000001 ffffffff	01ff	f clearing	CLR	f	0xO0→f,1→Z	Z
0001000F FFFFFFFF	10FF	A is transferred to F, and bit 9 specifies the page bit of the	MOVA	F	A→F	None

		extended address.				
000d001F FFFFFFFF	d2FF	Transfer F to D, bit 9 specifies the page bit of the extended address.	MOV	F, d	F→d	Z
000d0100 ffffffff	d4ff	f increasing	INC	f, d	f+1→d	Z
000d0101 ffffffff	d5ff	f decreasing	DEC	f, d	f-1→d	Z
000d0110 ffffffff	d6ff	f increases, and zero jumps.	INCSZ	f, d	f+1→d, skip if Z==1	None
000d0111 ffffffff	d7ff	f decreases, and zero jumps.	DECSZ	f, d	f-1→d, skip if Z==1	None
000d1000 ffffffff	d8ff	f high and low nibble exchange	SWAP	f, d	f[0:3]<>f[4:7]→d	None
000d1001 ffffffff	d9ff	A and f do AND operation.	AND	f, d	A&f→d	Z
000d1010 ffffffff	dAff	A and f do OR operation.	IOR	f, d	A|f→d	Z
000d1011 ffffffff	dBff	A and f do XOR operation	XOR	f, d	A^f→d	Z
000d1100 ffffffff	dCff	A plus f	ADD	f, d	A+f→d	Z, C
000d1101 ffffffff	dDff	F minus A	SUB	f, d	f-A→d	Z, C
000d1110 ffffffff	dEff	F-band c-cycle left shift	RCL	f, d	{f, C}<<1→d,f[7]→C	C
000d1111 ffffffff	dFff	F-band c cycle right shift	RCR	f, d	{C, f}>>1→d, f[0]→C	C
Constant class (16)						
00100000 kkkkkkkk	20kk	Subroutine with parameter return	RETL	k	k→A, TOS→PC	None
00100001 kkkkkkkk	21kk	Subroutine with parameters and returned in non-zero state.	RETLN	k	k→A,0→Z, TOS→PC	Z
0010001k kkkkkkkk	22kk	Constants are placed in indirect address register 1 and dedicated page bits.	MOVIP	k9	k9→{INDIR_RAM_PAGE, SFR_INDIR_ADDR}	INDIR_RAM _PAGE_BIT
001001kk kkkkkkkk	24kk	Constant is placed in indirect addressing register 2.	MOVIA	k10	k10→SFR_INDIR_ADDR2	None

00100011 kkkkkkkk	23kk	Constant is put into 1# fast register.	MOVA1F	k	k→1#ff	None
00100101 kkkkkkkk	25kk	Constant is put into 2# fast register.	MOVA2F	k	k→2#ff	None
00100110 kkkkkkkk	26kk	The constant is put into the register pointed by indirect address register 2.	MOVA2P	k	k→[@SFR_INDIR_ADDR2]	None
00100111 kkkkkkkk	27kk	The constant is put into the register pointed by indirect address register 1.	MOVA1P	k	k→[@SFR_INDIR_ADDR]	None
00101000 kkkkkkkk	28kk	Constant placement A	MOVL	k	k→A	None
00101001 kkkkkkkk	29kk	Constant and A do AND operation.	ANDL	k	k&A→A	Z
00101010 kkkkkkkk	2Akk	Constant and A do OR operation	IORL	k	k|A→A	Z
00101011 kkkkkkkk	2Bkk	XOR constant and A.	XORL	k	k^A→A	Z
00101100 kkkkkkkk	2Ckk	Constant plus A	ADDL	k	k+A→A	Z,C
00101101 kkkkkkkk	2Dkk	Constant minus A	SUBL	k	k-A→A	Z,C
00101110 kkkkkkkk	2Ekk	Constant plus A for comparison	CMPLN	k	k+A	Z,C
00101111 kkkkkkkk	2Fkk	Constant minus A for comparison.	CMPL	k	k-A	Z,C
Bit-oriented operation class (9)						
01000bbb fffffff	40ff	Clear bit b of f	BC	f, b	0→f[b]	None
01001bbb fffffff	48ff	Set bit b of f	BS	f, b	1→f[b]	None
01010bbb fffffff	50ff	If bit b of f is 0, it will jump.	BTSC	f, b	skip if f[b]==0	None
01011bbb fffffff	58ff	If bit b of f is 1, it will jump.	BTSS	f, b	skip if f[b]==1	None
00000000 000111aa	001C	Send the bits selected by A to C.	BCTC	a	bit[a]→C	C
00000000 100aabbb	008b	Transfer bits selected by bits b to a of the 1#	BP1F	a, b	1#bf[b]→bit[a]	C if a==0

		bit register.				
00000000 101aabbb	00Ab	Transfer bits selected by bits b to a of the 2# bit register.	BP2F	a, b	2#bf[b]→bit[a]	C if a==0
00000000 110aabbb	00Cb	Transfer the bit selected by a to bit b of the 1# bit register.	BG1F	a, b	bit[a]→1#bf[b]	None
00000000 111aabbb	00Eb	Transfer the bit selected by a to bit b of the 2# bit register.	BG2F	a, b	bit[a]→2#bf[b]	None
Transfer class (7)						
0110kkkk kkkkkkkk	6kkk	Jump	JMP	k12	k12→PC	None
0111kkkk kkkkkkkk	7kkk	Call subroutine	CALL	k12	PC+1→TOS,k12→PC	None
001100kk kkkkkkkk	3kkk	Jump when Z=0	JNZ	k10	k10→PC[9:0] if Z==0	None
001101kk kkkkkkkk	3kkk	Jump when Z=1	JZ	k10	k10→PC[9:0] if Z==1	None
001110kk kkkkkkkk	3kkk	Jump when C=0	JNC	k10	k10→PC[9:0] if C==0	None
001111kk kkkkkkkk	3kkk	Jump when C=1	JC	k10	k10→PC[9:0] if C==1	None
1KKKKKKK kkkkkkkk	8Kkk	A constant is compared with a, and if it is equal, it jumps.	CMPZ	K7, k	k→PC[7:0] if A==K7	None

3. Instruction Cycle

The instruction length of RISC8B is one word, but the instruction cycle is divided into four types: single cycle, double cycle, multi-cycle, and special multi-cycle. Among them, the number of cycles of multi-cycle depends on the process of the program ROM. Usually, multi-cycle is two cycles (The same double cycle), and it may exceed two cycles for individual chips with special processes (Currently only CH537X chips).

The instruction cycle is determined by the system working clock frequency SCLK, and the calculation formula is shown below:

$$Time\ of\ instruction\ cycle\ = \frac{1}{SCLK}$$

Special multi-cycle is suitable for individual special instructions, such as RDCODE, usually 3 cycles; WRCODE has a minimum of 4 cycles and a maximum execution time of several hundred milliseconds. EXEC is user-defined. In addition, all instructions and jump instructions to modify the program counter PC and SFR_PRG_COUNT are double-cycle or multi-cycle, and all others are single-cycle. The list of dual-cycle and multi-cycle instructions is as

follows:

Instruction mnemonic	Instruction Description	An additional condition for being double-cycle or multi-cycle
JMP, CALL	Direct jump	Always multi-cycle
RET, RETZ, RETIE, RETL, RETLN	Jump back	Always multi-cycle
JNZ, JZ, JNC, JC	Conditional jump	When the jump condition is true, it is multi-cycle, otherwise it is single-cycle.
CMPZ	Conditional jump	When the jump condition is true, it is multi-cycle, otherwise it is single-cycle.
BTSC, BTSS	Conditional jump	When the jump condition is true, it is multi-cycle, otherwise it is single-cycle.
Among all instructions oriented to byte operation classes, the instruction whose target register is SFR_PRG_COUNT, for example, ADD SFR_PRG_COUNT.	Modify program counter PC	Finally, the instruction to modify the SFR_PRG_COUNT register is multi-cycle, and the instruction not to modify or save is single-cycle. For example, "ADDSFR_PRG_COUNT, A" is a read operation, which is single-cycle.

4. Equivalence and Redefinition Instruction

4.1 Equivalent Instruction Mnemonic

The assembly tool WASM53B of RISC8B supports the following equivalent instruction mnemonics (Aliases):

Instruction description	Original instruction mnemonic	Equivalent instruction mnemonic
Clear watchdog timer	CLRWDT	WDT
Sleep	SLEEP	HALT
Wait for SPI to interrupt, write, or read.	WAITSPI	WAITWR
Save state to stack	PUSHAS	PUSH
Restore the state from the stack	POPAS	POP
Subroutine return	RET	RETURN
Subroutine returns to zero state (Successful return)	RETZ	RETOK
Interrupt return	RETIE	RETI
Subroutine with parameter return (Defining single byte data)	RETL	DB
Subroutine with parameters and return in non-zero state (Return with error code)	RETLN	RETER
Jump	JMP	GOTO
Instructions for byte operation classes, except CLRA, INCSZ, and DECSZ. Instructions for bit-oriented operation classes, except BTSC	Original mnemonics (e.g. INC, BC)	Add f after the original mnemonic (e.g. INCF, BCF)

and BTSS.		
F increases, and zero jumps.	INCSZ	INCFSZ
F decreases, and zero jumps.	DECSZ	DECFSZ
F-band c-cycle left shift	RCL	RCLF or RLF
F-band c cycle right shift	RCR	RCRF or RRF
If bit b of f is 0, it will jump.	BTSC	BTFSC
If bit b of f is 1, it will jump.	BTSS	BTFSS
Define double-byte data (Define new instructions, operands are new instruction codes)		DW

4.2 Redefine Instruction Code

In different chips or different applications, some instruction codes of RISC8B are redefined/reused. The following instruction codes represent different instructions:

Binary instruction code range	Basic instruction	Alternate instruction
00000000 00001100 → 00000000 000011kk	SLEEP	SLEEPX k2
00000000 00010000 → 00000000 00010bbb	WAITRD	WAITB b
00000000 00010100 → 00000000 00010bbb	WAITWR WAITSPI	WAITB b
00000000 00011000 → 00000000 000110kk	RDCODE	RCODE k2
00000000 00011100 → 00000000 000111kk	WRCODE	EXEC k2
00000000 00011100 → 00000000 000111aa	WRCODE	BCTC a
0010001k kkkkkkkk → 00100010 kkkkkkkk	MOVIP k9	MOVIP k8
0010001k kkkkkkkk → 00100011 kkkkkkkk	MOVIP k9	MOVA1F k
001001kk kkkkkkkk → 00100100 kkkkkkkk	MOVIA k10	MOVIA k8
001001kk kkkkkkkk → 00100101 kkkkkkkk	MOVIA k10	MOVA2F k
001001kk kkkkkkkk → 00100110 kkkkkkkk	MOVIA k10	MOVA2P k
001001kk kkkkkkkk → 00100111 kkkkkkkk	MOVIA k10	MOVA1P k

5. Addressing Mode

The addressing modes of RISC8B include: immediate number addressing, immediate number fast addressing, ordinary direct addressing, extended direct addressing, indirect addressing, and bit addressing, and the last four are used to address registers.

5.1 Immediate Number Addressing

Immediate addressing is used for constant instructions, and the operands in the instruction code are immediate

numbers, which are directly used for operation.

Example: MOVIA 0x246; SFR_INDIR_ADDR2=0x246, and write the data 0x246 into the address register of indirect addressing 2.

MOVIP 0x135; SB_INDIR_RAM_PAGE=1, SFR_INDIR_ADDR=0x35

ADDL 0x23; A=A+0x23

5.2 Extended Direct Addressing

The addressing range of extended direct addressing is 0x000-0x1FF, and the 9-bit register address is directly provided by the instruction code, which is only applicable to two instructions that directly read and write registers: MOV register, A or F instruction, and MOVA register instruction. Indirect addressing should be adopted when a wider range of addressing is needed.

Example: MOVA 0x167; [0x167]=A, write the data in A into the register with address 0x167, it is suggested to use the label instead.

MOV 0x289, A; A=[0x289], read the data in the register with address 0x289 into A, it is suggested to use a label instead.

5.3 Ordinary Direct Addressing

The addressing range of ordinary direct addressing is 0x000-0x0FF, and the 8-bit register address is directly provided by the instruction code, which applies to all direct address instructions except the above extended direct addressing instruction. Such as CLR register, ADD register, BS register, bit, etc. Indirect addressing should be adopted when a wider range of addressing is needed.

Example: CLR 0x67; [0x67]=0, clear the data in the register with address 0x67, and it is suggested to replace it with a label.

INC 0x89; [0x89]=[0x89]+1, add 1 to the data in the register with address 0x89, and it is suggested to use a label instead.

BS 0x9B, 3; [0x9B]=[0x9B]|0x08, set bit 3 of the register with address 0x9B to 1, and it is suggested to use a label instead.

5.4 Indirect Addressing

The addressing range of indirect addressing is 0x000-0x3FF, covering all registers of RISC8B. RISC8B provides two groups of indirect addressing registers, each of which consists of an address register and a data read-write port. Write the address of the destination register to the address register first, and then read and write the destination register through the read-write port.

The addressing range of indirect addressing is 0x000-0x3FF, covering all registers of RISC8B. RISC8B provides two groups of indirect addressing registers, each of which consists of an address register and a data read-write port. Write the address of the destination register to the address register first, and then read and write the destination register through the read-write port.

The indirectly addressed data read-write port SFR_INDIR_PORT is the destination register for reading and writing the address specified by the indirectly addressed address register SFR_INDIR_ADDR and its page bit SB_INDIR_RAM_PAGE, and the address range is 0x000-0x1FF, where SFR_INDIR_ADDR provides the lower 8-bit address. The indirectly addressed data read-write port SFR_INDIR_PORT2 is the destination register for reading and writing the address specified by the indirectly addressed address register SFR_INDIR_ADDR2. The

address range is 0x000-0x3FF, and all addresses are provided by SFR_INDIR_ADDR2.

Example: MOVL 0x45; A=0x45, which is immediate digital addressing.

MOVIA 0x357; SFR_INDIR_ADDR2=0x357, indirect addressing 0x357

MOVA SFR_INDIR_PORT2; [SFR_INDIR_ADDR2]=0x45. Write the data 0x45 into the register with address 0x357.

MOVIP 0x123; SB_INDIR_RAM_PAGE=1, SFR_INDIR_ADDR=0x23, indirect addressing 0x123.

ADD SFR_INDIR_PORT; [SB_INDIR_RAM_PAGE, SFR_INDIR_ADDR]+= 0x45

5.5 Immediate Fast Addressing

Immediate fast addressing is used for constant instructions, and the operands in the instruction code are immediate numbers, so they can be written directly to the target register without a transfer. MOVA1F and MOVA2F are suitable self-defined fast registers, which are equivalent to immediate digital addressing and ordinary direct addressing; MOVA1P and MOVA2P apply to all registers through indirect addressing, which is equivalent to immediate digital addressing and indirect addressing.

Example: MOVA1F 0x46; Write data 0x46 directly to the customized 1# fast register

MOVA1P 0x78; Write data 0x78 directly to the register pointed by SFR_INDIR_ADDR indirect addressing.

MOVA2P 0x92; Write data 0x92 directly to the register pointed to by SFR_INDIR_ADDR2 indirect addressing.

5.6 Bit Addressing

3-bit address is directly provided by instruction code for bit addressing, while ordinary direct addressing or indirect addressing can be used for addressing registers, respectively, to realize the addressing range of 0x000-0x0FF or 0x000-0x3FF, so that any bit of any register can be addressed.

The bit transfer instruction is used to copy the single-cycle bit data between the selected custom independent bit and the selected bit of the custom bit register.

Example: BC 0x67, 4; 0x67.4=0, set bit 4 of the register with address 0x67 to 0, and it is suggested to use a label instead.

MOVIA 0x135; SFR_INDIR_ADDR2=0x135, indirect addressing 0x135

BS SFR_INDIR_PORT2, 3; [SFR_INDIR_ADDR2].3=1, set bit 3 of the register with address 0x135 to 1.

BCTC 2; Copy the custom 2# independent bits into C.

BP2F 0, 5; Copy bit 5 of the self-defined 2# bit register to the 0# independent bit, namely C.

BG1F3,6; Copy the self-defined 3# independent bits to bit 6 of the self-defined 1# bit register.

6. Assembly Program

6.1 Assembly Tool

WASM53B.EXE is an assembly tool of a RISC8B single-chip microcomputer, which is used to compile the assembly source program into instruction code target data. WASM53B can run under DOS or Windows DOS, and the command line syntax is:

WASM53B Source file name

Under Windows, you can also drag the source file directly to the icon of the WASM53B tool to compile it. If the source file contains other files by using pseudo-instruction INCLUDE, you should indicate the full path including the drive letter for the included file.
The extension of the source file is ASM by default. If the extension of the source file is not ASM, you need to add a decimal point and extension. If there is no extension, you need to add a decimal point.
After running, WASM53B scans the source file twice. The first scan records all labels and constants. The second scan compiles the instruction mnemonics in the source file into corresponding instruction codes, generates list files and data files with the same names as the source file, and displays error messages and warning messages on the screen.
The extension of the list file is LST. The list file contains all source programs, corresponding instruction code target data, labels, constants, and various error and warning messages, and the number of all error and warning messages is displayed in the last line for checking errors and checking data.
The extension of the data file is BIN, and the data file contains the target data of the instruction code. Each instruction code occupies two bytes, with the low address byte being the low 8 bits of the instruction code and the high address byte being the high 8 bits of the instruction code. Because the program space of RISC8B can hold 4096 instructions, the length of the data file is always less than or equal to 8KB, and the unused program space will not be filled with any data by the assembler. The data in the data file can be directly placed in the program ROM of RISC8B.
Example: WASM53B SAMPLE.ASM
Generate: List file SAMPLE.LST and data file SAMPLE.BIN

6.2 Character Set

The label is a symbolic representation of the actual address of the program space. Usually, the label is automatically calculated by the assembler during the compilation process, rather than a predefined value. For example, the address to jump is labeled as the operand of the JMP instruction, and the starting address of the CALL subroutine is labeled as the operand of the call instruction.
Constants are symbolic representations of various numerical values. Usually, constants are values defined in advance by assembling pseudo-instruction EQU, including register address, register bit address, constant or immediate number, etc.
RISC8B assembler does not distinguish between labels and constants internally but handles them the same way, so the following descriptions no longer distinguish between labels and constants, and all descriptions about labels are also applicable to constants.
RISC8B's assembler is not case-sensitive. The valid characters of the label name include numbers (0-9), letters (A-Z, a-z), underscore (_), dollar sign ($), hashtag sign (#), and address sign (@), where the number (0-9) cannot be used as the first character of the label name, and the length of the label name cannot exceed 20.

The following words are reserved words, so they cannot be used as label names:
END, ORG, INCLUDE
The following words are reserved words and can be used as label names, but they are not easy to read and understand, so they are not recommended:
EQU, all instruction mnemonics, and corresponding equivalent instruction mnemonics.

6.3 Value

RISC8B's assembly tools support binary numbers, hexadecimal numbers, decimal numbers, and characters. Valid values range from -32768 to 65535.

Binary numbers can be expressed in two ways: prefix (0B), or prefix (B') and suffix ('), and the valid numeric characters are 0 and 1;

Hexadecimal numbers can be expressed in two ways: prefix (0X) or prefix (H') and suffix ('), and the valid numeric characters are 0-9 and A-f;

There are three representations of decimal numbers: direct representation, or prefix (0D), or prefix (D') and suffix ('), and valid numeric characters are 0-9;

There is only one representation of characters: prefix (') and suffix ('). Valid numeric characters are all ASCII codes.

Example: 0b01101001 and B'01101001' both represent 0x69 or 105.

0x5A, H'5A' both represent 0x5A or 90.

0d43, D'43', 43 both represent 0x2B or 43.

'a' represents 0x61 or 97, '6' represents 0x36 or 54.

6.4 Expression

RISC8B's assembler supports expressions with natural priority from right to left.

Operators include addition (+), subtraction (-), multiplication (*), division (/), remainder (%), bitwise AND (&), bitwise OR (|), bitwise XOR (), bitwise inversion (~), left shift (<<) and right shift (>>).

Bitwise inversion (~) only needs one operand on the right, and every other operator needs two operands, one on the left and the other on the right. The operands can be numeric values, previously defined labels, or nested expressions.

For the subtraction (-) operation, if there is no left operand, the left operand defaults to 0, which is equivalent to a negative sign.

Bitwise inversion (~), left shift (<<), and right shift (>>), the results of these three operations are automatically limited to the numerical range of 0 to 255, that is, the maximum 0xFF.

A nested expression means that there are multiple operators in the expression. In this case, the operator on the right has higher priority. The assembler first executes the right operator, operates its left operand and right operand, and the resulting result is used as the right operand of the further left operator, and then executes the further left operator to operate its left operand and the right result.

Example: The result of the expression {0x52 | H'0A'} is 0x52H|0x0A=0x5A.

The expression {2 * 7-4} results in 2*(7-4)=6.

The result of the expression {-0b11010110 &' c' 0xff} is 0x0000-(0xd6 & (0x43 0xff)) = 0xff6c.

NEXT_COUNT EQU THIS_COUNT + 1; Marker NEXT_COUNT = THIS_COUNT + 1.

ADDL 0xFF & 0 - LEN; A plus (0xFF & (0-LEN)) is equivalent to A minus LEN

6.5 Statement Format

RISC8B's statement of the assembly source program is in a line unit, and the format is as follows:

LABEL INSTRUCTION PARAMETER1, PARAMETER2 ; REMARK where:

LABEL is the label. The label must start from the first character of a line, a colon (:) can be added after the label, but the colon is only treated as a separator and has no effect. The assembler treats all words except reserved words starting from the first character of a line as labels. The label in front of the instruction mnemonic is optional, EQU

pseudo-instruction must have a label (constant), and other pseudo-instructions cannot have a label.

INSTRUCTION is an instruction mnemonic or pseudo instruction. To distinguish from labels, instruction mnemonics or EQU pseudo-instructions cannot start from the first character of a line, but other pseudo-instructions can start from any position in a line.

PARAMETER1 and PARAMETER2 are the first and second operands of an instruction or a pseudo-instruction. Operands can be numeric values, labels, or expressions. Depending on the instruction or pseudo-instruction, there may be only one operand or no operand. If there are two operands, they must be separated by commas or spaces.

A comment is a comment. Comments can be placed anywhere in a line with semicolons (;), the assembler ignores all the characters after the semicolon in a line as comments. Comments can be English characters or Chinese characters.

LABEL and INSTRUCTION, INSTRUCTION and PARAMETER1, and PARAMETER1 and PARAMETER2 must be separated by at least one space or other equivalent characters, including TAB, colon (:) and comma (,). A colon can only be used between LABEL and INSTRUCTION, and a comma can only be used between PARAMETER1 and PARAMETER2.

In the assembly source program of RISC8B, the length of each sentence can't exceed 249, that is, it can't exceed 249 characters. There can be blank lines in the source program, and each source program file can't exceed nine thousand nine hundred and ninety-nine lines. The source program with more than nine thousand nine hundred and ninety-nine lines can be divided into multiple inclusion files.

```
Example: PORTA       EQU     0x05            ; port A
         PA2         EQU     2
         ORG     0x80
         START:      CLR     0x0B            ; It is suggested to use a label instead of register address 0x0B.
                     MOVL    B'11111011'
                     MOVA    0x15            ; It is suggested to replace the register address 0x15 with a label.
         WAIT:       BTSC    PORTA, PA2      ;skip if PA2=0
                     JMP     WAIT
                     BC      3, 0            ;status register, C=0; It is suggested to replace register address 3 with
a label.
                     MOV     PORTA, A        ; Read data from port a to a.
                     PUSHAS                  ; Save A and status register to stack.
```

6.6 Pseudo Instruction

6.6.1 Label Assignment Pseudo Instruction EQU

EQU is used to assign values to labels. On the left of EQU is the assigned label name, and on the right of EQU is the numerical value, previously defined label or expression.

For the instructions of the byte-oriented operation class and bit-oriented operation class, if there are operands, usually the first operand may be the register address. It is suggested to use a label for this registered address to avoid directly using the immediate number as the registered address.

When compiling the source program, the assembler will automatically replace the corresponding label as the operand in the source program with the value of the label, and replace the numerical value with the label through EQU pseudo-instruction, which is convenient for program modification and easy to read and understand.

```
Example: MY_COUNT       EQU    0x23      ; Label MY_COUNT=0x23
```

6.6.2 Address Definition Pseudo Instruction ORG

ORG is used to define the starting address of subsequent instructions in the program space. On the right side of the ORG is the address, which can be a numerical value, a previously defined label, or an expression.
Without the ORG directive, the default starting address of the first directive in the whole source file is 0x0000. When the ORG directive defines the address forward, the assembler will provide a warning message of "go back by ORG". When the ORG directive defines the address backward, the skipped program space will be automatically filled with the empty operation instruction NOP by the assembler.
Example: ORG 0x0004; The next instruction starts at address 0x0004 in the program space.

6.6.3 The file Contains the Pseudo Instruction INCLUDE

INCLUDE is used to directly reference other assembler source files. To the right of INCLUDE is the referenced file name. If the referenced file is in the current directory, you only need to provide the file name and its extension, otherwise you need to provide the complete path. If compiled under Windows, the referenced file should provide the full path including the drive letter.
When the assembler processes the INCLUDE directive, it will open the referenced source file and compile it as a part of the current source file. After processing the referenced file, it will return to the next line of the current source file to continue compiling, which is equivalent to inserting the referenced file into the line where the INCLUDE directive is located in the current file. The referenced file can also refer to other files through the INCLUDE directive, and the assembly tool supports up to 8 levels of nesting of the INCLUDE directive. Through the INCLUDE pseudo-instruction, commonly used labels (constants) or subroutines can be used as general module files, which can be directly referenced by other assembly source programs without repeating the same definition or writing.
Example: INCLUDE C:\RISC8B\CH533INC.ASM; Reference the file C:\RISC8B\CH533INC.ASM here.

6.6.4 Source Program End Pseudo Instruction END

END is used to indicate the end of the source program.
When the assembler processes the END pseudo instruction, it will end the compilation process of the whole source program and will not continue to process the subsequent source programs. If the END directive is not processed, the assembler will always process the whole source file and provide a warning message of "END not found". There may be an END directive in the referenced file, but the assembler will only end the compilation process of the currently referenced file, not the whole source program.
Example: END; The compilation process ends here.

7. Instruction Specification

(Not yet)

8. Frequently Asked Questions

8.1. Manual checks after program modification: Single byte table lookup program.

Check whether the single-byte table lookup program is over-paged. The short jump page size of the RISC8B table lookup is 256. When using "ADDSFR_PRG_COUNT" and "RETL" instructions to look up tables, only the current address page of 256 bytes can be accessed, and the address of 0xXXFF must not be crossed.

WASM53B, an assembly tool, will analyze the program, and will usually give a warning when finding such anomalies.
It is suggested to use ORG to define the starting address of the table to avoid exceeding the limit.

8.2. Manual checks after program modification: double-byte table lookup program.

When using "RDCODE" and "DW" instructions to design a double-byte look-up table program, the address input method should be considered to avoid over-page.

8.3. Manual checks after program modification: long-expression or mathematical calculation.

Assembly tools are relatively simple, with long expressions or key mathematical calculations. It is recommended to manually check the object code in the LST file.

8.4. Manual checks after program modification: page-turning of the program.

Check the page-turning status in the program space. The length of the jump page of the RISC8B program is 4K. When switching between two program pages, the page to be jumped must be set manually in advance. If the subroutine is on another page, you need to manually set the page before calling, and the page will be automatically restored after calling back.
WASM53B, an assembly tool, will analyze the program, and will usually give a warning when finding such anomalies.

8.5. Subroutine libraries or programming examples

The following programming examples can be provided: 32-bit data addition and subtraction algorithm program, single-byte constant table lookup program, double-byte constant table lookup program, delay subroutine, serial port and SPI operation program, the program for reading and writing 24CXX series EEPROM and 25FXX series FLASH, basic framework program of USB products, and USB to serial port program compatible with CH341 chip.

8.6. If the program ROM is an OTP process, it is recommended to reserve empty operation instructions.

OTP process ROM can only be programmed once, and the ROM data bit is 0 when it is blank and may be set to 1 after programming. In the debugging process, to modify the local code again, you can temporarily choose to allow re-programming without read protection, and the ROM data bit with 0 can be re-programmed to 1, and the data bit 1 remains 1. In the program code, we suggest reserving the NOP null operation instruction in front of the key module, and its instruction code is 0. When the code needs to be modified, it can be replaced by other instructions, such as jump instructions, and then programmed again.