



CSKY ISA

USERGUIDE



Statement:

C-SKY Microsystems Co., Ltd. reserves all rights of this document.

Contents of this document can be altered, updated, deleted or changed, and no further notice will be given.

Copyright © 2001-2017 C-SKY Microsystems Co., Ltd.

Company address: 3 XiDoumen Rd,BldgA,15F,Hangzhou,China

Post Code: 310012

Tel: 0571-88157059

Fax: 0571-88157059-8888

Homepage: www.c-sky.com

E-mail: info@c-sky.com

Version history:

Version	Date	Description	Author
1.0	7/2/2010	The first version of user manual for CSKY instructions	C-SKY Microsystems Co., Ltd.
1.1	7/9/2010	1. Add 16-bit multiply instructions 2. Modify the descriptions about immediate operands of some instructions 3. Adjust the instruction code	C-SKY Microsystems Co., Ltd.
1.2	8/3/2010	1. Correct some coding and description errors 2. Modify the name of jmplr instruction 3. Modify the binary coding formats of ins, zext and sext operands	C-SKY Microsystems Co., Ltd.
1.3	10/13/2010	1. Modify the name of 16-bit instruction 2. Modify the codes of bmaski, bgenr, mvc, mvcc and rts instructions 3. Modify rotl instruction act 4. Add ff0 and lrw instructions 5. Modify the link register and define it as R15 6. Correct tstnbz instruction coding errors	C-SKY Microsystems Co., Ltd.
1.4	11/13/2010	1. Modify the binary coding formats of ins, zext and sext operands 2. Adjust the code of rts16 instruction 3. Correct coding errors and descriptions of lrw, ldm, stm, ldm16 and stm16 instructions 4. Modify the description about link register of jump instruction 5. Add idly instruction 6. Modify operand assembly formats and descriptions of all load/store instructions 7. Modify operand assembly format of trap instruction 8. Add descriptions about bit width of mnemonic instruction	C-SKY Microsystems Co., Ltd.
1.5	01/13/2011	1. Adjust the arrangement of instruction term list, add unified instructions, and give uniform descriptions to 16-bit and 32-bit instructions with the same function	C-SKY Microsystems Co., Ltd.

		<p>in instruction term list</p> <ol style="list-style-type: none"> 2. Modify the mnemonic symbols of 32-bit instructions and add the suffix of 32 3. Add not16, ldq16 and stq16 instructions 4. Modify descriptions and instruction formats of ldq and stq instructions 5. Modify tstnbz instruction 6. Modify mov instruction; the range of mov16 register is 32 7. Delete unaligned exception of jsri16 8. Modify andn32, asr32, lsl32, lsr32, subc32, subu32 and rtol32 instruction compiling results 9. Add bkpt16 instruction 10. Modify the code of mulsh32 instruction 	
1.6	02/11/2011	<ol style="list-style-type: none"> 1. Modify the definition of bmaski instruction function 	C-SKY Microsystems Co., Ltd.
1.7	03/29/2011	<ol style="list-style-type: none"> 1. Adjust RZ and RY/IMM5 coding positions of 32-bit instructions 2. Modify the codes of jmp32, jsr32 and rts32 instructions 	C-SKY Microsystems Co., Ltd.
1.8	04/14/2011	<ol style="list-style-type: none"> 1. Adjust the highest bit of 16-bit and 32-bit instructions 2. Adjust the codes of clrf, clrt, decf, dect, incf and inct as well as their pseudo instructions – movf and movt 3. Adjust the codes of pldr and pldw 4. Add idx instruction 	C-SKY Microsystems Co., Ltd.
1.9	05/17/2011	<ol style="list-style-type: none"> 1. Add ld32.d and st32.d instructions 2. Add sce instruction 	C-SKY Microsystems Co., Ltd.
1.10	05/24/2011	<ol style="list-style-type: none"> 1. Delete R15 register of supervisor mode 2. Adjust the codes of br32, bsr32, br16 and bsr16 	C-SKY Microsystems Co., Ltd.
1.11	05/25/2011	<ol style="list-style-type: none"> 1. Modify the operation of idly instruction 	C-SKY Microsystems Co., Ltd.

1.12	05/27/2011	1. Modify the operation of bsr32 and bsr16 instructions 2. Adjust the codes of bkpt32 and bkpt16 instructions	C-SKY Microsystems Co., Ltd.
1.13	11/15/2011	1. Add grs instruction 2. Add lrs.w, lrs.h, lrs.b, srs.w, srs.h and srs.b instructions 3. Correct the offset of ld.d and st.d 4. Modify abnormal descriptions of pldr and pldw instructions 5. Add descriptions about special operand of divs	C-SKY Microsystems Co., Ltd.
1.14	01/09/2012	1. Adjust the operands and codes of 16-bit instructions 2. Adjust the operands and codes of some 32-bit instructions 3. Add push and pop instructions 4. Add addi(sp) and subi(sp) instructions 5. Add jmp16, jsr16 and mvcv16 instructions 6. Add the definition about oimm18 operand of addi32 7. Delete jsri16, rotli16 and mvc16 instructions 8. Delete ldm16 and stm16 instructions 9. Delete be and bne instructions 10. Modify the range of lrw16 operand 11. Modify the code of grs instruction 12. Modify the definition of stack pointer register 13. Modify the jump range restriction of bsr instruction 14. Correct the description of mvtc instruction 15. Correct the description of cmpnei register type 16. Correct other errors	C-SKY Microsystems Co., Ltd.
1.14.1	02/06/2012	1. Add bmset32, bmclr32, cmpix32 and jmpix16 instructions	C-SKY Microsystems Co., Ltd.
1.14.2	02/13/2012	1. Add jmpix32 instruction 2. Modify the operands of push and pop instructions 3. Modify the offset displacement of ld.d and st.d instructions	C-SKY Microsystems Co., Ltd.
1.14.3	02/15/2012	1. Modify the codes of jmpi32 and jsri32	C-SKY Microsystems

			Co., Ltd.
1.14.4	03/30/2012	1. Correct the errors in coding mode of instruction set and description of operand type 2. Correct the abnormal description of load/store instructions 3. Correct mvtc code 4. Modify jsr16 code	C-SKY Microsystems Co., Ltd.
1.15	04/16/2012	1. Add floating point subset	C-SKY Microsystems Co., Ltd.
1.15.1	06/07/2012	1. Delete FMTD, FMTS, FMFD and FMFS	C-SKY Microsystems Co., Ltd.
1.15.2	06/28/2012	1. Add FMTVRH, FMTVRL, FMFVRH and FMFVRL 2. Add FLDD, FLDS, FLDRD, FLDRS, FLDMD, FLDMS, FSTD, FSTS, FSTRD, FSTRS, FSTMD and FSTMS	C-SKY Microsystems Co., Ltd.
1.15.3	07/01/2012	1. Modify the definitions of push and pop instructions 2. Add FLDM, FLDRM, FLDMM, FSTM, FSTRM and FSTMM	C-SKY Microsystems Co., Ltd.
1.15.4	08/01/2012	1. Correct the codes of some instructions	C-SKY Microsystems Co., Ltd.
1.15.5	08/17/2012	1. Add FMOVM	C-SKY Microsystems Co., Ltd.
1.15.6	08/17/2012	1. Modify the definition of co-processor instruction	C-SKY Microsystems Co., Ltd.
1.15.7	12/01/2012	1. Modify the definitions of sync and idly instructions	C-SKY Microsystems Co., Ltd.
1.15.8	10/14/2013	1. Add bpop.h, bpop.w, bpush.h and bpush.w	C-SKY Microsystems Co., Ltd.

1.15.9	31/3/2014	<ul style="list-style-type: none"> 1. Correct the names of FCMPZLSD and FCMPZLSS instructions 2. Correct the descriptions of FCMPZLSD, FCMPZLSS, FDTOSI, FDTOUI, FSITOD, FSITOS, FSTOSI, FSTOUI, FUITOD and FUITOS instructions 3. Correct the codes of VFPU instructions like FABSD, FABSM and FABSS 4. Delete cmpix instruction 5. Add strap and srte instructions 	C-SKY Microsystems Co., Ltd.
1.15.10	15/5/2014	<ul style="list-style-type: none"> 1. Delete bsr16 instruction 2. Add btsti16 instruction 3. Modify the code of lrw16 instruction 	C-SKY Microsystems Co., Ltd.
1.15.11	13/12/2014	<ul style="list-style-type: none"> 1. Correct the descriptions of FSTM and FLDM 2. Correct the description of FMULD instruction 3. Add FLRWS and FLRWD instructions 4. Add nie16, nir16, ipush16 and ipop16 instructions 5. Modify the name of floating point data operation instruction 	C-SKY Microsystems Co., Ltd.
1.15.12	22/12/2014	<ul style="list-style-type: none"> 1. Correct the descriptions of overflow bits of muls, mulsa, mulss, mulu, mulua and mulus instructions 2. Supplement the descriptions of mulsha, mulshs, mulsw, mulswa, mulsws, mfhis and mflos instructions 3. Supplement the descriptions of vmulsh, vmulsha, vmulshs, vmulsw, vmulswa and vmulsws instructions 	C-SKY Microsystems Co., Ltd.

Content

1. INTRODUCTION.....	11
1.1. INTRODUCTION	11
1.2. INSTRUCTION MIXING MODE	11
1.3. PROGRAMMING MODEL.....	12
1.3.1. <i>General-purpose register</i>	14
1.3.2. <i>Alternative register</i>	15
1.3.3. <i>Accumulator register</i>	15
1.3.4. <i>Program counter</i>	15
1.3.5. <i>Condition/carry bit</i>	16
2. 32-BIT INSTRUCTION SET.....	17
2.1. FUNCTIONAL CLASSIFICATION OF 32-BIT INSTRUCTIONS.....	17
2.1.1. <i>Data operation instruction</i>	17
2.1.2. <i>Branch jump instruction</i>	21
2.1.3. <i>Memory access instruction</i>	21
2.1.4. <i>Co-processor instruction</i>	23
2.1.5. <i>Privileged instruction</i>	23
2.1.6. <i>Special function instruction</i>	24
2.2. ENCODING OF 32-BIT INSTRUCTIONS	24
2.2.1. <i>Jump type</i>	25
2.2.2. <i>Immediate operand type</i>	25
2.2.3. <i>Register type</i>	26
2.3. OPERAND ADDRESSING MODE OF 32-BIT INSTRUCTIONS	26
2.3.1. <i>Addressing mode of jump-type instructions</i>	26
2.3.2. <i>Addressing mode of immediate operand-type instructions</i>	27
2.3.3. <i>Addressing mode of register-type instructions</i>	29
3. 16-BIT INSTRUCTION SET.....	32
3.1. MAPPING MODE OF 32-BIT/16-BIT INSTRUCTIONS	32
3.2. FUNCTIONAL CLASSIFICATION OF 16-BIT INSTRUCTIONS.....	37
3.2.1. <i>Data operation instruction</i>	37
3.2.2. <i>Branch jump instruction</i>	39
3.2.3. <i>Memory access instruction</i>	39
3.3. CODING MODE OF 16-BIT INSTRUCTIONS	40
3.3.1. <i>Jump type</i>	40

3.3.2. <i>Immediate operand type</i>	41
3.3.3. <i>Register type</i>	43
3.4. OPERAND ADDRESSING MODE OF 16-BIT INSTRUCTIONS	43
3.4.1. <i>Addressing mode of jump-type instructions</i>	43
3.4.2. <i>Addressing mode of immediate operand-type instructions</i>	44
3.4.3. <i>Addressing mode of register-type instructions</i>	46
4. FLOATING POINT INSTRUCTION SET	48
4.1. FUNCTIONAL CLASSIFICATION OF FLOATING POINT INSTRUCTIONS	48
4.1.1. <i>Data operation instruction</i>	48
4.1.2. <i>Vector operation instruction</i>	50
4.1.3. <i>Transfer instruction</i>	51
4.1.4. <i>Memory access instruction</i>	51
4.2. CODING MODE OF FLOATING POINT INSTRUCTIONS.....	52
4.2.1. <i>Register type</i>	52
4.3. OPERAND ADDRESSING MODE OF FLOATING POINT INSTRUCTIONS.....	52
4.3.1. <i>Addressing mode of register-type instructions</i>	53
5. TERM LIST OF BASIC INSTRUCTIONS.....	55
6. TERM LIST OF FLOATING POINT INSTRUCTIONS	433

List of charts:

CHART 1-1 MIXING PRINCIPLE OF 32-BIT/16-BIT INSTRUCTIONS	12
CHART 1-2 PROGRAMMING MODEL.....	12
CHART 1-3 DEFINITION OF GENERAL-PURPOSE REGISTER PROGRAMMING	14
CHART 2-1 LIST OF 32-BIT ADD-SUBTRACT INSTRUCTIONS.....	17
CHART 2-2 LIST OF 32-BIT LOGICAL OPERATION INSTRUCTIONS.....	18
CHART 2-3 LIST OF 32-BIT SHIFT INSTRUCTIONS.....	18
CHART 2-4 LIST OF 32-BIT COMPARE INSTRUCTIONS	18
CHART 2-5 LIST OF 32-BIT DATA TRANSFER INSTRUCTIONS	19
CHART 2-6 LIST OF 32-BIT BIT OPERATION INSTRUCTIONS.....	19
CHART 2-7 LIST OF 32-BIT EXTRACT AND INSERT INSTRUCTIONS	19
CHART 2-8 LIST OF 32-BIT MULTIPLY-DIVIDE INSTRUCTIONS.....	20
CHART 2-9 LIST OF 32-BIT MISCELLANEOUS OPERATION INSTRUCTIONS	20
CHART 2-10 LIST OF 32-BIT BRANCH INSTRUCTIONS	21
CHART 2-11 LIST OF 32-BIT JUMP INSTRUCTIONS.....	21
CHART 2-12 LIST OF 32-BIT IMMEDIATE OPERAND OFFSET ACCESS INSTRUCTIONS	21
CHART 2-13 LIST OF 32-BIT VECTOR REGISTER OFFSET ACCESS INSTRUCTIONS	22
CHART 2-14 LIST OF 32-BIT MULTI-REGISTER ACCESS INSTRUCTIONS	22
CHART 2-15 LIST OF 32-BIT EXCLUSIVE ACCESS INSTRUCTIONS.....	22
CHART 2-16 LIST OF 32-BIT SIGN ACCESS INSTRUCTIONS	23
CHART 2-17 LIST OF 32-BIT CO-PROCESSOR DATA TRANSFER INSTRUCTIONS	23
CHART 2-18 LIST OF 32-BIT CO-PROCESSOR MEMORY ACCESS INSTRUCTIONS	23
CHART 2-19 LIST OF 32-BIT CO-PROCESSOR OPERATION INSTRUCTIONS	23
CHART 2-20 LIST OF 32-BIT CONTROL REGISTER OPERATION INSTRUCTIONS	23
CHART 2-21 LIST OF 32-BIT LOW POWER CONSUMPTION INSTRUCTIONS	24
CHART 2-22 LIST OF 32-BIT ABNORMAL RETURN INSTRUCTIONS.....	24
CHART 2-23 LIST OF 32-BIT SAFE STATE INSTRUCTIONS	24
CHART 2-24 LIST OF 32-BIT SPECIAL FUNCTION INSTRUCTIONS	24
CHART 3-1 MAPPING TABLE OF 32-BIT/16-BIT INSTRUCTIONS	32
CHART 3-2 LIST OF 16-BIT ADD-SUBTRACT INSTRUCTIONS.....	37
CHART 3-3 LIST OF 16-BIT LOGICAL OPERATION INSTRUCTIONS.....	37
CHART 3-4 LIST OF 16-BIT SHIFT INSTRUCTIONS.....	38
CHART 3-5 LIST OF 16-BIT COMPARE INSTRUCTIONS	38
CHART 3-6 LIST OF 16-BIT DATA TRANSFER INSTRUCTIONS	38
CHART 3-7 LIST OF 16-BIT BIT OPERATION INSTRUCTIONS.....	38
CHART 3-8 LIST OF 16-BIT EXTRACT AND INSERT INSTRUCTIONS	39

CHART 3-9 LIST OF 16-BIT MULTIPLY-DIVIDE INSTRUCTIONS	39
CHART 3-10 LIST OF 16-BIT BRANCH INSTRUCTIONS	39
CHART 3-11 LIST OF 16-BIT JUMP INSTRUCTIONS.....	39
CHART 3-12 LIST OF 16-BIT IMMEDIATE OPERAND OFFSET ACCESS INSTRUCTIONS	40
CHART 3-13 LIST OF 16-BIT MULTI-REGISTER ACCESS INSTRUCTIONS	40
CHART 3-14 LIST OF 16-BIT BINARY TRANSLATED STACK INSTRUCTIONS.....	40
CHART 3-15 LIST OF 16-BIT INTERRUPT NESTING ACCELERATION INSTRUCTION	40
CHART 4-1 LIST OF SINGLE-PRECISION DATA OPERATION INSTRUCTIONS.....	48
CHART 4-2 LIST OF DOUBLE-PRECISION DATA OPERATION INSTRUCTIONS	49
CHART 4-3 LIST OF VECTOR OPERATION INSTRUCTIONS	50
CHART 4-4 LIST OF DATA TRANSFER INSTRUCTIONS	51
CHART 4-5 LIST OF DATA TRANSFER INSTRUCTIONS	51

1. Introduction

1.1. Introduction

CSKY instruction set architecture(ISA) refers to the second-generation independent intellectual property instruction set architecture of CK-Core family. CSKY ISA has characteristics like high performance, high code density, low power consumption and extensibility. CSKY ISA is designed by directing at different demands for embedded applications of high performance and low power consumption in the future. 32-bit/16-bit mixed length encoding is adopted. Among them, with perfect functions, 32-bit instruction is used to improve the comprehensive performance of instruction set; as the subset of 32-bit instruction, 16-bit instruction possesses relatively simple functions, and it is applied to improve instruction code density and to reduce power consumption.

Main characteristics of CSKY instruction set architecture are as follows:

- 32/16-bit instructions are realized by way of hybrid coding, and no performance loss will be caused in the process of instruction switch;
- As a complete set of instruction set architecture, 32-bit instructions have perfect functions and excellent performance;
- Most 16-bit instructions are subsets of 32-bit instructions and they can realize instructions with the highest frequency in 32-bit instructions;
- 32-bit instructions adopt 32 general-purpose registers and 3-operand addressing mode;
- 16-bit instructions adopt 16 general-purpose registers and 2-operand addressing mode.

1.2. Instruction mixing mode

CSKY distinguishes 32-bit instructions from 16-bit instructions through two highest bits in instruction codes. As for the two highest bits, 11 represents 32-bit instruction and the other one means 16-bit instruction. The specific instruction mixing mode is presented in Chart 1.1.



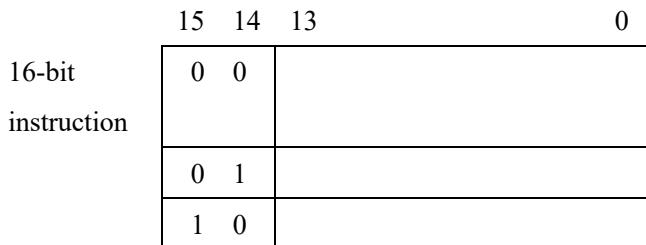


Chart 1-1 Mixing principle of 32-bit/16-bit instructions

1.3. Programming model

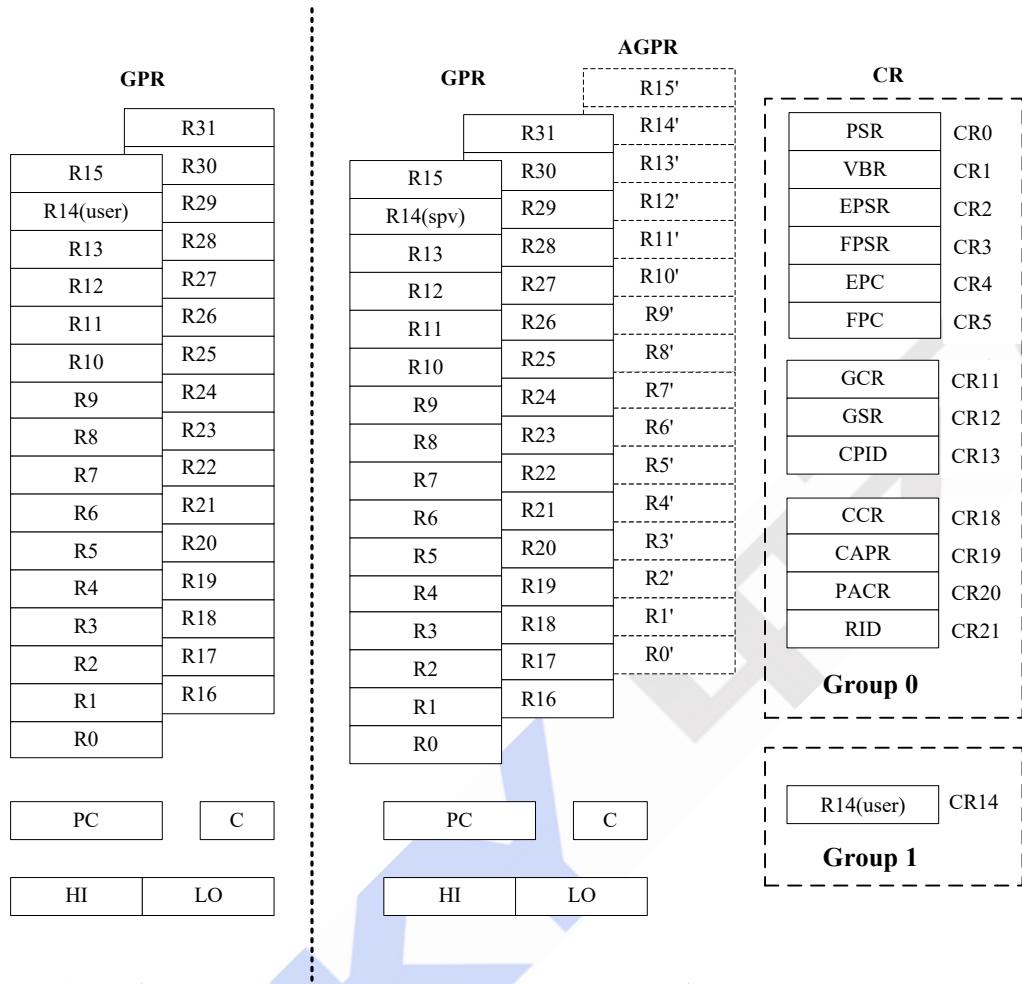


Chart 1-2 Programming model

CSKY defines two operation modes: user mode and supervisor mode. The above two operation modes are corresponding to different operation rights, and their differences are mainly reflected in two aspects: 1) access to the register; 2) use of privileged instructions.

Register accessing of user mode:

- 32 32-bit general-purpose registers (GPR)
- 32-bit program counter (PC)

- Condition/carry bit (C)
- Accumulator register (HI and LO)

Register access rights of supervisor mode cover:

- 32 32-bit general-purpose registers (GPR)
- 16 32-bit alternative registers (AGPR)
- 32-bit program counter (PC)
- Condition/carry bit (C)
- Accumulator register (HI and LO)
- Control register (CR)

The stack pointer (R14) of user mode and supervisor mode is independent, and only R14 (user) can be accessed under user mode. Under supervisor mode, R14 (spv) can be accessed; besides, R14 (user) of user can be indirectly visited by accessing CR14 in control register Group1.

In CK801, 16 alternative registers (AGPR) (R0'~R15') can be realized through hardware allocation. We can choose to access GPR or AGPR via AF bit in PSR. Meanwhile, AF bit can be set via mfcr and mtcr instructions. When AF bit in PSR is set, alternative general-purpose register block will be selected, and the processor can choose figures from alternative general-purpose register block for operation. When AF bit is cleared to zero, operation data will be extracted from general-purpose register block. Alternative general-purpose register block is used to reduce the time spent in context switch during real-time event processing.

Most instructions can be executed under two modes. However, a few instructions can be executed only under supervisor mode; otherwise, privilege exception will be triggered. These instructions include mfcr, mtcr, psrset, psrlcr, rte, rfi, wait, doze, stop, etc.

The operation right is decided by S bit of control register: 0 represents user mode and 1 means privileged user mode. The program will access register according to the rights. Common user program is only allowed to visit register under user mode, so as to prevent it from contacting privileged information of the system. System program under supervisor mode can access all registers and supervisor operation can be conducted with control register.

Under user mode, condition/carry bit (C) is located in the lowest bit of PSR, and it can be accessed and changed by common user instructions. It is the only data bit that can be visited under user mode in PSR. High accumulator register HI and low accumulator register LO are used to store multiplication and multiplication-accumulation results. Besides, they can be visited or altered through mfhi, mflo, mfhis, mflos, mthi and mtlo.

Besides registers that can be visited by user mode, supervisor mode also includes PSR register containing operation control and state information, a set of abnormal shadow registers (EPSR and EPC) used to save PSR and PC when exceptions take place, and a set of fast interrupt shadow registers (FPSR and FPC) used to save context switch time during fast interrupt. Besides, supervisor program can save the base address of interrupt vector table by utilizing VBR of a

register, covering a global status register (GSR), a global control register (GCR), and other relevant control registers.

1.3.1. General-purpose register

CSKY has realized 33 32-bit general-purpose registers which can be used to store instruction operands and instruction operation results. The stack pointer (R14) of user mode and supervisor mode is independent. Under user mode, the general-purpose register visits R14 (user); under supervisor mode, the general-purpose register visits R14 (spv). The specific rules of applying general-purpose register are as follows:

Chart 1-3 Definition of general-purpose register programming

Name	Function
R0	
R1	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14(user), SP(user)	Stack pointer
R14(spv), SP(user)	Stack pointer
R15	Link register
R16	
R17	
R18	
R19	
R20	
R21	
R22	
R23	

R24	
R25	
R26	
R27	
R28	Base address register of data segment
R29	
R30	Base address register of software vector
R31	

1.3.2. Alternative register

When switching to time-critical tasks, alternative register can be used to reduce response delay time caused by content conversion and site protection. When PSR (AF) is 1, alternative register will be selected and all instructions that use general-purpose register in the past will apply alternative register. On the contrary, if PSR (AF) is 0, alternative register will not be selected. Some important parameters and pointer values will be stored in these alternative registers. As long as alternative register is selected when task of high priority is executed, these important data can be used.

In addition, R14 serves the task as stack pointer in alternative register, thus the independent stack becomes more effective during realization.

In actual use, alternative register can also be accessed when AF is 1 under user mode. When exceptions appear and abnormal service program is executed, the low bit of entry value of abnormal service program vector will be copied to AF bit, so as to select the register block.

1.3.3. Accumulator register

Accumulator register includes two 32-bit registers (HI and LO). HI and LO can either be used separately to store 32-bit operation results or be combined into a 64-bit register [HI,LO] which can store multiplication and multiplication-accumulation results. Instructions of using and altering HI/LO include mulu, mulua, mulus, muls, mulsa and mulss; in addition, CSKY provides mfhi, mflo, mfhis, mflos, mthi and mtlo which can realize the communication function between common register and HI & LO registers.

1.3.4. Program counter

Program counter contains the current address of executing instructions. During normal operation or exception handling of program, the processor will automatically accumulate program counters or place a new value into the program counter according to the program

operation situations. For some special instructions, program counter can also participate in calculation as relative address. In addition, the low bit in program counter is 0 all the time unless unaligned access exception happens.

1.3.5. Condition/carry bit

Condition or carry bit represents the result after one operation. Condition/carry bit can be clearly set according to the results of compare instructions or unclearly set as some high-precision arithmetic or logical instructions. In addition, special instructions such as DEC[GT,LT,NE] and XTRB[0-3] will influence the value of condition/carry bit.

2. 32-bit instruction set

In this chapter, 32-bit instruction set of CSKY is introduced, covering functions, encoding and addressing mode, etc. of the 32-bit instruction set.

2.1. Functional classification of 32-bit instructions

According to functions of instruction realization, 32-bit instruction set of CSKY can be divided into:

- Data operation instruction
- Branch jump instruction
- Memory access instruction
- Co-processor instruction
- Privileged instruction
- Special function instruction

2.1.1. Data operation instruction

Data operation instruction can be further divided into:

Add-subtract instruction:

Chart 2-1 List of 32-bit add-subtract instructions

ADDU32	Add unsigned
ADDC32	Add with carry unsigned
ADDI32	Add immediate unsigned
SUBU32	Subtract unsigned
SUBC32	Subtract with borrow unsigned
SUBI32	Subtract immediate unsigned
RSUB32	Reverse subtract
IXH32	Index half-word
IXW32	Index word
IXD32	Index double word
INCF32	C=0 add immediate
INCT32	C=1 add immediate
DECF32	C=0 subtract immediate
DECT32	C=1 subtract immediate
DEC GT32	Set C bit when greater than zero in subtraction
DECLT32	Set C bit when smaller than zero in subtraction
DECNE32	Set C bit when not equal to zero in subtraction

Logical operation instruction:

Chart 2-2 List of 32-bit logical operation instructions

AND32	Bitwise AND
ANDI32	Bitwise AND immediate
ANDN32	Bitwise AND-NOT
ANDNI32	Bitwise AND-NOT immediate
OR32	Bitwise OR
ORI32	Bitwise OR immediate
XOR32	Bitwise XOR
XORI32	Bitwise XOR immediate
NOR32	Bitwise NOT-OR
NOT32	Bitwise NOT

Shift instruction:

Chart 2-3 List of 32-bit shift instructions

LSL32	Logical shift left
LSLI32	Logical shift left immediate
LSLC32	Logical shift left immediate to C
LSR32	Logical shift right
LSRI32	Logical shift right immediate
LSRC32	Logical shift right immediate to C
ASR32	Arithmetic shift right
ASRI32	Arithmetic shift right immediate
ASRC32	Arithmetic shift right immediate to C
ROTL32	Rotate left
ROTLI32	Rotate left immediate
XSR32	Extended shift right

Compare instruction:

Chart 2-4 List of 32-bit compare instructions

CMPNE32	Compare unequal
CMPNEI32	Compare unequal immediate
CMPHS32	Compare unsigned when greater or equal
CMPHSI32	Compare immediate unsigned when greater or equal
CMPLT32	Compare signed when smaller
CMPLTI32	Compare immediate signed when smaller
TST32	Null-test
TSTNBZ32	Register test without byte equal to zero

Data transfer instruction:

Chart 2-5 List of 32-bit data transfer instructions

MOV32	Move
MOVF32	C=0 Move
MOVT32	C=1 Move
MOVI32	Move immediate
MOVIH32	Move immediate high
MTHI32	Write transfer to high bit of accumulator
MTLO32	Write transfer to low bit of accumulator
MFHI32	Read transfer from high bit of accumulator
MFLO32	Read transfer from low bit of accumulator
MFHIS32	Read transfer saturate from high bit of accumulator
MFLOS32	Read transfer saturate from low bit of accumulator
MVCV32	C bit reverse move
MVC32	C bit move
MVTC32	Copy overflow bit to C bit
CLRF32	C=0 clear
CLRT32	C=1 clear
LRW32	Memory read-in
GRS32	Sign generation

Bit operation instruction:

Chart 2-6 List of 32-bit bit operation instructions

BCLRI32	Bit clear immediate
BSETI32	Bit set immediate
BTSTI32	Bit test immediate

Extract and insert instruction:

Chart 2-7 List of 32-bit extract and insert instructions

ZEXT32	Extract bit and extend unsigned
SEXT32	Extract bit and extend signed
INS32	Bit insert
ZEXTB32	Extract byte and extend unsigned
ZEXTH32	Extract half-word and extend unsigned
SEXTB32	Extract byte and extend signed
SEXTH32	Extract half-word and extend signed
XTRB0.32	Extract byte 0 and extend unsigned
XTRB1.32	Extract byte 1 and extend unsigned
XTRB2.32	Extract byte 2 and extend unsigned

XTRB3.32	Extract byte 3 and extend unsigned
BREV32	Bit-reverse
REVB32	Byte-reverse
RE VH32	Half-word byte-reverse

Multiply-divide instruction:

Chart 2-8 List of 32-bit multiply-divide instructions

MULU32	Multiply unsigned
MULUA32	Multiply-accumulate unsigned
MULUS32	Multiply-subtract unsigned
MULS32	Multiply signed
MULSA32	Multiply-accumulate signed
MULSS32	Multiply-subtract signed
MULSH32	16-bit multiply signed
MULSHA32	16-bit multiply-accumulate signed
MULSHS32	16-bit multiply-subtract signed
MULSW32	16x32 multiply signed
MULSWA32	16x32 multiply-accumulate signed
MULSWS32	16x32 multiply-subtract signed
VMULSH32	16-bit multiply signed in two branches
VMULSHA32	16-bit multiply-accumulate signed in two branches
VMULSHS32	16-bit multiply-subtract signed in two branches
VMULSW32	16x32 multiply signed in two branches
VMULSWA32	16x32 multiply-accumulate signed in two branches
VMULSWS32	16x32 multiply-subtract signed in two branches
MULT32	Multiply
DIVU32	Divide unsigned
DIVS32	Divide signed

Miscellaneous operation instruction:

Chart 2-9 List of 32-bit miscellaneous operation instructions

ABS32	Absolute value
FF0.32	Fast find 0
FF1.32	Fast find 1
BMASKI32	Bit mask generation immediate
BGENR32	Register bit generation

BGENI32	Bit generation immediate
---------	--------------------------

2.1.2. Branch jump instruction

Branch jump instruction can be further divided into:

Branch instruction:

Chart 2-10 List of 32-bit branch instructions

BT32	C=1 branch instruction
BF32	C=0 branch instruction
BEZ32	Branch instruction when register is equal to zero
BNEZ32	Branch instruction when register is not equal to zero
BHZ32	Branch instruction when register is greater than zero
BLSZ32	Branch instruction when register is smaller than or equal to zero
BLZ32	Branch instruction when register is smaller than zero
BHSZ32	Branch instruction when register is greater than or equal to zero

Jump instruction:

Chart 2-11 List of 32-bit jump instructions

BR32	Unconditional jump
BSR32	Jump to subprogram
JMPI32	Jump indirect
JSRI32	Jump to subprogram indirect
JMP32	Register jump
JSR32	Register jump to subprogram
RTS32	Link register jump
JMPIX32	Register index jump

2.1.3. Memory access instruction

Memory access instruction can be further divided into:

Immediate operand offset access instruction:

Chart 2-12 List of 32-bit immediate operand offset access instructions

LD32.B	Load unsigned and extended byte
LD32.BS	Load signed and extended byte
LD32.H	Load unsigned and extended half-word
LD32.HS	Load signed and extended half-word

LD32.W	Load word
LD32.D	Load double word
ST32.B	Store byte
ST32.H	Store half-word
ST32.W	Store word
ST32.D	Store double word

Vector register offset access instruction:

Chart 2-13 List of 32-bit vector register offset access instructions

LDR32.B	Load unsigned and extended byte in register offset addressing
LDR32.BS	Load signed and extended byte in register offset addressing
LDR32.H	Load unsigned and extended half-word in register offset addressing
LDR32.HS	Load signed and extended half-word in register offset addressing
LDR32.W	Load word in register offset addressing
STR32.B	Store byte in register offset addressing
STR32.H	Store half-word in register offset addressing
STR32.W	Store word in register offset addressing

Multi-register access instruction:

Chart 2-14 List of 32-bit multi-register access instructions

LDQ32	Load consecutive quad word
LDM32	Load consecutive multiword
STQ32	Store consecutive quad word
STM32	Store consecutive multiword
PUSH32	Push
POP32	Pop

Exclusive access instruction:

Chart 2-15 List of 32-bit exclusive access instructions

LDEX32.W	Load word exclusive
STEX32.W	Store word exclusive

Sign access instruction:

Chart 2-16 List of 32-bit sign access instructions

LRS32.B	Load byte sign
LRS32.H	Load half-word sign
LRS32.W	Load word sign
SRS32.B	Store byte sign
SRS32.H	Store half-word sign
SRS32.W	Store word sign

2.1.4. Co-processor instruction

Co-processor instruction can be further divided into:

Co-processor data transfer instruction:

Chart 2-17 List of 32-bit co-processor data transfer instructions

CPRGR32	Read transfer from general-purpose register of co-processor
CPWGR32	Write transfer to general-purpose register of co-processor
CPRCR32	Read transfer from control register of co-processor
CPWCR32	Write transfer to control register of co-processor
CPRC32	Read transfer from condition bit of co-processor

Co-processor memory access instruction:

Chart 2-18 List of 32-bit co-processor memory access instructions

LDCPR32	Load word to co-processor
STCPR32	Store word in co-processor

Co-processor operation instruction:

Chart 2-19 List of 32-bit co-processor operation instructions

CPOP32	Co-processor operation instruction
--------	------------------------------------

2.1.5. Privileged instruction

Privileged instruction can be further divided into:

Control register operation instruction:

Chart 2-20 List of 32-bit control register operation instructions

MFCR32	Read from control register
MTCR32	Write to control register

PSRSET32	Set PSR bit
PSRCLR32	Clear PSR bit

Low power consumption instruction:

Chart 2-21 List of 32-bit low power consumption instructions

WAIT32	Enter low power consumption wait mode
DOZE32	Enter low power consumption doze mode
STOP32	Enter low power consumption stop mode

Abnormal return instruction:

Chart 2-22 List of 32-bit abnormal return instructions

RTE32	Return from abnormal and normal interrupt
RFI32	Return from fast interrupt

Safe state instruction:

Chart 2-23 List of 32-bit safe state instructions

STRAP32	Enter safe state
SRTE32	Return from safe state

2.1.6. Special function instruction

Specifically speaking, special function instruction includes:

Chart 2-24 List of 32-bit special function instructions

SYNC32	Synchronize CPU
BKPT32	Breakpoint instruction
SCE32	Set conditional execution
IDLY32	Ban interrupt identification
TRAP32	Unconditional operating system trap
PLDR32	Prefetch read data
PLDW32	Prefetch write data
WE32	Wait event
SE32	Send event
BMSET32	Set BCTM bit
BMCLR32	Clear BCTM bit

2.2. Encoding of 32-bit instructions

The 32-bit instruction set of CSKY can be divided into 3 categories in coding style:

- Jump type (J type)

- Immediate operand type (I type)
- Register type (R type)

2.2.1. Jump type

The coding mode of jump type (J type) of 32-bit instructions is shown in the following chart:

31	30 29	26 25	0
1	1	OP	Offset/User Define
2	4		26

OP field is the main operation code and instructions of this coding type can be identified through 4-bit operation code; Offset/User Define field is the offset of jump instruction or user defined reserved domain.

2.2.2. Immediate operand type

Immediate operand type (I type) of 32-bit instructions covers three coding modes including 18-bit immediate operand, 16-bit immediate operand and 12-bit immediate operand.

The coding mode of 18-bit immediate operand is shown in the following chart:

31	30 29	26 25	21 20	18 17	0
1	1	OP	RZ	SOP	IMM18
2	4		5	3	18

OP field is the main operation code and the instruction or instruction type can be identified through 4-bit main operation code; RZ field is the destination register field; SOP field is the sub-operation code field; IMM18 field is the 18-bit immediate operand. The instruction type can be gained after decoding the main operation code OP, and the specific instruction can be obtained only after further decoding the sub-operation code SOP.

The coding mode of 16-bit immediate operand is shown in the following chart:

31	30 29	26 25	21 20	16 15	0
1	1	OP	RZ/SOP	RX	IMM16
2	4		5		16

OP field is the main operation code and the instruction or instruction type can be identified through 4-bit main operation code; RZ/SOP field is the destination register field or sub-operation code field; RX field is the first source register; IMM16 field is the 16-bit immediate operand.

The coding mode of 12-bit immediate operand is shown in the following chart:

31	30 29	26 25	21 20	16 15	12 11	0
----	-------	-------	-------	-------	-------	---

1 1	OP	RZ/RY	RX	SOP	IMM12
2	4	5	5	4	12

OP field is the main operation code and the instruction type can be identified through 4-bit main operation code; RZ/RY field is the destination register field or second source register field; RX field is the first source register; SOP field is the sub-operation code field; IMM12 field is the 12-bit immediate operand. The instruction type can be gained after decoding the main operation code OP, and the specific instruction can be obtained only after further decoding the sub-operation code SOP.

2.2.3. Register type

The coding mode of register type (R type) of 32-bit instructions is shown in the following chart:

31 30 29	26 25	21 20	16 15	10 9	5 4	0
1 1	OP	RY/IMM5	RX	SOP	Pcode	RZ

2 4 5 5 6 5 5

OP field is the main operation code and the instruction type can be identified through 4-bit main operation code; RY/IMM5 field is the second source register field or 5-bit immediate operand; RX is the first source register; SOP field is the sub-operation code field; Pcode is the parallel operation code field; RZ field is the destination register field. As for some instructions, the instruction type is gained after decoding the main operation code OP, the subclass of instruction is obtained by decoding the sub-operation code SOP, and then the specific instruction is identified by decoding the parallel operation code Pcode. Pcode adopts one-hot coding mode.

2.3. Operand addressing mode of 32-bit instructions

The 32-bit instruction set of CSKY follows three instruction coding modes and each coding mode has its own operand addressing mode. In the following, various operand addressing modes will be introduced.

2.3.1. Addressing mode of jump-type instructions

The 32-bit instructions of jump type in CSKY only have one addressing mode.

2.3.1.1. Addressing mode of 26-bit immediate operand

In the instructions that adopt the addressing mode of 26-bit immediate operand, there is an immediate operand field with the length of 26 bits. This field is considered as offset which can be used to generate destination address. Instruction of this format includes bsr32.

31 30 29	26 25	0
----------	-------	---

1 1	OP	Offset
2	4	26

2.3.2. Addressing mode of immediate operand-type instructions

The 32-bit instructions of immediate operand type in CSKY have six addressing modes.

2.3.2.1. Addressing mode of single register 18-bit immediate operand

In the instructions that adopt the addressing mode of single register 18-bit immediate operand, RZ field is the destination register field or second source register field; SOP field is the sub-operation code field; IMM18 is the 18-bit relative offset used to generate destination address. Instructions of this format include lrs32.b, lrs32.h, lrs32.w, grs32, srs32.b, srs32.h, srs32.w and addi32.

31 3029	26 25	21 20	18 17	0
1 1	OP	RZ	SOP	IMM18

2 4 5 3 18

2.3.2.2. Addressing mode of two register 16-bit immediate operand

In the instructions that adopt the addressing mode of two register 16-bit immediate operand, the two register fields RX and RZ are source register field and destination register field; IMM16 field directly participates in data operation as 16-bit immediate operand. Instruction of this format includes ori32.

31 3029	26 25	21 20	16 15	0
1 1	OP	RZ	RX	IMM16

2 4 5 5 16

2.3.2.3. Addressing mode of single register 16-bit immediate operand

The instructions that adopt the addressing mode of single register 16-bit immediate operand have two formats.

In the first format, SOP field is the sub-operation code field; RX field is the source register field; IMM16 field can generate destination address as 16-bit relative offset; IMM16 field can also directly participate in data operation as 16-bit immediate operand. Instructions of this format include bez32, bnez32, bhz32, blsz32, blz32, bhsz32, cmphsi32, cmplti32 and cmpnei32.

31 3029	26 25	21 20	16 15	0
1 1	OP	SOP	RX	IMM16

2 4 5 5 16

In the second format, SOP field is the sub-operation code field; RZ field is the destination register field; IMM16 field can either directly participate in data operation as 16-bit immediate operand or be left to user for definition. Instructions of this format include movi32, movih32 and lrw32.

31	3029	26 25	21 20	16 15	0
1	1	OP	SOP	RZ	IMM16
2	4	5	5	16	

2.3.2.4. Addressing mode of 16-bit immediate operand

In the instructions that adopt the addressing mode of 16-bit immediate operand, there is an immediate operand field with the length of 16 bits. This field is considered as offset which can be used to generate destination address. Instructions of this format include br32, bf32, bt32, jmpi32 and jsri32.

31	3029	26 25	21 20	16 15	0
1	1	OP	SOP	0 0 0 0 0	IMM16
2	4	5	5	16	

2.3.2.5. Addressing mode of two register 12-bit immediate operand

In the instructions that adopt the addressing mode of two register 12-bit immediate operand, RZ field is the destination register field or second source register field; RX field is the first source register field; SOP field is the sub-operation code field; IMM12 field can be used to generate destination address as 12-bit relative offset. Instructions of this format include ld32.b, ld32.h, ld32.w, ld32.d, ld32.bs, ld32.hs, ldex32.w, pldr32, st32.b, st32.h, st32.w, st32.d, stex32.w, pldw, addi32, subi32, andi, andni and xorl.

31	3029	26 25	21 20	16 15	12 11	0
1	1	OP	RZ	RX	SOP	IMM12
2	4	5	5	4	12	

2.3.2.6. Addressing mode of single register 12-bit immediate operand

In the instructions that adopt the addressing mode of single register 12-bit immediate operand, RX field is the first source register field; CPRZ is the co-processor operation field; SOP field is the sub-operation code field; IMM12 field is left to user for definition. Instructions of this format include cprgr32, cpwgr32, cprcr32, cpwcr32, cprc32, ldcpr32, stcpr32 and cpop32.

31	3029	26 25	21 20	16 15	12 11	0
----	------	-------	-------	-------	-------	---

1 1	OP	CPRZ	RX	SOP	IMM12	
2	4	5	5	4	12	

2.3.3. Addressing mode of register-type instructions

The 32-bit instructions of register type in CSKY have five addressing modes.

2.3.3.1. Addressing mode of ternary register

In addressing mode of ternary register, RY is the second source register field; RX field is the first source register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field; RZ is the destination register field. Instructions of this format include addu32, addc32, subu32, subc32, ixh32, ixw32, ixd32, and32, andn32, or32, xor32, nor32, lsl32, lsr32, asr32, rotl32, ldr32.b, ldr32.h, ldr32.w, ldr32.bs, ldr32.hs, str32.b, str32.h, str32.w, divu32, divs32, mult32, mulsh32 and mulsw32.

31 3029	26 25	21 20	16 15	10 9	5 4	0
1 1	OP	RY	RX	SOP	Pcode	RZ

2.3.3.2. Addressing mode of two register 5-bit immediate operand

Addressing mode of two register 5-bit immediate operand can be further divided into two formats.

In the first format, IMM5 field is the 5-bit immediate operand and treated as source operand; RX field is the source register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field; RZ field is the destination register field. Instructions of this format include decgt32, declt32, decne32, lsli32, lsri32, asri32, rotli32, lscl32, lsrl32, asrc32, xsr32, bclri32 and bseti32.

31 3029	26 25	21 20	16 15	10 9	5 4	0
1 1	OP	IMM5	RX	SOP	Pcode	RZ

In the second format, IMM5 field is the 5-bit immediate operand and treated as source operand; RX field is the source register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field; RZ field is the destination register field or second source register field. Instructions of this format include incf32, inct32, decf32 and dect32.

31 3029	26 25	21 20	16 15	10 9	5 4	0
1 1	OP	RZ	RX	SOP	Pcode	IMM5

2	4	5	5	6	5	5
---	---	---	---	---	---	---

2.3.3.3. Addressing mode of two register

Instructions that adopt the addressing mode of two register can be further divided into two formats.

In the first format, RZ field is the destination register field; RX field is the source register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field. Instructions of this format include bgenr32, xtrb0.32, xtrb1.32, xtrb2.32, xtrb3.32, brev32, revb32, revh32, abs32, ff0.32 and ff1.32.

31	3029	26	25	21	20	16	15	10	9	5	4	0
1	1	OP	0	RX	SOP	Pcode	RZ					
2	4		5	5		6		5		5		5

In the second format, RY field is the second source register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field; RX field is the first source register field. Instructions of this format include cmpne32, cmphs32, cmplt32, tst32, mulu32, mulua32, mulus32, muls32, mulsa32, mulss32, mulsha32, mulshs32, mulswa32, mulsws32, vmulsh32, vmulsha32, vmulshs32, vmulsw32, vmulswa32 and vmulsws32.

31	3029	26	25	21	20	16	15	10	9	5	4	0
1	1	OP	RY	RX	SOP	Pcode	0					
2	4		5	5		6		5		5		5

2.3.3.4. Addressing mode of single register 5-bit immediate operand

Instructions that adopt the addressing mode of single register 5-bit immediate operand can be further divided into two formats.

In the first format, IMM5 field is the 5-bit immediate operand and treated as source operand; RX field is the source register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field. Instruction of this format includes btsti32.

31	3029	26	25	21	20	16	15	10	9	5	4	0
1	1	OP	IMM5	RX	SOP	Pcode	0					
2	4		5	5		6		5		5		5

In the second format, IMM5 field is the 5-bit immediate operand and treated as source operand; SOP field is the sub-operation code field; Pcode is the parallel operation code field; RZ field is the destination register field. Instruction of this format includes bmaski32.

31	3029	26	25	21	20	16	15	10	9	5	4	0

1 1	OP	IMM5	0	SOP	Pcode	RZ
2	4	5	5	6	5	5

2.3.3.5. Addressing mode of single register

Addressing mode of single register can be further divided into three formats.

In the first format, RZ is the destination register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field. Instructions of this format include mvc32 and mvcv32.

31 3029	26 25	21 20	16 15	10 9	5 4	0
1 1	OP	0	0	SOP	Pcode	RZ

1 1	OP	0	0	SOP	Pcode	RZ
2	4	5	5	6	5	5

In the second format, RX is the source register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field. Instruction of this format includes tstnbz32.

31 3029	26 25	21 20	16 15	10 9	5 4	0
1 1	OP	0	RX	SOP	Pcode	0

1 1	OP	0	RX	SOP	Pcode	0
2	4	5	5	6	5	5

In the third format, RZ is the destination register field and source register field; SOP field is the sub-operation code field; Pcode is the parallel operation code field. Instructions of this format include clrf32 and clrt32.

31 3029	26 25	21 20	16 15	10 9	5 4	0
1 1	OP	RZ	0	SOP	Pcode	0

1 1	OP	RZ	0	SOP	Pcode	0
2	4	5	5	6	5	5

3. 16-bit instruction set

In this chapter, 16-bit instruction set of CSKY is mainly introduced, covering the mapping mode from 32-bit instruction set to 16-bit instruction set as well as functional classification, encoding and addressing mode of 16-bit instruction set.

3.1. Mapping mode of 32-bit/16-bit instructions

16-bit instruction set in CSKY is a subset of 32-bit instruction set, and most 16-bit instructions have corresponding 32-bit instructions. The specific mapping mode between 32-bit instructions and 16-bit instructions is presented in the following table:

Chart 3-1 Mapping table of 32-bit/16-bit instructions

Assembly instruction	32-bit	16-bit	Instruction description
ADDU	○	○	Add unsigned
ADDC	○	○	Add with carry unsigned
ADDI	○	○	Add immediate unsigned
SUBU	○	○	Subtract unsigned
SUBC	○	○	Subtract with borrow unsigned
SUBI	○	○	Subtract immediate unsigned
RSUB	○	×	Reverse subtract
IXH	○	×	Index half-word
IXW	○	×	Index word
IXD	○	×	Index double word
INCF	○	×	C=0 add immediate
INCT	○	×	C=1 add immediate
DECF	○	×	C=0 subtract immediate
DECT	○	×	C=1 subtract immediate
DEC GT	○	×	Set C bit when greater than zero in subtraction
DECLT	○	×	Set C bit when smaller than zero in subtraction
DECNE	○	×	Set C bit when not equal to zero in subtraction
AND	○	○	Bitwise AND
ANDI	○	×	Bitwise AND immediate
ANDN	○	○	Bitwise AND-NOT
ANDNI	○	×	Bitwise AND-NOT immediate
OR	○	○	Bitwise OR
ORI	○	×	Bitwise OR immediate
XOR	○	○	Bitwise XOR
XORI	○	×	Bitwise XOR immediate
NOR	○	○	Bitwise NOT-OR
NOT	○	○	Bitwise NOT

LSL	○	○	Logical shift left
LSLI	○	○	Logical shift left immediate
LSLC	○	×	Logical shift left immediate to C
LSR	○	○	Logical shift right
LSRI	○	○	Logical shift right immediate
LSRC	○	×	Logical shift right immediate to C
ASR	○	○	Arithmetic shift right
ASRI	○	○	Arithmetic shift right immediate
ASRC	○	×	Arithmetic shift right immediate to C
ROTL	○	○	Rotate left
ROTLI	○	×	Rotate left immediate
XSR	○	×	Extended shift right
CMPNE	○	○	Compare unequal
CMPNEI	○	○	Compare unequal immediate
CMPHS	○	○	Compare unsigned when greater or equal
CMPHSI	○	○	Compare immediate unsigned when greater or equal
CMPLT	○	○	Compare signed when smaller
CMPLTI	○	○	Compare immediate signed when smaller
TST	○	○	Null-test
TSTNBZ	○	○	Register test without byte equal to zero
MOV	○	○	Move
MOVF	○	×	C=0 move
MOVT	○	×	C=1 move
MOVI	○	○	Move immediate
MOVIH	○	×	Move immediate high
LRW	○	○	Memory read-in
GRS	○	×	Sign generation
MTHI	○	×	Write transfer to high bit of accumulator
MTLO	○	×	Write transfer to low bit of accumulator
MFHI	○	×	Read transfer from high bit of accumulator
MFLO	○	×	Read transfer from low bit of accumulator
MFHIS	○	×	Read transfer saturate from high bit of accumulator
MFLOS	○	×	Read transfer saturate from low bit of accumulator
MVCV	○	○	C bit reverse move
MVC	○	×	C bit move
MVTC	○	×	Copy overflow bit to C bit
CLRF	○	×	C=0 clear
CLRT	○	×	C=1 clear
BCLRI	○	○	Bit clear immediate
BSETI	○	○	Bit set immediate

BTSTI	○	○	Bit test immediate
ZEXT	○	×	Extract bit and extend unsigned
SEXT	○	×	Extract bit and extend signed
INS	○	×	Bit insert
ZEXTB	○	○	Extract byte and extend unsigned
ZEXTH	○	○	Extract half-word and extend unsigned
SEXTB	○	○	Extract byte and extend signed
SEXTH	○	○	Extract half-word and extend signed
XTRB0	○	×	Extract byte 0 and extend unsigned
XTRB1	○	×	Extract byte 1 and extend unsigned
XTRB2	○	×	Extract byte 2 and extend unsigned
XTRB3	○	×	Extract byte 3 and extend unsigned
BREV	○	×	Bit-reverse
REVB	○	○	Byte-reverse
REVH	○	○	Half-word byte-reverse
MULU	○	×	Multiply unsigned
MULUA	○	×	Multiply-accumulate unsigned
MULUS	○	×	Multiply-subtract unsigned
MULS	○	×	Multiply signed
MULSA	○	×	Multiply-accumulate signed
MULSS	○	×	Multiply-subtract signed
MULSH	○	○	16-bit multiply signed
MULSHA	○	×	16-bit multiply-accumulate signed
MULSHS	○	×	16-bit multiply-subtract signed
MULSW	○	×	16x32 multiply signed
MULSWA	○	×	16x32 multiply-accumulate signed
MULSWS	○	×	16x32 multiply-subtract signed
VMULSH	○	×	16-bit multiply signed in two branches
VMULSH	○	×	16-bit multiply-accumulate signed in two branches
VMULSHS	○	×	16-bit multiply-subtract signed in two branches
VMULSW	○	×	16x32 multiply signed in two branches
VMULSW	○	×	16x32 multiply-accumulate signed in two branches
VMULSW	○	×	16x32 multiply-subtract signed in two branches
MULT	○	○	Multiply
DIVU	○	×	Divide unsigned
DIVS	○	×	Divide signed
ABS	○	×	Absolute value
FF0	○	×	Fast find 0

FF1	○	×	Fast find 1
BMASKI	○	×	Bit mask generation immediate
BGENR	○	×	Register bit generation
BGENI	○	×	Bit generation immediate
BT	○	○	C=1 branch instruction
BF	○	○	C=0 branch instruction
BEZ	○	×	Branch instruction when register is equal to zero
BNEZ	○	×	Branch instruction when register is not equal to zero
BHZ	○	×	Branch instruction when register is greater than zero
BLSZ	○	×	Branch instruction when register is smaller than or equal to zero
BLZ	○	×	Branch instruction when register is smaller than zero
BHSZ	○	×	Branch instruction when register is greater than or equal to zero
BR	○	○	Unconditional jump
BSR	○	×	Jump to subprogram
JMPI	○	×	Jump indirect
JSRI	○	×	Jump to subprogram indirect
JMP	○	○	Register jump
JSR	○	○	Register jump to subprogram
RTS	○	○	Link register jump
LD.B	○	○	Load unsigned and extended byte
LD.BS	○	×	Load signed and extended byte
LD.H	○	○	Load unsigned and extended half-word
LD.HS	○	×	Load signed and extended half-word
LD.W	○	○	Load word
LD.D	○	×	Load double word
ST.B	○	○	Store byte
ST.H	○	○	Store half-word
ST.W	○	○	Store word
ST.D	○	×	Store double word
LDR.B	○	×	Load unsigned and extended byte in register offset addressing
LDR.BS	○	×	Load signed and extended byte in register offset addressing
LDR.H	○	×	Load unsigned and extended half-word in register offset addressing
LDR.HS	○	×	Load signed and extended half-word in register offset addressing

LDR.W	o	x	Load word in register offset addressing
STR.B	o	x	Store byte in register offset addressing
STR.H	o	x	Store half-word in register offset addressing
STR.W	o	x	Store word in register offset addressing
LDQ	o	x	Load consecutive quad word
LDM	o	x	Load consecutive multiword
STQ	o	x	Store consecutive quad word
STM	o	x	Store consecutive multiword
PUSH	o	o	Push
POP	o	o	Pop
BPUSH.H	x	o	Binary push of translated half-word
BPUSH.W	x	o	Binary push of translated word
BPOP.H	x	o	Binary pop of translated half-word
BPOP.W	x	o	Binary pop of translated word
NIE	x	o	Interrupt nesting enable
NIR	x	o	Interrupt nesting return
IPUSH	x	o	Interrupt push
IPOP	x	o	Interrupt pop
LDEX.W	o	x	Load word exclusive
STEX.W	o	x	Store word exclusive
LRS.B	o	x	Load byte sign
LRS.H	o	x	Load half-word sign
LRS.W	o	x	Load word sign
SRS.B	o	x	Store byte sign
SRS.H	o	x	Store half-word sign
SRS.W	o	x	Store word sign
CPRGR	o	x	Read transfer from general-purpose register of co-processor
CPWGR	o	x	Write transfer to general-purpose register of co-processor
CPRCR	o	x	Read transfer from control register of co-processor
CPWCR	o	x	Write transfer to control register of co-processor
CPRC	o	x	Read transfer from condition bit of co-processor
LDCPR	o	x	Load word to co-processor
STCPR	o	x	Store word in co-processor
CPOP	o	x	Co-processor operation instruction
MFCR	o	x	Read transfer from control register
MTCR	o	x	Write transfer to control register
PSRSET	o	x	Set PSR bit
PSRCLR	o	x	Clear PSR bit

WAIT	○	×	Enter low power consumption wait mode
DOZE	○	×	Enter low power consumption doze mode
STOP	○	×	Enter low power consumption stop mode
RTE	○	×	Return from abnormal and normal interrupt
RFI	○	×	Return from fast interrupt
STRAP	○	×	Enter safe state
SRTE	○	×	Return from safe state
SYNC	○	×	Synchronize CPU
BKPT	×	○	Breakpoint instruction
SCE	○	×	Set conditional execution
IDLY	○	×	Ban interrupt identification
TRAP	○	×	Unconditional operating system trap
PLDR	○	×	Prefetch read data
PLDW	○	×	Prefetch write data
WE	○	×	Wait event
SE	○	×	Send event

Note: ○ means that the instruction exists in the corresponding instruction set and × means that the instruction does not exist in the corresponding instruction set.

3.2. Functional classification of 16-bit instructions

According to functions of instruction realization, 16-bit instruction set of CSKY can be divided into:

- Data operation instruction
- Branch jump instruction
- Memory access instruction

3.2.1. Data operation instruction

Data operation instruction can be further divided into:

Add-subtract instruction:

Chart 3-2 List of 16-bit add-subtract instructions

ADDU16	Add unsigned
ADDC16	Add with carry unsigned
ADDI16	Add immediate unsigned
SUBU16	Subtract unsigned
SUBC16	Subtract with borrow unsigned
SUBI16	Subtract immediate unsigned

Logical operation instruction:

Chart 3-3 List of 16-bit logical operation instructions

AND16	Bitwise AND
-------	-------------

ANDN16	Bitwise AND-NOT
OR16	Bitwise OR
XOR16	Bitwise XOR
NOR16	Bitwise NOT-OR
NOT16	Bitwise NOT

Shift instruction:

Chart 3-4 List of 16-bit shift instructions

LSL16	Logical shift left
LSLI16	Logical shift left immediate
LSR16	Logical shift right
LSRI16	Logical shift right immediate
ASR16	Arithmetic shift right
ASRI16	Arithmetic shift right immediate
ROTL16	Rotate left

Compare instruction:

Chart 3-5 List of 16-bit compare instructions

CMPNE16	Compare unequal
CMPNEI16	Compare unequal immediate
CMPHS16	Compare unsigned when greater or equal
CMPHSI16	Compare immediate unsigned when greater or equal
CMPLT16	Compare signed when smaller
CMPLTI16	Compare immediate signed when smaller
TST16	Null-test
TSTNBZ16	Register test without byte equal to zero

Data transfer instruction:

Chart 3-6 List of 16-bit data transfer instructions

MOV16	Move
MOVI16	Move immediate
MVCV16	C bit reverse move
LRW16	Memory read-in

Bit operation instruction:

Chart 3-7 List of 16-bit bit operation instructions

BCLRI16	Bit clear immediate
---------	---------------------

BSETI16	Bit set immediate
BTSTI16	Bit test immediate

Extract and insert instruction:

Chart 3-8 List of 16-bit extract and insert instructions

ZEXTB16	Extract byte and extend unsigned
ZEXTH16	Extract half-word and extend unsigned
SEXTB16	Extract byte and extend signed
SEXTH16	Extract half-word and extend signed
REVB16	Byte-reverse
REVH16	Half-word byte-reverse

Multiply-divide instruction:

Chart 3-9 List of 16-bit multiply-divide instructions

MULT16	Multiply
MULSH16	16-bit multiply signed

3.2.2. Branch jump instruction

Branch jump instruction can be further divided into:

Branch instruction:

Chart 3-10 List of 16-bit branch instructions

BT16	C=1 branch instruction
BF16	C=0 branch instruction

Jump instruction:

Chart 3-11 List of 16-bit jump instructions

BR16	Unconditional jump
JMP16	Register jump
JSR16	Register jump to subprogram
RTS16	Link register jump
JMPIX16	Register index jump

3.2.3. Memory access instruction

Memory access instruction can be further divided into:

Immediate operand offset access instruction:

Chart 3-12 List of 16-bit immediate operand offset access instructions

LD16.B	Load unsigned and extended byte
LD16.H	Load unsigned and extended half-word
LD16.W	Load word
ST16.B	Store byte
ST16.H	Store half-word
ST16.W	Store word

Multi-register access instruction:

Chart 3-13 List of 16-bit multi-register access instructions

POP16	Pop
PUSH16	Push

Binary translated stack instructions:

Chart 3-14 List of 16-bit binary translated stack instructions

BPUSH16.H	Binary push of translated half-word
BPUSH16.W	Binary push of translated word
BPOP16.H	Binary pop of translated half-word
BPOP16.W	Binary pop of translated word

Interrupt nesting acceleration instruction:

Chart 3-15 List of 16-bit interrupt nesting acceleration instruction

NIE	Interrupt nesting enable
NIR	Interrupt nesting return
IPUSH	Interrupt push
IPOP	Interrupt pop

3.3. Coding mode of 16-bit instructions

The 16-bit instruction set of CSKY is almost consistent with the subset of 32-bit instructions in coding style and it can be divided into three categories:

- Jump type (J type)
- Immediate operand type (I type)
- Register type (R type)

3.3.1. Jump type

The coding mode of jump type (J type) is shown in the following chart:

15 14 13	10 9	0
----------	------	---

0 0	OP	Offset
2	4	10

OP field is the main operation code and instructions of this coding type can be identified through 4-bit main operation code; Offset field is the offset of jump instruction.

3.3.2. Immediate operand type

Immediate operand type (I type) covers four coding modes including 3-bit immediate operand, 5-bit immediate operand, 7-bit immediate operand, and 8-bit immediate operand.

The coding mode of 3-bit immediate operand is shown in the following chart:

15 14 13	11 10	8 7	5 4	2 1	0
0 1	OP	RX	RZ	IMM3	SOP

2	3	3	3	3	2
---	---	---	---	---	---

OP field is the main operation code and the instruction or instruction type can be identified through 3-bit main operation code; RZ field is the destination register field; IMM3 field is the 3-bit immediate operand; SOP field is the sub-operation code field. The instruction type can be gained after decoding the main operation code OP, and the specific instruction can be obtained only after further decoding the sub-operation code SOP.

The coding mode of 5-bit immediate operand has three formats and the first format is shown in the following chart:

15 14 13	11 10	8 7	5 4	0
0 1	OP	RX	RZ	IMM5

2	3	3	3	5
---	---	---	---	---

OP field is the main operation code and the instruction or instruction type can be identified through 3-bit main operation code; RX field is the source register field; RZ field is the destination register field; IMM5 field is the 5-bit immediate operand.

The second coding mode of 5-bit immediate operand is shown in the following chart:

15 14 13	11 10	8 7	5 4	0
1 0	OP	RX	RZ	IMM5

2	3	3	3	5
---	---	---	---	---

OP field is the main operation code and the instruction or instruction type can be identified through 3-bit main operation code; RX field is the source register field; RZ field is the destination register field; IMM5 field is the 5-bit immediate operand.

The third coding mode of 5-bit immediate operand is shown in the following chart:

15 14 13	11 10	8 7	5 4	0
0 0	OP	RX	SOP	IMM5
2	3	3	3	5

OP field is the main operation code and the instruction or instruction type can be identified through 3-bit main operation code; RX field is the source register field; SOP field is the sub-operation code field; IMM5 field is the 5-bit immediate operand. The instruction type can be gained after decoding the main operation code OP, and the specific instruction can be obtained only after further decoding the sub-operation code SOP.

The coding mode of 7-bit immediate operand is shown in the following chart:

15 14 13	10 9	8 7	5 4	0
0 0	OP	IMM2	SOP/RZ	IMM5
2	4	2	3	5

OP field is the main operation code and the instruction or instruction type can be identified through 4-bit main operation code; IMM2 field and IMM5 field are two high bits and five low bits of 7-bit immediate operand; SOP/RZ field is the sub-operation code field or destination register field. The instruction type can be gained after decoding the main operation code OP, and the specific instruction can be obtained only after further decoding the sub-operation code SOP.

The coding mode of 8-bit immediate operand has two formats and the first format is shown in the following chart:

15 14 13	11 10	8 7	0
0 0	OP	RX/RZ	IMM8
2	3	3	8

OP field is the main operation code and the instruction or instruction type can be identified through 3-bit main operation code; RZ/RX field is the destination register field or source register field; IMM8 field is the 8-bit immediate operand.

The second coding mode of 8-bit immediate operand is shown in the following chart:

15 14 13	11 10	8 7	5 4	0
1 0	OP	IMM3	RZ	IMM5
2	3	3	3	5

OP field is the main operation code and the instruction or instruction type can be identified through 3-bit main operation code; IMM3 field and IMM5 field are three high bits and five low bits of 8-bit immediate operand; RZ field is the destination register field.

3.3.3. Register type

Register type (R type) covers two coding modes including 3-bit operand and 2-bit operand.

The coding mode of 3-bit operand is shown in the following chart:

15	14	13	11	10	8	7	5	4	2	1	0
0	1	OP	RX	RZ	RY	SOP					
2	3	3	3	3	3	2					

OP field is the main operation code and the instruction or instruction type can be identified through 3-bit main operation code; RX field is the first source register field; RZ field is the destination register field; RY field is the second source register field; SOP field is the sub-operation code field. The instruction type can be gained after decoding the main operation code OP, and the specific instruction can be obtained only after further decoding the sub-operation code SOP.

The coding mode of 2-bit operand is shown in the following chart:

15	14	13	10	9	6	5	2	1	0
0	1	OP	RZ/RY	RX	SOP				
2	4	4	4	4	2				

OP field is the main operation code and the instruction or instruction type can be identified through 4-bit main operation code; RZ/RY field is the destination register field and second source register field; RX field is first source register field; SOP field is the sub-operation code field. The instruction type can be gained after decoding the main operation code OP, and the specific instruction can be obtained only after further decoding the sub-operation code SOP.

3.4. Operand addressing mode of 16-bit instructions

The 16-bit instruction set of CSKY follows three instruction coding modes and each coding mode has its own operand addressing mode. In the following, various operand addressing modes will be introduced.

3.4.1. Addressing mode of jump-type instructions

The 16-bit instructions of jump type in CSKY only have one addressing mode.

3.4.1.1. Addressing mode of 10-bit immediate operand

In the instructions that adopt the addressing mode of 10-bit immediate operand, there is an immediate operand field with the length of 10 bits. This field is used to generate destination address as offset. Instructions of this format include br16, bt16 and bf16.

15	14	13	10	9	0
----	----	----	----	---	---

0 0	OP	Offset
2	4	10

3.4.2. Addressing mode of immediate operand-type instructions

The 16-bit instructions of immediate operand type in CSKY have six addressing modes.

3.4.2.1. Addressing mode of two register 3-bit immediate operand

In the instructions that adopt the addressing mode of two register 3-bit immediate operand, RX field is source register field; RZ field is the destination register field; IMM3 field can also directly participate in data operation as 3-bit immediate operand; SOP field is the sub-operation code field. Instructions of this format include addi16 and subi16.

15 14 13	11 10	8 7	5 4	2 1	0
0 1	OP	RX	RZ	IMM3	SOP

2	3	3	3	3	2
---	---	---	---	---	---

3.4.2.2. Addressing mode of two register 5-bit immediate operand

The instructions that adopt the addressing mode of two register 5-bit immediate operand can be further divided into two formats.

In the first format, RX field is source register field; RZ field is the destination register field; IMM5 field can also directly participate in data operation as 5-bit immediate operand; SOP field is the sub-operation code field. Instructions of this format include lsl16, lsri16 and asrl16.

15 14 13	11 10	8 7	5 4	0
0 1	OP	RX	RZ	IMM5

2	3	3	3	5
---	---	---	---	---

In the second format, RX field is source register field; RZ field is the destination register field; IMM5 field can also directly participate in data operation as 5-bit immediate operand; SOP field is the sub-operation code field. Instructions of this format include st16.b, st16.h, st16.w, ld16.b, ld16.h and ld16.w.

15 14 13	11 10	8 7	5 4	0
1 0	OP	RX	RZ	IMM5

2	3	3	3	5
---	---	---	---	---

3.4.2.3. Addressing mode of single register 5-bit immediate operand

In the instructions that adopt the addressing mode of single register 5-bit immediate operand, RX field is the source register field or destination register; SOP field is the sub-operation code

field. Instructions of this format include cmphsi16, cmplti16, cmpnei16, bclri16, bseti16 and btsti16.

15	14	13	11	10	8	7	5	4	0
0	0	OP	RX	SOP	IMM5				
2	3	3	3	3	5				

3.4.2.4. Addressing mode of single register 7-bit immediate operand

In the instructions that adopt the addressing mode of single register 7-bit immediate operand, RZ field is the destination register field; IMM2 field and IMM5 field can be combined into 7-bit immediate operand to directly participate in data operation. Instruction of this format includes lrw16.

15	14	13	10	9	8	7	5	4	0
0	0	OP	IMM2	RZ	IMM5				
2	4	2	3	3	5				

3.4.2.5. Addressing mode of 7-bit immediate operand

In the instructions that adopt the addressing mode of 7-bit immediate operand, IMM2 field and IMM5 field can be combined into 7-bit immediate operand to directly participate in data operation; SOP field is the sub-operation code field. Instructions of this format include push16, pop16, bpush16.h, bpush16.w, bpop16.h, bpop16.w, addi16(SP) and subi16(SP).

15	14	13	10	9	8	7	5	4	0
0	0	OP	IMM2	SOP	IMM5				
2	4	2	3	3	5				

3.4.2.6. Addressing mode of single register 8-bit immediate operand

The instructions that adopt the addressing mode of single register 8-bit immediate operand can be further divided into three formats.

In the first format, RZ field is the destination register field; IMM8 field can also directly participate in data operation as 8-bit immediate operand; SOP field is the sub-operation code field. Instructions of this format include addi16(SP), subi16(SP) and movi16.

15	14	13	11	10	8	7	0
0	0	OP	RZ		IMM8		
2	3	3	8				

In the second format, RZ field is the source register field or destination register field; IMM

field can also directly participate in data operation as 8-bit immediate operand; SOP field is the sub-operation code field. Instructions of this format include addi16 and subi16.

15 14 13	11 10	8 7	0
0 0	OP	RZ	IMM8
2 3	3	8	

In the third format, RZ field is the source register field or destination register field; IMM3 field and IMM5 field can be combined into 8-bit immediate operand to directly participate in data operation. Instructions of this format include st16.w(SP) and ld16.w(SP).

15 14 13	11 10	8 7	5 4	0
1 0	OP	IMM3	RZ	IMM5
2 3	3	3	5	

3.4.3. Addressing mode of register-type instructions

The 16-bit instructions of register type in CSKY have three addressing modes.

3.4.3.1. Addressing mode of ternary register

In the instructions that adopt the addressing mode of ternary register, the two register fields RX and RY are the first source register field and second source register field respectively; RZ field is the destination register field; SOP field is the sub-operation code field. Instructions of this format include addu16 and subu16.

15 14 13	11 10	8 7	5 4	2 1 0	
0 1	OP	RX	RZ	RY	SOP
2	3	3	3	3	2

3.4.3.2. Addressing mode of two register

The instructions that adopt the addressing mode of two register can be further divided into three formats.

In the first format, the two register fields RX and RY are the first source register field and second source register field respectively; SOP field is the sub-operation code field. Instructions of this format include cmphs16, cmplt16, cmpne16 and tst16.

15 14 13	10 9	6 5	2 1 0	
0 1	OP	RY	RX	SOP
2	4	4	4	2

In the second format, RZ field is the destination register field; RX is the source register;

SOP field is the sub-operation code field. Instructions of this format include mov16, zextb16, zexth16, sextb16, sexth16, revb16 and revh16.

15	14	13	10	9	6	5	2	1	0
0	1	OP	RZ		RX	SOP			
2	4	4	4	4	2				

In the third format, RZ field is the destination register field and second source register field; RX field is the first source register field; SOP field is the sub-operation code field. Instructions of this format include addu16, addc16, subu16, subc16, and16, andn16, or16, xor16, nor16, lsl16, lsr16, asr16, rotl16, mult16 and mulsh16.

15	14	13	10	9	6	5	2	1	0
0	1	OP	RZ		RX	SOP			
2	4	4	4	4	2				

3.4.3.3. Addressing mode of single register

The instructions that adopt the addressing mode of single register can be further divided into two formats.

In the first format, RX field is the source register field; SOP field is the sub-operation code field. Instructions of this format include tstnbz16, jmp16 and jsr16.

15	14	13	10	9	6	5	2	1	0
0	1	OP		0	RX	SOP			
2	4	4	4	4	2				

In the second format, RZ field is the destination register field; SOP field is the sub-operation code field. Instruction of this format includes mvcv16.

15	14	13	10	9	6	5	2	1	0
0	1	OP	RZ		0	SOP			
2	4	4	4	4	2				

4. Floating point instruction set

In this chapter, floating point instruction set of CSKY is introduced, covering functional classification, encoding and addressing mode of floating point instruction set.

4.1. Functional classification of floating point instructions

According to functions of instruction realization, floating point instruction set of CSKY can be divided into:

- Single-precision data operation instruction
- Double-precision data operation instruction
- Vector operation instruction
- Transfer instruction
- Memory access instruction

4.1.1. Data operation instruction

Chart 4-1 List of single-precision data operation instructions

FSTOSI	Transform single-precision floating point into signed integer
FSTOUI	Transform single-precision floating point into unsigned integer
FSITOS	Transform signed integer into single-precision floating point
FUITOS	Transform unsigned integer into single-precision floating point
FCMPZHSS	Single-precision floating point compare when greater than or equal to zero
FCMPZLSS	Single-precision floating point compare when smaller than or equal to zero
FCMPZNES	Single-precision floating point compare when not equal to zero
FCMPZUOS	Judge whether the single operand of single-precision floating point is NaN
FCMPHSS	Single-precision floating point compare when greater than or equal
FCMPLTS	Single-precision floating point compare when smaller
FCMPNES	Single-precision floating point compare when not equal
FCMPUOS	Judge whether the double operand of single-precision floating point is NaN
FMOVS	Single-precision floating point move

FABSS	Single-precision floating point absolute value
FNEGS	Single-precision floating point negate
FSQRTS	Single-precision floating point square root
FRECIPS	Single-precision floating point reciprocal
FADDS	Single-precision floating point add
FSUBS	Single-precision floating point subtract
FMULS	Single-precision floating point multiply
FDIVS	Single-precision floating point division
FMACS	Single-precision floating point multiply-accumulate
FMSCS	Single-precision floating point multiply-subtract
FNMACS	Single-precision floating point multiply-negate-accumulate
FNMSCS	Single-precision floating point multiply-negate-subtract
FNMULS	Single-precision floating point multiply-negate

Chart 4-2 List of double-precision data operation instructions

FDTOSI	Transform double-precision floating point into signed integer
FDTOUI	Transform double-precision floating point into unsigned integer
FSITOD	Transform signed integer into double-precision floating point
FUITOD	Transform unsigned integer into double-precision floating point
FDTOS	Transform double-precision floating point into single-precision floating point
FSTOD	Transform single-precision floating point into double-precision floating point
FCMPZHSD	Double-precision floating point compare when greater than or equal to zero
FCMPZLSD	Double-precision floating point compare when smaller than or equal to zero
FCMPZNED	Double-precision floating point compare when not equal to zero
FCMPZUOD	Judge whether the single operand of double-precision floating point is NaN
FCMPHSD	Double-precision floating point compare when greater than or equal

FCMPLTD	Double-precision floating point compare when smaller
FCMPNED	Double-precision floating point compare when not equal
FCMPUOD	Judge whether the double operand of double-precision floating point is NaN
FMOVD	Double-precision floating point move
FABSD	Double-precision floating point absolute value
FNEGD	Double-precision floating point negate
FSQRTD	Double-precision floating point square root
FRECIPD	Double-precision floating point reciprocal
FADDD	Double-precision floating point add
FSUBD	Double-precision floating point subtract
FMULD	Double-precision floating point multiply
FDIVD	Double-precision floating point division
FMACD	Double-precision floating point multiply-accumulate
FMSCD	Double-precision floating point multiply-subtract
FNMACD	Double-precision floating point multiply-negate-accumulate
FNMSCD	Double-precision floating point multiply-negate-subtract
FNMULD	Double-precision floating point multiply-negate

4.1.2. Vector operation instruction

Chart 4-3 List of vector operation instructions

FMOV	SIMD single-precision floating point move
FABSM	SIMD single-precision floating point absolute value
FNEGM	SIMD single-precision floating point negate
FADDM	SIMD single-precision floating point add
FSUBM	SIMD single-precision floating point subtract
FMULM	SIMD single-precision floating point multiply
FMACM	SIMD single-precision floating point multiply-accumulate
FMSCM	SIMD single-precision floating point

	multiply-subtract
FNMACM	SIMD single-precision floating point multiply-negate-accumulate
FNMSCM	SIMD single-precision floating point multiply-negate-subtract
FNMULM	SIMD single-precision floating point multiply-negate

4.1.3. Transfer instruction

Chart 4-4 List of data transfer instructions

FMTVRL	Write transfer to low word of floating point register
FMTVRH	Write transfer to high word of floating point register
FMFVRL	Read transfer low word from floating point register
FMFVRH	Read transfer high word from floating point register
FLRWS	Single-precision floating point storage read-in
FLRWD	Double-precision floating point storage read-in

4.1.4. Memory access instruction

Chart 4-5 List of data transfer instructions

FLDS	Load single-precision floating point
FLDD	Load double-precision floating point
FLDM	Load vector floating point
FLDRS	Load single-precision floating point in register offset addressing
FLDRD	Load double-precision floating point in register offset addressing
FLDRM	Load vector floating point in register offset addressing
FLDMS	Load consecutive single-precision floating point
FLDMD	Load consecutive double-precision floating

	point
FLDMM	Load consecutive vector floating point
FSTS	Store single-precision floating point
FSTD	Store double-precision floating point
FSTM	Store vector floating point
FSTRS	Store single-precision floating point in register offset addressing
FSTRD	Store double-precision floating point in register offset addressing
FSTRM	Store vector floating point in register offset addressing
FSTMS	Store consecutive single-precision floating point
FSTMD	Store consecutive double-precision floating point
FSTMM	Store consecutive vector floating point

4.2. Coding mode of floating point instructions

Floating point instruction set of CSKY only has one category in the coding style:

- Register type (R type)

4.2.1. Register type

The coding mode of register type (R type) of 32-bit floating point instructions is shown in the following chart:

31	26	25	24	21	20	16	15	10	9	5	4	0
1	1	Op	0	VRY	0	VRX		SOP		0	VRZ	

OP field is the main operation code and the instruction type can be identified through 4-bit main operation code; VRY refers to the code of the second operand register and there are only 16 vector general-purpose registers, so a 4-bit code is required (VRX and VRZ are similar); VRX refers to the first operand register; VRZ means destination register. SOP signifies the operation code of the instruction.

4.3. Operand addressing mode of floating point instructions

Generally speaking, floating point instructions of CSKY follow one instruction coding mode, and such coding mode has its own operand addressing mode. In the following, this

operand addressing mode will be introduced.

4.3.1. Addressing mode of register-type instructions

The floating point instructions of register type in CSKY have four addressing modes.

4.3.1.1. Addressing mode of single register

Floating point instruction set of CSKY includes several instructions that adopt the addressing mode of single register. These instructions are mainly instructions of comparing floating point data with 0, covering FCMPZHSS, FCMPZLSS, FCMPZNES, FCMPZUOS, FCMPZHSD, FCMPZLSD, FCMPZNED and FCMPZUOD.

31	26 25 24	21 20	16 15	10 9	5 4	0
1 1	OP	0 0 0 0 0	VRX	SOP	0 0 0 0 0	
2	4	4	4	11	4	

4.3.1.2. Addressing mode of two register

Floating point instruction set of CSKY includes several instructions that adopt the addressing mode of two register. The first category is the register addressing mode including 1 source operand and 1 destination operand; the second category is the register addressing mode including 2 source operands.

Instructions of the first category include FSTOSI, FSTOUI, FSITOS, FUITOS, FDTOSI, FDTOUI, FSITOD, FUITOD, FDTOS, FSTOD, FMOVS, FABSS, FNEGS, FSQRTS, FRECIPS, FMOVD, FABSD, FNEGD, FSQRTD, FRECIPD, FMOVVM, FABSM, FNEGM, FMTVRH, FMTVRH, FMTVRL, FMFVRH and FMFVRL. The specific format is as follows:

31	26 25 24	21 20 19	16 15	5 4	0
1 1	OP	0 0 0 0 0	VRX	0 0 0 0 0 0 1 1 0 1 0 0	VRZ
2	4	4	4	11	4

Instructions of the second category include FCMPHSS, FCMPLTS, FCMPNES, FCMPUOS, FCMPHSD, FCMPLTD, FCMPNED and FCMPUOD. The specific format is as follows:

31	26 25 24	21 20	16 15	10 9	5 4	0
1 1	OP	0	VRY	0	VRX	0 0 0 0 0 1 0 0 1 0 0 0
2	4	4	4	11	4	

4.3.1.3. Addressing mode of ternary register

Instructions that adopt the addressing mode of ternary register cover 2 source operands and 1 destination operand, and such instructions include FADDS, FSUBS, FMULS, FDIVS, FMACS, FMSCS, FNMACS, FNMSCS, FNMLUS, FADDD, FSUBD, FMULD, FDIVD, FMACD, FMSCD, FNMACD, FNMSCD, FNMLUD, FADDM, FSUBM, FMULM, FMACM, FMSCM, FNMACM, FNMSCM, FNMLUM, FLDRS, FLDRD, FSTRM, FSTRD and FSTRM.

31	26 25 24	21 20 19	16 15	5 4	0
1 1	OP	0 VRY	0 VRX	0 0 0 0 1 1 1 1 1 0	0 VRZ
2 4	4	4	4	11	4

4.3.1.4. Addressing mode of two register 8-bit immediate operand

In the instructions that adopt the addressing mode of two register 8-bit immediate operand, RX field is the first source register field; VRZ field is the vector destination register field; SOP field or S field is the sub-operation code field; IMM8 field is the 8-bit immediate operand. Instructions of this format include FLDS, FLDD, FLDM, FSTS, FSTD, FSTM, FLRWS and FLRWD.

31 30	26 25	21 20	16 15	8 7	4 3	0
1	OP	0 IMM4H	RX	SOP	IMM4L	VRZ

5. Term list of basic instructions

Specific descriptions of each CSKY instruction are provided in the following and each instruction is described in details according to the alphabetical order.

At the end of mnemonic symbol in each CSKY instruction, the figure “32” or “16” is used to represent bit width of the instruction. For instance, “addc32” means that this instruction is 32-bit instruction of add with carry unsigned, and “addc16” means that this instruction is 16-bit instruction of add with carry unsigned.

If bit width (such as “addc”) of the instruction in mnemonic symbol is omitted, the system will automatically compile it into the optimized instruction.

Among them, instructions carrying # in the Chinese name are pseudo instructions.

ABS – Absolute value

**Unified
instruction**

Grammar	Operation	Compiling result
rz, rx	RZ \leftarrow RX	Only 32-bit instructions exist. abs32 rz, rx

Description: Take the absolute value of RX value and save the result in RZ.

Attention: The result of operand 0x80000000 is 80000000.

Influence on flag No influence

bit:

Exception: None

**32-bit
instruction**

Operation: RZ \leftarrow |RX|

Grammar: abs32 rz, rx

Description: Take the absolute value of RX value and save the result in RZ.

Attention: The result of operand 0x80000000 is 80000000.

Influence on flag No influence

bit:

Exception: None

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0				
1	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	RZ

ADDC – Add with carry unsigned

**Unified
instruction**

Grammar	Operation	Compiling result
addc rz, rx	$RZ \leftarrow RZ + RX + C$, $C \leftarrow \text{carry}$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then addc16 rz, rx; else addc32 rz, rx;
addc rz, rx, ry	$RZ \leftarrow RX + RY + C$, $C \leftarrow \text{carry}$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($y == z$) and ($x < 16$) and ($z < 16$), then addc16 rz, rx; else addc32 rz, rx, ry;

Description: Add the values in RZ/RY, RX and C bits, save the result in RZ, and save the carry in C bit.

Influence on flag bit: $C \leftarrow \text{carry}$

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow RZ + RX + C$, $C \leftarrow \text{carry}$

Grammar: addc16 rz, rx

Description: Add the values in RZ, RX and C bits, save the result in RZ, and save the carry in C bit.

Influence on flag bit: $C \leftarrow \text{carry}$

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 0 0	RZ	RX	0 1

32-bit

instruction

Operation: $RZ \leftarrow RX + RY + C$, $C \leftarrow \text{carry}$

Grammar: addc32 rz, rx, ry

Description: Add the values in RX, RY and C bits, save the result in RZ, and save the carry in C bit.

Influence on flag C \leftarrow carry

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 0 0 0 0	0 0 0 1 0	RZ	

ADDI - Add immediate unsigned

**Unified
instruction**

Grammar	Operation	Compiling result
addi rz, oimm12	$RZ \leftarrow RZ +$ zero_extend(OIMM12)	Compiled into corresponding 16-bit or 32-bit instructions according to the range of immediate operand and register. if ($z < 8$) and ($oimm12 < 257$), addi16 rz, oimm8; else addi32 rz, rz, oimm12;
addi rz, rx, oimm12	$RZ \leftarrow RX +$ zero_extend(OIMM12)	Compiled into corresponding 16-bit or 32-bit instructions according to the range of immediate operand and register. if ($oimm12 < 9$) and ($z < 8$) and ($x < 8$), addi16 rz, rx, oimm3; elsif ($oimm12 < 257$) and ($x == z$) and ($z < 8$), addi16 rz, oimm8; else addi32 rz, rx, oimm12;
addi rz, r28, oimm18	$RZ \leftarrow R28 +$ zero_extend(OIMM18)	Only 32-bit instructions exist. addi32 rz, r28, oimm18;

Description: Zero-extend the immediate operand with offset 1 to 32 bits, add it to RX/RZ value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: If the source register is R28, the range of immediate operand is 0x1-0x40000.
If the source register is not R28, the range of immediate operand is 0x1-0x1000.

Exception: None

16-bit

instruction ----1

Operation: $RZ \leftarrow RZ + \text{zero_extend(OIMM8)}$

Grammar: addi16 rz, oimm8

Description: Zero-extend the 8-bit immediate operand with offset 1 (OIMM8) to 32 bits, add it to RZ value, and save the result in RZ.

Attention: The binary operand IMM8 is equal to OIMM8 – 1.

Influence on No influence

flag bit:

Restriction: The range of register is r0-r7; the range of immediate operand is 1-256.

Exception: None

Instruction

format:

15	14	11	10	8	7	0
0	0	1	0	0	RZ	IMM8

IMM8 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM8, the value OIMM8 added into the register requires offset 1.

00000000 – +1

00000001 – +2

.....

11111111 – +256

16-bit
instruction ----2

Operation: $RZ \leftarrow RX + \text{zero_extend(OIMM3)}$

Grammar: addi16 rz, rx, oimm3

Description: Zero-extend the 3-bit immediate operand with offset 1 (OIMM3) to 32 bits, add it to RX value, and save the result in RZ.

Attention: The binary operand IMM3 is equal to OIMM3 – 1.

Influence on No influence

flag bit:

Restriction: The range of register is r0-r7; the range of immediate operand is 1-8.

Exception: None

Instruction

format:

15 14	10	8 7	5 4	2 1	0
0 1 0 1 1	RX	RZ	IMM3	1 0	

IMM3 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM3, the value OIMM3 added into the register requires offset 1.

000 – +1

001 – +2

.....

111 – +8

32-bit

instruction ----1

Operation: $RZ \leftarrow RX + \text{zero_extend(OIMM12)}$

Grammar: addi32 rz, rx, oimm12

Description: Zero-extend the 12-bit immediate operand with offset 1 (OIMM12) to 32 bits, add it to RX value, and save the result in RZ.

Attention: The binary operand IMM12 is equal to OIMM12 – 1.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0x1-0x1000.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
1 1 1 0 0 1	RZ	RX	0 0 0 0	IMM12	

IMM12 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM12, the value OIMM12 added into the register requires offset 1.

00000000000000 – +0x1
 00000000000001 – +0x2

 111111111111 – +0x1000

32-bit

instruction ----2

Operation: $RZ \leftarrow R28 + \text{zero_extend(OIMM18)}$

Grammar: addi32 rz, r28, oimm18

Description: Zero-extend the 18-bit immediate operand with offset 1 (OIMM18) to 32 bits, add it to R28 value, and save the result in RZ.

Attention: The binary operand IMM18 is equal to OIMM18 – 1.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0x1-0x40000.

Exception: None

Instruction

format:

31	30	26	25	21	20	18	17	0	
1	1	0	0	1	1	RZ	1	1	IMM18

IMM18 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM18, the value OIMM18 added into the register requires offset 1.

0000000000000000 – +0x1
 0000000000000001 – +0x2

 11111111111111 – +0x40000

ADDI(SP) – Add immediate unsigned (stack pointer)

Unified instruction

Grammar	Operation	Compiling result
addi rz, sp, imm	$RZ \leftarrow SP +$ zero_extend(IMM)	Only 16-bit instructions exist. addi rz, sp, imm
addi sp, sp, imm	$SP \leftarrow SP +$ zero_extend(IMM)	Only 16-bit instructions exist. addi sp, sp, imm

Description: Zero-extend the immediate operand (IMM) to 32 bits, add it to stack pointer (SP) value, and save the result in RZ or SP.

Influence on flag bit: No influence

Restriction: The range of register is r0-r7; the range of immediate operand is 0x0-0x3fc.

Exception: None

16-bit

instruction ----1

Operation: $RZ \leftarrow SP +$ zero_extend(IMM)

Grammar: addi16 rz, sp, imm8

Description: Zero-extend the immediate operand (IMM) to 32 bits, add it to stack pointer (SP) value, and save the result in RZ.

Attention: The immediate operand (IMM) is equal to the binary operand IMM8 << 2.

Influence on flag bit: No influence

Restriction: The range of register is r0-r7; the range of immediate operand is (0x0-0xff) << 2.

Exception: None

Instruction format:

15 14 11 10 8 7 0

0	0 0 1 1	RZ	IMM8
---	---------	----	------

IMM8 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM8, the value IMM added into the register needs to shift left by 2 bits.

00000000 – +0x0

00000001 – +0x4

.....

11111111 – +0x3fc

16-bit

instruction ----2

Operation: $SP \leftarrow SP + \text{zero_extend}(\text{IMM})$

Grammar: addi16 sp, sp, imm

Description: Zero-extend the immediate operand (IMM) to 32 bits, add it to stack pointer (SP) value, and save the result in SP.

Attention: The immediate operand (IMM) is equal to the binary operand $\{\text{IMM2}, \text{IMM5}\} \ll 2$.

Influence on No influence

flag bit:

Restriction: Both source and destination registers are stack pointer registers (R14); the range of immediate operand is $(0x0-0x7f) \ll 2$.

Exception: None

Instruction

format:

15	14	11	10	9	8	7	5	4	0
0	0	0	1	0	1	IMM2	0	0	0

IMM field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand $\{\text{IMM2}, \text{IMM5}\}$, the value IMM added into the register needs to shift left by 2 bits.

{00, 00000} – +0x0

{00, 00001} – +0x4

.....

{11, 11111} – +0x1fc

ADDU – Add unsigned

**Unified
instruction**

Grammar	Operation	Compiling result
addu rz, rx	$RZ \leftarrow RZ + RX$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 16$) and ($x < 16$), then addu16 rz, rx; else addu32 rz, rx, rx;
addu rz, rx, ry	$RZ \leftarrow RX + RY$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 8$) and ($x < 8$) and ($y < 8$), then addu16 rz, rx, ry; elseif ($y == z$) and ($x < 16$) and ($z < 16$), then addu16 rz, rx; else addu32 rz, rx, ry;

Description: Add the values of RZ/RY and RX, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

**16-bit
instruction ----1**

Operation: $RZ \leftarrow RZ + RX$

Grammar: addu16 rz, rx

Description: Add the values of RZ and RX, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 0 0	RZ	RX	0 0

16-bit

instruction ----2

Operation: $RZ \leftarrow RX + RY$

Grammar: addu16 rz, rx, ry

Description: Add the values of RX and RY, and save the result in RZ.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7.

Exception: None

Instruction

format:

15 14	11 10	8 7	5 4	2 1 0
0 1 0 1 1	RX	RZ	RY	0 0

32-bit instruction

Operation: $RZ \leftarrow RX + RY$

Grammar: addu32 rz, rx, ry

Description: Add the values of RX and RY, and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

3130 2625 2120 1615 109 54 0

1	1 0 0 0 1	RY	RX	0 0 0 0 0 0	0 0 0 0 1	RZ
---	-----------	----	----	-------------	-----------	----



AND – Bitwise AND

**Unified
instruction**

Grammar	Operation	Compiling result
and rz, rx	$RZ \leftarrow RZ \text{ and } RX$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then and16 rz, rx; else and32 rz, rz, rx;
and rz, rx, ry	$RZ \leftarrow RX \text{ and } RY$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($y == z$) and ($x < 16$) and ($z < 16$), then and16 rz, rx; else and32 rz, rx, ry;

Description: Perform a bitwise-AND of the values of RZ/RY and RX, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

16-bit instruction

Operation: $RZ \leftarrow RZ \text{ and } RX$

Grammar: and16 rz, rx

Description: Perform a bitwise-AND of the values of RZ and RX, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 1 0	RZ	RX	0 0

32-bit instruction

Operation: RZ \leftarrow RX and RY

Grammar: and32 rz, rx, ry

Description: Perform a bitwise-AND of the values of RX and RY, and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 1 0 0 0	0 0 0 0 1	RZ	

ANDI – Bitwise AND immediate

Unified instruction

Grammar	Operation	Compiling result
andi rz, rx, imm16	$RZ \leftarrow RX \text{ and zero_extend(IMM12)}$	Only 32-bit instructions exist. andi32 rz, rx, imm12

Description: Zero-extend the 12-bit immediate operand to 32 bits, perform a bitwise-AND with RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow RX \text{ and zero_extend(IMM12)}$

Grammar: andi32 rz, rx, imm12

Description: Zero-extend the 12-bit immediate operand to 32 bits, perform a bitwise-AND with RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	12	11	0
1	1	1	0	0	1	RZ	RX	0	0	IMM12

ANDN – Bitwise AND-NOT

Unified instruction

Grammar	Operation	Compiling result
andn rz, rx	$RZ \leftarrow RZ \text{ and } (!RX)$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then andn16 rz, rx; else andn32 rz, rz, rx;
andn rz, rx, ry	$RZ \leftarrow RX \text{ and } (!RY)$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x == z$) and ($y < 16$) and ($z < 16$), then andn16 rz, ry; else andn32 rz, rz, rx;

Description: For andn rz, rx, perform a bitwise-AND of RZ value and negative value of RX, and save the result in RZ; for andn rz, rx, ry, perform a bitwise-AND of RX value and negative value of RY, and save the result in RZ.

Influence on No influence

flag bit:

Exception: None

16-bit instruction

Operation: $RZ \leftarrow RZ \text{ and } (!RX)$

Grammar: andn16 rz, rx

Description: Perform a bitwise-AND of RZ value and negative value of RX, and save the result in RZ

Influence on No influence

flag bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 1 0	RZ	RX	0 1

32-bit

instruction

Operation: $RZ \leftarrow RX \text{ and } (!RY)$

Grammar: andn32 rz, rx, ry

Description: Perform a bitwise-AND of RX value and negative value of RY, and save the result in RZ.

Influence on No influence

flag bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 1 0 0 0	0 0 0 1 0	RZ	

ANDNI – Bitwise AND-NOT immediate

Unified instruction

Grammar	Operation	Compiling result
andni rz, rx, imm16	$RZ \leftarrow RX \text{ and } !(\text{zero_extend(IMM12)})$	Only 32-bit instructions exist. andni32 rz, rx, imm12

Description: Zero-extend the 12-bit immediate operand to 32 bits, perform a bitwise NOT, perform a bitwise-AND with RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow RX \text{ and } !(\text{zero_extend(IMM12)})$

Grammar: andni32 rz, rx, imm12

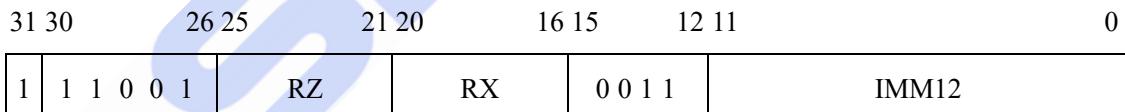
Description: Zero-extend the 12-bit immediate operand to 32 bits, perform a bitwise NOT, perform a bitwise-AND with RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

Instruction format:



ASR – Arithmetic shift right

**Unified
instruction**

Grammar	Operation	Compiling result
asr rz, rx	$RZ \leftarrow RZ >>> RX[5:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then asr16 rz, rx; else asr32 rz, rx;
asr rz, rx, ry	$RZ \leftarrow RX >>> RY[5:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x == z$) and ($y < 16$) and ($z < 16$), then asr16 rz, ry; else asr32 rz, rx, ry;

Description: For asr rz, rx, perform an arithmetic right shift on RZ value (the original value shifts right and the copy of original sign bit will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 30, RZ value (0 or -1) is decided by the sign bit of the original RZ value;

For asr rz, rx, ry, perform an arithmetic right shift on RX value (the original value shifts right and the copy of original sign bit will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RY (RY[5:0]). If the value of RY[5:0] is greater than 30, RZ value (0 or -1) is decided by the sign bit of RX.

Influence on flag No influence

bit:

Exception: None

16-bit instruction

Operation: $RZ \leftarrow RZ >>> RX[5:0]$

Grammar: asr16 rz, rx

Description: Perform an arithmetic right shift on RZ value (the original value shifts right and the copy of original sign bit will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 30, RZ value (0 or -1) is decided by the sign bit of the original RZ value.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15	14	10	9	6	5	2	1	0
0	1 1 1 0 0	RZ	RX	1 0				

32-bit

instruction

Operation: $RZ \leftarrow RX >>> RY[5:0]$

Grammar: asr32 rz, rx, ry

Description: Perform an arithmetic right shift on RX value (the original value shifts right and the copy of original sign bit will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RY (RY[5:0]). If the value of RY[5:0] is greater than 30, RZ value (0 or -1) is decided by the sign bit of RX.

Influence on No influence

flag bit:

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 0 1 0 0	RZ						

CSKY 中天微電

ASRC – Arithmetic shift right immediate to C

Unified instruction

Grammar	Operation	Compiling result
asrc rz, rx, oimm5	$RZ \leftarrow RX \ggg OIMM5,$ $C \leftarrow RX[OIMM5 - 1]$	Only 32-bit instructions exist. asrc32 rz, rx, oimm5

Description: Perform an arithmetic right shift on RX value (the original value shifts right and the copy of original sign bit will shift to the left side), save the end bit shifting out in C, and save the shifting result in RZ; the range of right shift is decided by the value of 5-bit immediate operand with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, then the condition bit C is the sign bit (the highest bit) of RX and RZ value (0 or -1) is decided by the sign bit of RX.

Influence on flag $C \leftarrow RX[OIMM5 - 1]$

bit:

Restriction: The range of immediate operand is 1-32.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow RX \ggg OIMM5, C \leftarrow RX[OIMM5 - 1]$

Grammar: asrc32 rz, rx, oimm5

Description: Perform an arithmetic right shift on RX value (the original value shifts right and the copy of original sign bit will shift to the left side), save the end bit shifting out in C, and save the shifting result in RZ; the range of right shift is decided by the value of 5-bit immediate operand with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, then the condition bit C is the sign bit (the highest bit) of RX and RZ value (0 or -1) is decided by the sign bit of RX. Attention: The binary operand IMM5 is equal to OIMM5 – 1.

Influence on flag $C \leftarrow RX[OIMM5 - 1]$

bit:

Restriction: The range of immediate operand is 1-32.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 IMM5 RX 0 1 0 0 1 1 0 0 1 0 0 RZ						

IMM5 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM5, the shifting value OIMM5 requires offset 1.

00000 – shift by 1 bit

00001 – shift by 2 bits

.....

11111 – shift by 32 bits

ASRI – Arithmetic shift right immediate

Unified instruction

Grammar	Operation	Compiling result
asri rz, rx, imm5	$RZ \leftarrow RX \ggg IMM5$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 8$) and ($z < 8$), then asri16 rz, rx, imm5; else asri32 rz, rx, imm5;

Description:	For asri rz, rx, imm5, perform an arithmetic right shift on RX value (the original value shifts right and the copy of original sign bit will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value is equal to RX.
Influence on flag bit:	No influence
Exception:	None

16-bit instruction

Operation:	$RZ \leftarrow RX \ggg IMM5$
Grammar:	asri16 rz, rx, imm5
Description:	Perform an arithmetic right shift on RX value (the original value shifts right and the copy of original sign bit will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value remains unchanged.
Influence on flag bit:	No influence
Restriction:	The range of register is r0-r7; the range of immediate operand is 0-31.
Exception:	None
Instruction format:	

15 14	11 10	8 7	5 4	0
0 1 0 0 1 0 RX RZ IMM5				

32-bit instruction

Operation: $RZ \leftarrow RX \ggg IMM5$

Grammar: asri32 rz, rx, imm5

Description: Perform an arithmetic right shift on RX value (the original value shifts right and the copy of original sign bit will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value is equal to RX.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 IMM5 RX 0 1 0 0 1 0 0 0 1 0 0 RZ						

BCLRI – Bit clear immediate

**Unified
instruction**

Grammar	Operation	Compiling result
bclri rz, imm5	RZ \leftarrow RZ[IMM5] clear	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 8$), then bclri16 rz, imm5; else bclri32 rz, rz, imm5;
bclri rz, rx, imm5	RZ \leftarrow RX[IMM5] clear	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x == z$) and ($z < 8$), then bclri16 rz, imm5; else bclri32 rz, rx, imm5;

- Description:** Clear the bits indicated by the value of IMM5 field in RZ/RX value, keep other bits unchanged, and save the result after clearing in RZ.
- Influence on flag bit:** No influence
- Restriction:** The range of immediate operand is 0-31.

16-bit instruction

- Operation:** RZ \leftarrow RZ[IMM5] clear
- Grammar:** bclri16 rz, imm5
- Description:** Clear the bits indicated by the value of IMM5 field in RZ value, keep other bits unchanged, and save the result after clearing in RZ.
- Influence on flag bit:** No influence
- Restriction:** The range of register is r0-r7;
The range of immediate operand is 0-31.

Exception: None

Instruction

format:

15 14	10	8 7	5 4	0
0 0 1 1 1	RZ	1 0 0	IMM5	

32-bit instruction

Operation: $RZ \leftarrow RX[IMM5]$ clear

Grammar: bclri32 rz, rx, imm5

Description: Clear the bits indicated by the value of IMM5 field in RX value, keep other bits unchanged, and save the result after clearing in RZ.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	IMM5	RX	0 0 1 0 1 0	0 0 0 0 1	RZ	

BEZ –Branch instruction when register is equal to zero

Unified instruction

Grammar	Operation	Compiling result
bez rx, label	<p>When the register is equal to zero, the program will shift</p> <pre>if (RX == 0) PC ← PC + sign_extend(offset << 1) else PC ← PC + 4</pre>	<p>Only 32-bit instructions exist.</p> <p>bez32 rx, label</p>

Description: If the register RX is equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BEZ instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

32-bit instruction

Operation: When the register is equal to zero, the program will shift.

```
if(RX == 0)
    PC ← PC + sign_extend(offset << 1)
else
    PC ← PC + 4
```

Grammar: bez32 rx, label

Description: If the register RX is equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range

of BEZ instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 1 0 0 0	RX		Offset

BF – C=0 branch instruction

Unified instruction

Grammar	Operation	Compiling result
bf label	<p>When C is equal to zero, the program will shift.</p> <pre>if(C==0) PC←PC + sign_extend(offset << 1); else PC ← next PC;</pre>	<p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of jump.</p> <pre>if (offset<1KB), then bf16 label; else bf32 label;</pre>

Description: If the condition flag bit C is equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction.
Label is gained by adding the current PC to the value of sign-extending the relative offset shifting left by 1 bit to 32 bits. The shifting range of BF instruction is the address space of ±64KB.

Influence on flag bit: No influence

Exception: None

16-bit instruction

Operation: When C is equal to zero, the program will shift.

```
if(C==0)
    PC ← PC + sign_extend(offset << 1)
else
    PC ← PC + 2
```

Grammar:

Description: If the condition flag bit C is equal to zero, the program will shift to label

position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 2$.

Label is gained by adding the current PC to the value of sign-extending the 10-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BF16 instruction is the address space of $\pm 1KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

15 14	10 9	0
0	0 0 0 1 1	Offset

32-bit

instruction

Operation: When C is equal to zero, the program will shift.

`if(C == 0)`

`PC ← PC + sign_extend(offset << 1)`

`else`

`PC ← PC + 4`

Grammar: bf32 label

Description: If the condition flag bit C is equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BF16 instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
-------	-------	-------	-------	---

1	1 1 0 1 0	0 0 0 1 0	0 0 0 0 0	Offset
---	-----------	-----------	-----------	--------



BGENI – Bit generation immediate#

Unified instruction

Grammar	Operation	Compiling result
bgeni rz, imm5	$RZ \leftarrow (2)^{IMM5}$	Only 32-bit instructions exist. bgeni32 rz, imm5

Description: Set the bit of RZ decided by the 5-bit immediate operand ($RZ[IMM5]$) and clear other bits of RZ.

Attention: If IMM5 is smaller than 16, this instruction is the pseudo instruction of movi rz, $(2)^{IMM5}$; if IMM5 is greater than 16, this instruction is the pseudo instruction of movih rz, $(2)^{IMM5}$.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0-31.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow (2)^{IMM5};$

Grammar: bgeni32 rz, imm5

Description: Set the bit of RZ decided by the 5-bit immediate operand ($RZ[IMM5]$) and clear other bits of RZ.

Attention: If IMM5 is smaller than 16, this instruction is the pseudo instruction of movi32 rz, $(2)^{IMM5}$; if IMM5 is greater than 16, this instruction is the pseudo instruction of movih32 rz, $(2)^{IMM5}$.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction format:

If IMM5 is smaller than 16:

3130	2625	2120	1615	0
1 1 1 0 1 0	1 0 0 0 0	RZ		(2) ^{IMM5}

If IMM5 is greater than 16:

3130	2625	2120	1615	0
1 1 1 0 1 0	1 0 0 0 1	RZ		(2) ^{IMM5}

BGENR – Register bit generation

**Unified
instruction**

Grammar	Operation	Compiling result
bgenr rz, rx	If (RX[5] == 0), then RZ $\leftarrow 2^{RX[4:0]}$; else RZ $\leftarrow 0$;	Only 32-bit instructions exist. bgenr32 rz, rx

Description: If RX[5] is 0, set the register bit of RZ decided by five low bits of RX (RX[4:0]) and clear other bits of RZ; otherwise, clear RZ.

Influence on flag bit:

Exception: None

**32-bit
instruction**

Operation: If (RX[5] == 0) , then
 RZ $\leftarrow 2^{RX[4:0]}$;
else
 RZ $\leftarrow 0$;

Grammar: bgenr32 rz, rx

Description: If RX[5] is 0, set the register bit of RZ decided by five low bits of RX (RX[4:0]) and clear other bits of RZ; otherwise, clear RZ.

Influence on flag bit:

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 0 0 0 0 0 RX 0 1 0 1 0 0 0 0 0 1 0 RZ						

BHSZ – Branch instruction when register is greater than or equal to zero

Unified instruction

Grammar	Operation	Compiling result
bhsz rx, label	<p>When the register is greater than or equal to zero, the program will shift if($RX \geq 0$)</p> <pre>PC ← PC + sign_extend(offset << 1); else PC ← PC + 4;</pre>	<p>Only 32-bit instructions exist.</p> <p>bhsz32 rx, label</p>

Description: If the register is greater than or equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.
Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BHSZ instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

32-bit instruction

Operation: When the register is greater than or equal to zero, the program will shift if($RX \geq 0$)

```
PC ← PC + sign_extend(offset << 1)
```

else

```
PC ← PC + 4
```

Grammar: bhsz32 rx, label

Description: If the register RX is greater than or equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BHSZ instruction is the address space of ±64KB.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 1 1 0 1	RX		Offset

BHZ – Branch instruction when register is greater than zero

Unified instruction

Grammar	Operation	Compiling result
bhz rx, label	<p>When the register is greater than zero, the program will shift</p> <p>if($RX > 0$)</p> $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ <p>else</p> $PC \leftarrow PC + 4$	<p>Only 32-bit instructions exist.</p> <p>bhz32 rx, label</p>

Description: If the register RX is greater than zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BHZ instruction is the address space of $\pm 64KB$.

Influence on flag bit: No influence

Exception:

None

32-bit instruction

Operation: When the register is greater than zero, the program will shift

if($RX > 0$)

$$PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$$

else

$$PC \leftarrow PC + 4$$

Grammar: bhz32 rx, label

Description: If the register RX is greater than zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range

of BHZ instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 1 0 1 0	RX		Offset

BKPT – Breakpoint instruction

Unified instruction

Grammar	Operation	Compiling result
bkpt	Trigger a breakpoint exception or enter debugging mode	Always compiled into 16-bit instructions. bkpt16

Description: Breakpoint instruction

Influence on flag bit: No influence

bit:

Exception: Breakpoint exception

16-bit

instruction

Operation: Trigger a breakpoint exception or enter debugging mode

Grammar: bkpt16

Description: Breakpoint instruction

Influence on flag bit: No influence

bit:

Exception: Breakpoint exception

Instruction

format:

15	14	10	9	0
0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		

BLSZ – Branch instruction when register is smaller than or equal to zero

Unified instruction

Grammar	Operation	Compiling result
blsz rx, label	<p>When the register is smaller than or equal to zero, the program will shift if($RX \leq 0$)</p> $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ <p>else</p> $PC \leftarrow PC + 4$	<p>Only 32-bit instructions exist.</p> <p>blsz32 rx, label</p>

Description: If the register RX is smaller than or equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.
 Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BLSZ instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

32-bit instruction

Operation: When the register is smaller than or equal to zero, the program will shift if($RX \leq 0$)

$$PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$$

else

$$PC \leftarrow PC + 4$$

Grammar:

Description:

If the register RX is smaller than or equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BLSZ instruction is the address space of ±64KB.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 1 0 1 1	RX		Offset

BLZ – Branch instruction when register is smaller than zero

Unified instruction

Grammar	Operation	Compiling result
blz rx, label	<p>When the register is smaller than zero, the program will shift</p> <p>if($RX < 0$)</p> $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ <p>else</p> $PC \leftarrow PC + 4$	<p>Only 32-bit instructions exist.</p> <p>blz32 rx, label</p>

Description: If the register RX is smaller than zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BLZ instruction is the address space of $\pm 64KB$.

Influence on flag bit: No influence

Exception: None

32-bit instruction

Operation: When the register is smaller than zero, the program will shift

if($RX < 0$)

$$PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$$

else

$$PC \leftarrow PC + 4$$

Grammar: blz32 rx, label

Description: If the register RX is smaller than zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range

of BLZ instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 1 1 0 0	RX		Offset

BMASKI – Bit mask generation immediate

Unified instruction

Grammar	Operation	Compiling result
bmaski rz, oimm5	$RZ \leftarrow (2)^{OIMM5} - 1$	Only 32-bit instructions exist. bmaski32 rz, oimm5

Description: Generate the immediate operand whose consecutive low bit is 1 and high bit is 0, and save this immediate operand in RZ. Assign the bit of consecutive low bit set as 1 for the immediate operand OIMM5 ($RX[OIMM5-1:0]$), and clear other bits. When OIMM5 is 0 or 32, all bits of RX are set as 1.

Attention: When OIMM5 is 1-16, movi instruction will be executed.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0 and 17-32.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow (2)^{OIMM5} - 1$

Grammar: bmaski32 rz, oimm5

Description: Generate the immediate operand whose consecutive low bit is 1 and high bit is 0, and save this immediate operand in RZ. Assign the bit of consecutive low bit set as 1 for the immediate operand OIMM5 ($RX[OIMM5-1:0]$), and clear other bits. When OIMM5 is 0 or 32, all bits of RX are set as 1.

Attention: When OIMM5 is 1-16, movi instruction will be executed; the binary operand IMM5 is equal to OIMM5 – 1.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0 and 17-32.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	IMM5	0 0 0 0 0	0 1 0 1 0 0	0 0 0 0 1	RZ	

IMM5 field – Assign the highest bit of consecutive low bit set as 1.

Attention: Compared with the binary operand IMM5, the immediate operand OIMM5 requires offset 1.

10000 – set 0-16 bits

10001 – set 0-17 bits

.....

11111 – set 0-31 bits

BMCLR – Clear BCTM bit

Unified instruction

Grammar	Operation	Compiling result
bmclr	Clear BM bit of status register. PSR(BM) ← 0	Only 32-bit instructions exist. bmclr32

Description: Clear BM bit of PSR.

Influence on No influence

flag bit:

Exception: None

32-bit

instruction

Operation: Clear BM bit of status register.
PSR(BM) ← 0

Grammar: bmclr32

Description: Clear the BM bit of PSR.

Influence on No influence

flag bit:

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	0	0	0	0	0	1	0	1

BMSET – Set BCTM bit

Unified instruction

Grammar	Operation	Compiling result
bmset	Set BM bit of status register. PSR(BM) ← 1	Only 32-bit instructions exist. bmset32

Description: Set BM bit of PSR.

Influence on No influence

flag bit:

Exception: None

32-bit

instruction

Operation: Set BM bit of status register.

PSR(BM) ← 1

Grammar: bmset32

Description: Set BM bit of PSR.

Influence on No influence

flag bit:

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	0	0	0	0	0	0	0	0

BNEZ – Branch instruction when register is not equal to zero

Unified instruction

Grammar	Operation	Compiling result
bnez rx, label	<p>When the register is not equal to zero, the program will shift</p> <p>if($RX \neq 0$)</p> $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$ <p>else</p> $PC \leftarrow PC + 4$	<p>Only 32-bit instructions exist.</p> <p>bnez32 rx, label</p>

Description: If the register RX is not equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BNEZ instruction is the address space of $\pm 64KB$.

Influence on flag bit: No influence

Exception: None

32-bit instruction

Operation: When the register is not equal to zero, the program will shift

if($RX \neq 0$)

$$PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$$

else

$$PC \leftarrow PC + 4$$

Grammar: bnez32 rx, label

Description: If the register RX is not equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range

of BNEZ instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30 26 25 21 20 16 15 0

1	1 1 0 1 0	0 1 0 0 1	RX	Offset
---	-----------	-----------	----	--------

BPOP.H – Binary push of translated half-word

Unified instruction

Grammar	Operation	Compiling result
bpop.h rz	<p>Update the binary translated stack register to the top of binary translated stack storage, and load half-word from binary translated stack storage to register RZ;</p> <p>if (BSP - 2 < FP')</p> <p> R15 ← next PC</p> <p> PC ← SVBR - 12</p> <p>else</p> <p> BSP ← BSP - 2;</p> <p> RZ ← zero_extend(MEM[BSP]);</p>	<p>Only 16-bit instructions exist.</p> <p>bpop.h rz;</p>

Description: Compare the value after subtracting 2 from the binary translated stack pointer register (BSP) with the binary translated frame pointer register (FP'). If the value after subtracting 2 from BSP is smaller than FP', the return address of subprogram (PC of the next instruction) is saved in link register R15 and the program will shift to SVBR-12 position before execution. Otherwise, update BSP to the top of binary translated stack storage, and load half-word in binary translated stack storage to register RZ after zero-extending it to 32 bits. Adopt direct addressing mode of stack register.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Update the binary translated stack register to the top of binary translated stack storage, and load half-word from binary translated stack storage to

```

register RZ;
if (BSP - 2 < FP')
    R15 ← next PC
    PC ← SVBR - 12
else
    BSP ← BSP - 2;
    RZ ← zero_extend(MEM[BSP]);

```

Grammar:

bpop16.h rz

Description:

Compare the value after subtracting 2 from the binary translated stack pointer register (BSP) with the binary translated frame pointer register (FP'). If the value after subtracting 2 from BSP is smaller than FP', the return address of subprogram (PC of the next instruction) is saved in link register R15 and the program will shift to SVBR-12 position before execution. Otherwise, update BSP to the top of binary translated stack storage, and load half-word in binary translated stack storage to register RZ after zero-extending it to 32 bits. Adopt direct addressing mode of stack register.

Influence on flag No influence

bit:

Restriction: The range of register is r0 – r7.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

15	14	10	9	8	7	6	5	4	2	1	0
0	0	0	1	0	1	0	1	RZ	0	0	

BPOP.W – Binary push of translated word

Unified instruction

Grammar	Operation	Compiling result
bpop.w rz	<p>Update the binary translated stack register to the top of binary translated stack storage, and load word from binary translated stack storage to register RZ;</p> <p>if ($BSP - 4 < FP'$)</p> <p style="padding-left: 40px;">R15 \leftarrow next PC</p> <p style="padding-left: 40px;">PC \leftarrow SVBR - 12</p> <p>else</p> <p style="padding-left: 40px;">$BSP \leftarrow BSP - 4;$</p> <p style="padding-left: 40px;">$RZ \leftarrow MEM[BSP];$</p>	<p>Only 16-bit instructions exist.</p> <p style="text-align: center;">$bpop.w\ rz;$</p>

Description: Compare the value after subtracting 4 from the binary translated stack pointer register (BSP) with the binary translated frame pointer register (FP'). If the value after subtracting 4 from BSP is smaller than FP', the return address of subprogram (PC of the next instruction) is saved in link register R15 and the program will shift to SVBR-12 position before execution. Otherwise, update BSP to the top of binary translated stack storage, and load word in binary translated stack storage to register RZ. Adopt direct addressing mode of stack register.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Update the binary translated stack register to the top of binary translated stack storage, and load word from binary translated stack storage to register RZ;

if ($BSP - 4 < FP'$)

R15 ← next PC

PC ← SVBR - 12

else

BSP ← BSP - 4;

RZ ← MEM[BSP];

Grammar:

bpop16.w rz

Description:

Compare the value after subtracting 4 from the binary translated stack pointer register (BSP) with the binary translated frame pointer register (FP'). If the value after subtracting 4 from BSP is smaller than FP', the return address of subprogram (PC of the next instruction) is saved in link register R15 and the program will shift to SVBR-12 position before execution. Otherwise, update BSP to the top of binary translated stack storage, and load word in binary translated stack storage to register RZ. Adopt direct addressing mode of stack register.

Influence on flag

No influence

bit:

Restriction:

The range of register is r0 – r7.

Exception:

Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

15	14	10	9	8	7	6	5	4	2	1	0
0	0	0	1	0	1	0	1	RZ	1	0	

BPUSH.H – Binary push of translated half-word

Unified instruction

Grammar	Operation	Compiling result
bpush.h rz	<p>Save half-word in register RZ into the binary translated stack storage, and update the binary translated stack register to the top of binary translated stack storage;</p> <pre>if (BSP + 2 > TOP) R15 ← next PC PC ← SVBR - 12 else MEM[BSP] ← RZ[15:0]; BSP ← BSP + 2;</pre>	<p>Only 16-bit instructions exist.</p> <pre>bpush.h rz;</pre>

Description: Compare the value after adding 2 to the binary translated stack pointer register (BSP) with the binary translated stack top register (TOP). If the value after adding 2 to BSP is greater than TOP, the return address of subprogram (PC of the next instruction) is saved in link register R15 and the program will shift to SVBR-12 position before execution. Otherwise, save half-word in register RZ into the binary translated stack storage, and update BSP to the top of binary translated stack storage. Adopt direct addressing mode of stack register.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Save half-word in register RZ into the binary translated stack storage, and update the binary translated stack register to the top of binary translated stack storage;

```
if (BSP + 2 > TOP)
```

$R15 \leftarrow \text{next PC}$
 $PC \leftarrow \text{SVBR} - 12$

else

```

MEM[BSP] ← RZ[15:0];
BSP ← BSP + 2;

```

Grammar:

bpush16.h rz

Description:

Compare the value after adding 2 to the binary translated stack pointer register (BSP) with the binary translated stack top register (TOP). If the value after adding 2 to BSP is greater than TOP, the return address of subprogram (PC of the next instruction) is saved in link register R15 and the program will shift to SVBR-12 position before execution. Otherwise, save half-word in register RZ into the binary translated stack storage, and update BSP to the top of binary translated stack storage. Adopt direct addressing mode of stack register.

Influence on flag

No influence

bit:

Restriction:

The range of register is r0 – r7.

Exception:

Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

15	14	10	9	8	7	6	5	4	2	1	0
0	0	0	1	0	1	0	0	1	1	1	RZ

BPUSH.W – Binary push of translated word

Unified instruction

Grammar	Operation	Compiling result
bpush.w rz	<p>Save word in register RZ into the binary translated stack storage, and update the binary translated stack register to the top of binary translated stack storage;</p> <p>if ($BSP + 4 > TOP$)</p> <p style="padding-left: 20px;">R15 \leftarrow next PC</p> <p style="padding-left: 20px;">PC \leftarrow SVBR - 12</p> <p>else</p> <p style="padding-left: 20px;">MEM[BSP] \leftarrow RZ[31:0];</p> <p style="padding-left: 20px;">BSP \leftarrow BSP + 4;</p>	<p>Only 16-bit instructions exist.</p> <p style="text-align: center;"><code>bpush.w rz;</code></p>

Description: Compare the value after adding 4 to the binary translated stack pointer register (BSP) with the binary translated stack top register (TOP). If the value after adding 4 to BSP is greater than TOP, the return address of subprogram (PC of the next instruction) is saved in link register R15 and the program will shift to SVBR-12 position before execution. Otherwise, save word in register RZ into the binary translated stack storage, and update BSP to the top of binary translated stack storage. Adopt direct addressing mode of stack register.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Save word in register RZ into the binary translated stack storage, and update the binary translated stack register to the top of binary translated stack storage;

if ($BSP + 4 > TOP$)

```

R15 ← next PC
PC ← SVBR - 12
else
    MEM[BSP] ← RZ[31:0];
    BSP ← BSP + 4;

```

Grammar:

bpush16.w rz

Description:

Compare the value after adding 4 to the binary translated stack pointer register (BSP) with the binary translated stack top register (TOP). If the value after adding 4 to BSP is greater than TOP, the return address of subprogram (PC of the next instruction) is saved in link register R15 and the program will shift to SVBR-12 position before execution. Otherwise, save word in register RZ into the binary translated stack storage, and update BSP to the top of binary translated stack storage. Adopt direct addressing mode of stack register.

Influence on flag

No influence

bit:

Restriction:

The range of register is r0 – r7.

Exception:

Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

15	14	10	9	8	7	6	5	4	2	1	0
0	0	0	1	0	1	0	0	1	1	1	RZ

BR – Unconditional jump

**Unified
instruction**

Grammar	Operation	Compiling result
br label	$PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of jump. if($\text{offset} < 1KB$), then br16 label; else br32 label;

Description: The program unconditionally jumps to label for execution.
Label is gained by adding the current PC to the value of sign-extending the relative offset shifting left by 1 bit to 32 bits.

Influence on flag bit: No influence

Exception: None

**16-bit
instruction**

Operation: $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$

Grammar: br16 label

Description: The program unconditionally jumps to label for execution.

Label is gained by adding the current PC to the value of sign-extending the 10-bit relative offset shifting left by 1 bit to 32 bits. The jump range of BR16 instruction is the address space of $\pm 1KB$.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

15 14	10 9	0
-------	------	---

0	0 0 0 0 1	Offset
---	-----------	--------

32-bit

instruction

Operation: $PC \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1)$

Grammar: br32 label

Description: The program unconditionally jumps to label for execution.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The jump range of BR instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 0 0 0 0	0 0 0 0 0		Offset

BREV – Bit-reverse

**Unified
instruction**

Grammar	Operation	Compiling result
brev rz, rx	for i=0 to 31 RZ[i] ← RX[31-i];	Only 32-bit instructions exist. brev32 rz, rx

Description: Perform a bitwise reverse operation on RX value and save the result in RZ.

If RX value is “abcdefgijklmnopqrstuvwxyz012345”, RZ value changes into “543210zyxwvutsrqponmlkjihgfedcba” after bitwise reverse operation.

Influence on flag bit: No influence

Exception: None

32-bit

instruction

Operation: for i=0 to 31
RZ[i] ← RX[31-i];

Grammar: brev32 rz, rx

Description: Perform a bitwise reverse operation on RX value and save the result in RZ.

If RX value is “abcdefgijklmnopqrstuvwxyz012345”, RZ value changes into “543210zyxwvutsrqponmlkjihgfedcba” after bitwise reverse operation.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 0 0 0	1 0 0 0 0	RZ
---	-----------	-----------	----	-------------	-----------	----



BSETI – Bit set immediate

**Unified
instruction**

Grammar	Operation	Compiling result
bseti rz, imm5	$RZ \leftarrow RZ[IMM5] \text{ set}$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 8$), then bseti16 rz, imm5; else bseti32 rz, rz, imm5;
bseti rz, rx, imm5	$RZ \leftarrow RX[IMM5] \text{ set}$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if(($x == z$) and ($z < 8$), then bseti16 rz, imm5; else bseti32 rz, rx, imm5;

Description: Set the bit indicated by the value of IMM5 field as 1 in RZ/RX value, keep other bits unchanged, and save the result in RZ.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

16-bit instruction

Operation: $RZ \leftarrow RZ[IMM5] \text{ set}$

Grammar: bseti16 rz, imm5

Description: Set the bit indicated by the value of IMM5 field as 1 in RZ value, keep other bits unchanged, and save the result in RZ.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7; the range of immediate operand is 0-31.

Exception: None

Instruction

format:

15 14	10	8 7	5 4	0
0 0 1 1 1 RZ 1 0 1 IMM5				

32-bit instruction

Operation: $RZ \leftarrow RX[IMM5]$ set

Grammar: bseti32 rz, rx, imm5

Description: Set the bit indicated by the value of IMM5 field as 1 in RX value, keep other bits unchanged, and save the result in RZ.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 IMM5 RX 0 0 1 0 1 0 0 0 0 1 0 RZ						

BSR – Jump to subprogram

**Unified
instruction**

Grammar	Operation	Compiling result
bsr label	Link and jump to the subprogram: R15 \leftarrow next PC PC \leftarrow PC + sign_extend(offset \ll 1)	Only 32-bit instructions exist. bsr32 label;

Description: The subprogram jumps, the return address of subprogram (PC of the next instruction) is saved in link register R15, and the program will shift to label position before execution.
Label is gained by adding the current PC to the value of sign-extending the relative offset shifting left by 1 bit to 32 bits.

Influence on flag No influence

bit:

Exception: None

32-bit

instruction

Operation: Link and jump to the subprogram:
R15 \leftarrow PC+4
PC \leftarrow PC + sign_extend(offset \ll 1)

Grammar: bsr32 label

Description: The subprogram jumps, the return address of subprogram (PC of the next instruction, i.e. PC+4 at present) is saved in link register R15, and the program will shift to label position before execution.
Label is gained by adding the current PC to the value of sign-extending the 26-bit relative offset shifting left by 1 bit to 32 bits. The jump range of BSR instruction is the address space of $\pm 64KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

3130

2625

0

1	1 1 0 0 0	Offset
---	-----------	--------



BT – C=1 branch instruction

Unified instruction

Grammar	Operation	Compiling result
bt label	<pre>if(C == 1) PC ← PC + sign_extend(offset << 1); else PC ← next PC;</pre>	<p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of jump.</p> <p>if (offset<1KB), then</p> <p> bt16 label;</p> <p>else</p> <p> bt32 label;</p>

Description: If the condition flag bit C is equal to 1, the program will shift to label position before execution; otherwise the program will execute the next instruction.

Label is gained by adding the current PC to the value of sign-extending the relative offset shifting left by 1 bit to 32 bits. The shifting range of BT instruction is the address space of ±64KB.

Influence on flag bit: No influence

Exception: None

16-bit instruction

Operation: When C is equal to 1, the program will shift

```
if(C == 1)
    PC ← PC + sign_extend(offset << 1)
else
    PC ← PC + 2
```

Grammar: bt16 label

Description: If the condition flag bit C is equal to 1, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $PC \leftarrow PC + 2$.

Label is gained by adding the current PC to the value of sign-extending

the 10-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BT16 instruction is the address space of $\pm 1\text{KB}$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

15 14	10 9	0
0	0 0 0 1 0	Offset

**32-bit
instruction**

Operation: When C is equal to 1, the program will shift
if($C == 1$)

$\text{PC} \leftarrow \text{PC} + \text{sign_extend}(\text{offset} \ll 1)$

else

$\text{PC} \leftarrow \text{PC} + 4$

Grammar: bt32 label

Description: If the condition flag bit C is equal to 1, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. $\text{PC} \leftarrow \text{PC} + 4$.

Label is gained by adding the current PC to the value of sign-extending the 16-bit relative offset shifting left by 1 bit to 32 bits. The shifting range of BT instruction is the address space of $\pm 64\text{KB}$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1	1 1 0 1 0	0 0 0 1 1	0 0 0 0 0	Offset

BTSTI – Bit test immediate

Unified instruction

Grammar	Operation	Compiling result
btsti rx, imm5	$C \leftarrow RX[IMM5]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 8$), then btsti16 rx, imm5; else btsti32 rx, imm5;

Description: Test the bit of RX decided by IMM5 ($RX[IMM5]$), and make the value of condition bit C equal to value of this bit.

Influence on flag $C \leftarrow RX[IMM5]$

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

16-bit instruction

Operation: $C \leftarrow RX[IMM5]$

Grammar: btsti16 rx, imm5

Description: Test the bit of RX decided by IMM5 ($RX[IMM5]$), and make the value of condition bit C equal to value of this bit.

Influence on flag No influence

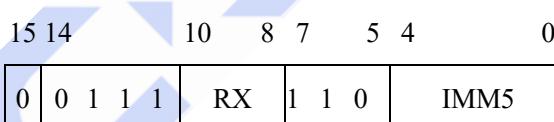
bit:

Restriction: The range of register is r0-r7; the range of immediate operand is 0-31.

Exception: None

Instruction

format:



32-bit

instruction

Operation: $C \leftarrow RX[IMM5]$

Grammar: btsti32 rx, imm5

Description: Test the bit of RX decided by IMM5 ($RX[IMM5]$), and make the value of condition bit C equal to value of this bit.

Influence on flag $C \leftarrow RX[IMM5]$

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 IMM5 RX 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0						

CLRF – C=0 clear

Unified instruction

Grammar	Operation	Compiling result
clrf rz	if C==0, then RZ ← 0; else RZ ← RZ;	Only 32-bit instructions exist. clrf32 rz

Description: If C is equal to zero, clear the register RZ; otherwise, keep the register RZ unchanged.

Influence on flag bit: No influence

Exception: None

32-bit

instruction

Operation: if C==0, then
 RZ ← 0;
else
 RZ ← RZ;

Grammar: clrf32 rz

Description: If C is equal to zero, clear the register RZ; otherwise, keep the register RZ unchanged.

Influence on flag bit: No influence

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RZ	0	0	0	0	1	0

CLRT – C=1 clear

Unified instruction

Grammar	Operation	Compiling result
clrt rz	if C==1, then RZ ← 0; else RZ ← RZ;	Only 32-bit instructions exist. clrt32 rz

Description: If C is equal to 1, clear the register RZ; otherwise, keep the register RZ unchanged.

Influence on flag bit: No influence

Exception: None

32-bit

instruction

Operation: if C==1, then
 RZ ← 0;
else
 RZ ← RZ;

Grammar: clrt32 rz

Description: If C is equal to 1, clear the register RZ; otherwise, keep the register RZ unchanged.

Influence on flag bit: No influence

Exception: None

Instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RZ	0 0 0 0 0	0 0 1 0 1 1	0 0 0 1 0	0 0 0 0 0	

CMPHS – Compare unsigned when greater or equal

Unified instruction

Grammar	Operation	Compiling result
cmphs rx, ry	<p>Make an unsigned comparison between RX and RY.</p> <p>If RX \geq RY, then</p> <pre> C ← 1; else C ← 0;</pre>	<p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of register.</p> <p>if (x<16) and (y<16), then</p> <pre>cmphs16 rx, ry;</pre> <p>else</p> <pre>cmphs32 rx, ry;</pre>

- Description:** Subtract RY value from RX value, compare the result with 0, and update C bit. Make an unsigned comparison via cmphs; in another word, operand is considered as unsigned number. If RX is greater than or equal to RY, it means that the subtraction result is greater than or equal to zero. Set the condition bit C; otherwise, clear the condition bit C.
- Influence on flag bit:** Set the condition bit C according to the comparison result
- Exception:** None

16-bit instruction

- Operation:** Make an unsigned comparison between RX and RY.
- If RX \geq RY, then
- ```

C ← 1;
else
 C ← 0;
```
- Grammar:** cmphs16 rx, ry
- Description:** Subtract RY value from RX value, compare the result with 0, and update C bit. Make an unsigned comparison via cmphs16; in another word, operand is considered as unsigned number. If RX is greater than or equal to RY, it means that the subtraction result is greater than or equal to zero. Set the condition bit C; otherwise, clear the condition bit C.

**Influence on flag** Set the condition bit C according to the comparison result bit:

**Restriction:** The range of register is r0-r15.

**Exception:** None

#### Instruction

##### format:

|                               |      |     |       |
|-------------------------------|------|-----|-------|
| 15 14                         | 10 9 | 6 5 | 2 1 0 |
| 0   1 1 0 0 1   RY   RX   0 0 |      |     |       |

#### 32-bit instruction

**Operation:** Make an unsigned comparison between RX and RY.

If RX  $\geq$  RY, then

C  $\leftarrow$  1;

else

C  $\leftarrow$  0;

**Grammar:** cmphs32 rx, ry

**Description:** Subtract RY value from RX value, compare the result with 0, and update C bit. Make an unsigned comparison via cmphs32; in another word, operand is considered as unsigned number. If RX is greater than or equal to RY, it means that the subtraction result is greater than or equal to zero. Set the condition bit C; otherwise, clear the condition bit C.

**Influence on flag** Set the condition bit C according to the comparison result bit:

**Exception:** None

#### Instruction

##### format:

|                                                               |       |       |       |      |     |   |
|---------------------------------------------------------------|-------|-------|-------|------|-----|---|
| 31 30                                                         | 26 25 | 21 20 | 16 15 | 10 9 | 5 4 | 0 |
| 1   1 0 0 0 1   RY   RX   0 0 0 0 0 1   0 0 0 0 1   0 0 0 0 0 |       |       |       |      |     |   |

## CMPHSI – Compare immediate unsigned when greater or equal

### Unified instruction

| Grammar           | Operation                                                                                                                                        | Compiling result                                                                                                                                                                                                                |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cmphsi rx, oimm16 | <p>Make an unsigned comparison between RX and immediate operand.</p> <p>If RX &gt;= zero_exten(d(OI MM16),</p> <p>C ← 1;<br/>else<br/>C ← 0;</p> | <p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of immediate operand and register.</p> <pre>if (oimm16&lt;33) and (x&lt;8),then     cmphsi16 rx, oimm5; else     cmphsi32 rx, oimm16;</pre> |

**Description:** Zero-extend the 16-bit immediate operand with offset 1 (OIMM16) to 32 bits, subtract this 32-bit value from RX value, compare the result with 0, and update C bit. Make an unsigned comparison via cmphsi; in another word, operand is considered as unsigned number. If RX is greater than or equal to OIMM16 after zero-extension, it means that the subtraction result is greater than or equal to zero. Set the condition bit C; otherwise, clear the condition bit C.

**Influence on flag bit:** Set the condition bit C according to the comparison result

**Restriction:** The range of immediate operand is 0x1-0x10000.

**Exception:** None

### 16-bit instruction

#### Operation:

Make an unsigned comparison between RX and immediate operand.

If RX >= zero\_extend(OIMM5), then

C ← 1;

else

$C \leftarrow 0;$

**Grammar:** cmphsi16 rx, oimm5

**Description:** Zero-extend the 5-bit immediate operand with offset 1 (OIMM5) to 32 bits, subtract this 32-bit value from RX value, compare the result with 0, and update C bit. Make an unsigned comparison via cmphsi16; in another word, operand is considered as unsigned number. If RX is greater than or equal to OIMM5 after zero-extension, it means that the subtraction result is greater than or equal to zero. Set the condition bit C; otherwise, clear the condition bit C.

Attention: The binary operand IMM5 is equal to OIMM5 – 1.

**Influence on flag bit:** Set the condition bit C according to the comparison result

**Restriction:** The range of register is r0-r7; the range of immediate operand is 1-32.

**Exception:** None

**Instruction**

**format:**

|    |    |    |   |   |    |   |   |
|----|----|----|---|---|----|---|---|
| 15 | 14 | 10 | 8 | 7 | 5  | 4 | 0 |
| 0  | 0  | 1  | 1 | 1 | RX | 0 | 0 |

IMM5

IMM5 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM5, the immediate operand OIMM5 participating in comparison requires offset 1.

00000 – make a comparison with 1

00001 – make a comparison with 2

.....

11111 – make a comparison with 32

### 32-bit instruction

**Operation:** Make an unsigned comparison between RX and immediate operand.

If  $RX \geq \text{zero\_extend(OIMM16)}$ , then

$C \leftarrow 1;$

else

$C \leftarrow 0;$

**Grammar:** cmphsi32 rx, oimm16

**Description:** Zero-extend the 16-bit immediate operand with offset 1 (OIMM16) to 32 bits, subtract this 32-bit value from RX value, compare the result with 0, and update C bit. Make an unsigned comparison via cmphsi32; in another word, operand is considered as unsigned number. If RX is greater than or equal to OIMM16 after zero-extension, it means that the subtraction result is greater than or equal to zero. Set the condition bit C; otherwise, clear the condition bit C.

Attention: The binary operand IMM16 is equal to OIMM16 – 1.

**Influence on flag bit:** Set the condition bit C according to the comparison result

**Restriction:** The range of immediate operand is 0x1-0x10000.

**Exception:** None

**Instruction**

**format:**

| 31 30 | 26 25     | 21 20     | 16 15 | 0     |
|-------|-----------|-----------|-------|-------|
| 1     | 1 1 0 1 0 | 1 1 0 0 0 | RX    | IMM16 |

IMM16 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM16, the immediate operand OIMM16 participating in comparison requires offset 1.

0000000000000000 – make a comparison with 0x1

0000000000000001 – make a comparison with 0x2

.....

1111111111111111 – make a comparison with 0x10000

## CMPLT – Compare signed when smaller

### Unified instruction

| Grammar      | Operation                                                                                                                                                                     | Compiling result                                                                                                                                                                                                                                               |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cmplt rx, ry | <p>Make a signed comparison between RX and RY.</p> <p>If RX &lt; RY, then</p> <p style="padding-left: 40px;">C ← 1;</p> <p>else</p> <p style="padding-left: 40px;">C ← 0;</p> | <p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of register.</p> <p>if (x&lt;16) and (y&lt;16), then</p> <p style="padding-left: 40px;">cmplt16 rx, ry;</p> <p>else</p> <p style="padding-left: 40px;">cmplt32 rx, ry;</p> |

**Description:** Subtract RY value from RX value, compare the result with 0, and update C bit. Make a signed comparison via cmplt; in another word, operand is considered as signed number of complement form. If RX is smaller than RY, it means that the subtraction result is smaller than zero. Set the condition bit C; otherwise, clear the condition bit C.

**Influence on flag bit:** Set the condition bit C according to the comparison result

**Exception:** None

### 16-bit instruction

**Operation:** Make a signed comparison between RX and RY.

If RX < RY, then

C ← 1;

else

C ← 0;

**Grammar:** cmplt16 rx, ry

**Description:** Subtract RY value from RX value, compare the result with 0, and update C bit. Make a signed comparison via cmplt16; in another word, operand is considered as signed number of complement form. If RX is smaller than RY, it means that the subtraction result is smaller than zero. Set the condition bit C; otherwise, clear the condition bit C.

**Influence on flag** Set the condition bit C according to the comparison result bit:

**Restriction:** The range of register is r0-r15.

**Exception:** None

#### Instruction

##### format:

|       |           |     |     |     |
|-------|-----------|-----|-----|-----|
| 15 14 | 10 9      | 6 5 | 2 1 | 0   |
| 0     | 1 1 0 0 1 | RY  | RX  | 0 1 |

#### 32-bit

#### instruction

**Operation:** Make a signed comparison between RX and RY.

If RX < RY, then

C ← 1;

else

C ← 0;

**Grammar:** cmplt32 rx, ry

**Description:** Subtract RY value from RX value, compare the result with 0, and update C bit. Make a signed comparison via cmplt32; in another word, operand is considered as signed number of complement form. If RX is smaller than RY, it means that the subtraction result is smaller than zero. Set the condition bit C; otherwise, clear the condition bit C.

**Influence on flag** Set the condition bit C according to the comparison result bit:

**Exception:** None

#### Instruction

##### format:

|       |           |       |       |             |           |           |
|-------|-----------|-------|-------|-------------|-----------|-----------|
| 31 30 | 26 25     | 21 20 | 16 15 | 10 9        | 5 4       | 0         |
| 1     | 1 0 0 0 1 | RY    | RX    | 0 0 0 0 0 1 | 0 0 0 1 0 | 0 0 0 0 0 |

## CMPLTI – Compare immediate signed when smaller

**Unified  
instruction**

| Grammar              | Operation                                                                                                                  | Compiling result                                                                                                                                                                                                          |
|----------------------|----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cmplti rx,<br>oimm16 | Make a signed comparison between RX and immediate operand.<br><br>If RX < zero_extend(OIMM16),<br>C ← 1;<br>else<br>C ← 0; | Compiled into corresponding<br>16-bit or 32-bit instructions<br>according to the range of<br>immediate operand and register.<br><br>if (x<8) and (oimm16<33), then<br>cmplti16 rx, oimm5;<br>else<br>cmplti32 rx, oimm16; |

**Description:** Zero-extend the 16-bit immediate operand with offset 1 (OIMM16) to 32 bits, subtract this 32-bit value from RX value, compare the result with 0, and update C bit. Make a signed comparison via cmplti; in another word, RX value is considered as signed number of complement form. If RX is smaller than OIMM16 after zero-extension, it means that the subtraction result is smaller than zero. Set the condition bit C; otherwise, clear the condition bit C.

**Influence on flag bit:** Set the condition bit C according to the comparison result

**Restriction:** The range of immediate operand is 0x1-0x10000.

**Exception:** None

**16-bit**

**instruction**

**Operation:** Make a signed comparison between RX and immediate operand.

If RX < zero\_extend(OIMM5), then

    C ← 1;

else

    C ← 0;

cmplti16 rx, oimm5

**Grammar:**

**Description:** Zero-extend the 5-bit immediate operand with offset 1 (OIMM5) to 32 bits, subtract this 32-bit value from RX value, compare the result with 0,

and update C bit. Make a signed comparison via cmplti16; in another word, RX value is considered as signed number of complement form. If RX is smaller than OIMM5 after zero-extension, it means that the subtraction result is smaller than zero. Set the condition bit C; otherwise, clear the condition bit C.

Attention: The binary operand IMM5 is equal to OIMM5 – 1.

**Influence on flag bit:** Set the condition bit C according to the comparison result

**Restriction:** The range of register is r0-r7; the range of immediate operand is 1-32.

**Exception:** None

**Instruction**

**format:**

|    |    |    |   |   |    |   |   |
|----|----|----|---|---|----|---|---|
| 15 | 14 | 10 | 8 | 7 | 5  | 4 | 0 |
| 0  | 0  | 1  | 1 | 1 | RX | 0 | 0 |

IMM5

IMM5 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM5, the immediate operand OIMM5 participating in comparison requires offset 1.

00000 – make a comparison with 1

00001 – make a comparison with 2

.....

11111 – make a comparison with 32

### 32-bit instruction

**Operation:** Make a signed comparison between RX and immediate operand.

If RX < zero\_extend(OIMM16), then

C ← 1;

else

C ← 0;

**Grammar:** cmplti32 rx, oimm16

**Description:** Zero-extend the 16-bit immediate operand with offset 1 (OIMM16) to 32 bits, subtract this 32-bit value from RX value, compare the result with 0, and update C bit. Make a signed comparison via cmplti32; in another word, RX value is considered as signed number of complement form. If RX is smaller than OIMM16 after zero-extension, it means that

the subtraction result is smaller than zero. Set the condition bit C; otherwise, clear the condition bit C.

Attention: The binary operand IMM16 is equal to OIMM16 – 1.

**Influence on flag bit:** Set the condition bit C according to the comparison result

**Restriction:** The range of immediate operand is 0x1-0x10000.

**Exception:** None

**Instruction**

**format:**

| 31 30         | 26 25     | 21 20 | 16 15 | 0     |
|---------------|-----------|-------|-------|-------|
| 1   1 1 0 1 0 | 1 1 0 0 1 | RX    |       | IMM16 |

IMM16 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM16, the immediate operand OIMM16 participating in comparison requires offset 1.

0000000000000000 – make a comparison with 0x1

0000000000000001 – make a comparison with 0x2

.....

1111111111111111 – make a comparison with 0x10000

## CMPNE – Compare unequal

**Unified  
instruction**

| Grammar      | Operation                                                                                               | Compiling result                                                                                                                                                                                               |
|--------------|---------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cmpne rx, ry | <p>Make a comparison between RX and RY.</p> <p>If RX != RY, then</p> <pre> C ← 1; else     C ← 0;</pre> | <p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of register.</p> <p>if (x&lt;16) and (y&lt;16), then</p> <pre>cmpne16 rx, ry;</pre> <p>else</p> <pre>cmpne32 rx, ry;</pre> |

- Description:** Subtract RY value from RX value, compare the result with 0, and update C bit. If RX is not equal to RY, it means that the subtraction result is not equal to zero. Set the condition bit C; otherwise, clear the condition bit C.
- Influence on flag bit:** Set the condition bit C according to the comparison result
- Exception:** None

**16-bit  
instruction**

- Operation:** Make a comparison between RX and RY.
- If RX != RY, then
- ```

C ← 1;
else
    C ← 0;
```
- Grammar:** cmpne16 rx, ry
- Description:** Subtract RY value from RX value, compare the result with 0, and update C bit. If RX is not equal to RY, it means that the subtraction result is not equal to zero. Set the condition bit C; otherwise, clear the condition bit C.
- Influence on flag bit:** Set the condition bit C according to the comparison result
- Restriction:** The range of register is r0-r15.
- Exception:** None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 0 1	RY	RX	1 0

32-bit

instruction

Operation: Make a comparison between RX and RY.

If RX != RY, then

C ← 1;

else

C ← 0;

Grammar: cmpne32 rx, ry

Description: Subtract RY value from RX value, compare the result with 0, and update C bit. If RX is not equal to RY, it means that the subtraction result is not equal to zero. Set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: Set the condition bit C according to the comparison result

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 0 0 0 1	0 0 1 0 0	0 0 0 0 0	

CMPNEI – Compare unequal immediate

Unified instruction

Grammar	Operation	Compiling result
cmpnei rx, imm16	<p>Make a comparison between RX and immediate operand.</p> <p>If RX != zero_extend(imm16), $C \leftarrow 1;$ else $C \leftarrow 0;$</p>	<p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of immediate operand and register.</p> <p>if ($x < 7$) and ($imm16 < 33$), then <code>cmpnei16 rx, imm5;</code> else <code>cmpnei32 rx, imm16;</code></p>

Description: Subtract the value of 16-bit immediate operand that is zero-extended to 32 bits from RX value, compare the result with 0, and update C bit. If RX is not equal to IMM16 after zero-extension, it means that the subtraction result is not equal to zero. Set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: Set the condition bit C according to the comparison result

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

16-bit instruction

Operation: Make a comparison between RX and immediate operand.

If RX != zero_extend(IMM5), then

```

 $C \leftarrow 1;$ 
else
 $C \leftarrow 0;$ 

```

Grammar:

Description:

Subtract the value of 5-bit immediate operand that is zero-extended to 32 bits from RX value, compare the result with 0, and update C bit. If RX is not equal to IMM5 after zero-extension, it means that the subtraction result is not equal to zero. Set the condition bit C; otherwise, clear the

condition bit C.

Influence on flag bit: Set the condition bit C according to the comparison result

Restriction: The range of register is r0-r7;

The range of immediate operand is 0-31.

Exception: None

Instruction

format:

15	14	10	8	7	5	4	0
0	0 1 1 1	RX	0 1 0		IMM5		

32-bit instruction

Operation: Make a comparison between RX and immediate operand.

If RX != zero_extend(imm16), then

C ← 1;

else

C ← 0;

Grammar: cmpnei rx, imm16

Description: Subtract the value of 16-bit immediate operand that is zero-extended to 32 bits from RX value, compare the result with 0, and update C bit. If RX is not equal to IMM16 after zero-extension, it means that the subtraction result is not equal to zero. Set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: Set the condition bit C according to the comparison result

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	0
1	1 1 0 1 0	1 1 0 1 0	RX		IMM16			

CPOP – Co-processor operation instruction

Unified instruction

Grammar	Operation	Compiling result
cpop <cpid, func>	Co-processor operation instruction executes one or multiple user defined co-processor operations.	Only 32-bit instructions exist. cpop32 <cpid, func>

- Description:** This instruction executes user defined co-processor operation. The code space of {[20-16], [14-0]} is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define relevant operations (such as transfer of co-processor instruction code).
- Influence on flag bit:** This co-processor operation instruction does not influence the main assembly line flag bit, but might affect the flag bit of co-processor.
- Exception:** Illegal instruction exception

32-bit instruction

- Operation:** Co-processor operation instruction executes one or multiple user defined co-processor operations.
- Grammar:** cpop32 <cpid, func>
- Description:** This instruction executes user defined co-processor operation. The code space of {[20-16], [14-0]} is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define relevant operations (such as transfer of co-processor instruction code).
- Influence on flag bit:** This co-processor operation instruction does not influence the main assembly line flag bit, but might affect the flag bit of co-processor.
- Exception:** Illegal instruction exception
- Instruction format:**

31 30	26 25	21 20	16 15 14	0
1 1 1 1 1 1 0 CPID User-define_1 1 User-define_0				



CPRC – Read transfer from condition bit of co-processor

Unified instruction

Grammar	Operation	Compiling result
cprc <cpid, func>	Read the condition bit of co-processor to C bit of host processor; co-processor number is user defined.	Only 32-bit instructions exist. cprc32 <cpid, func>

- Description:** This instruction executes user defined read operation of co-processor condition bit. The code space of 12 low bits is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define relevant operations.
- Influence on flag bit:** The flag bit of main assembly line is decided by the condition bit of the appointed co-processor.
- Exception:** Illegal instruction exception

32-bit instruction

- Operation:** Read the condition bit of co-processor to C bit of host processor; co-processor number is user defined.
- Grammar:** cprc32 <cpid, func>
- Description:** This instruction executes user defined read operation of co-processor condition bit. The code space of 12 low bits is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define relevant operations.
- Influence on flag bit:** The flag bit of main assembly line is decided by the condition bit of the appointed co-processor.
- Exception:** Illegal instruction exception
- Instruction format:**

31 30	26 25	21 20	16 15	12 11	0
-------	-------	-------	-------	-------	---

1	1 1 1 1 1	0	CPID	C	0 1 0 0	User-define
---	-----------	---	------	---	---------	-------------



CPRCR – Read transfer from control register of co-processor

Unified instruction

Grammar	Operation	Compiling result
cprcr rz, <cpid, func>	Read the control register of co-processor to the general-purpose register of host processor; co-processor number and co-processor control register number are user defined.	Only 32-bit instructions exist. cprcr32 rz, <cpid, func>

Description: This instruction executes user defined read operation of co-processor control register. The code space of 12 low bits is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define co-processor control register number and relevant operations.

**Influence on flag
bit:** No influence

Exception: Illegal instruction exception

32-bit instruction

Operation: Read the control register of co-processor to the general-purpose register of host processor; co-processor number and co-processor control register number are user defined.

Grammar: cprcr32 rz, <cpid, func>

Description: This instruction executes user defined read operation of co-processor control register. The code space of 12 low bits is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define co-processor control register number and relevant operations.

**Influence on flag
bit:** No influence

Exception: Illegal instruction exception

Instruction**format:**

31 30 26 25 21 20 16 15 12 11 0

1	1	1	1	1	1	0	CPID	RZ	0	0	1	0	User-define
---	---	---	---	---	---	---	------	----	---	---	---	---	-------------

CPRGR – Read transfer from general-purpose register of co-processor

**Unified
instruction**

Grammar	Operation	Compiling result
cprgr rz, <cpid, func>	Read the general-purpose register of co-processor to the general-purpose register of host processor; co-processor number and co-processor general-purpose register number are user defined.	Only 32-bit instructions exist. cprgr32 rz, <cpid, func>

Description: This instruction executes user defined read operation of co-processor general-purpose register. The code space of 12 low bits is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define co-processor general-purpose register number and relevant operations.

Influence on flag bit: No influence

Exception: Illegal instruction exception

32-bit instruction

Operation: Read the general-purpose register of co-processor to the general-purpose register of host processor; co-processor number and co-processor general-purpose register number are user defined.

Grammar: cprgr32 rz, <cpid, func>

Description: This instruction executes user defined read operation of co-processor general-purpose register. The code space of 12 low bits is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define co-processor general-purpose register number and relevant operations.

Influence on flag No influence

bit:

Exception: Illegal instruction exception

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
1 1 1 1 1 1 0 CPID RZ 0 0 0 0 User-define					

CPWCR – Write transfer to control register of co-processor

Unified instruction

Grammar	Operation	Compiling result
cpwcr rx, <cpid, func>	Write contents in the general-purpose register of host processor into the control register of co-processor; co-processor number and co-processor control register number are user defined.	Only 32-bit instructions exist. cpwcr32 rx, <cpid, func>

Description: This instruction executes user defined write operation of co-processor control register. The code space of 12 low bits is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define co-processor control register number and relevant operations.

Influence on flag bit: No influence

Exception: Illegal instruction exception

32-bit instruction

Operation: Write contents in the general-purpose register of host processor into the control register of co-processor; co-processor number and co-processor control register number are user defined.

Grammar: cpwcr32 rx, <cpid, func>

Description: This instruction executes user defined write operation of co-processor control register. The code space of 12 low bits is reserved for user defined application. Among them, bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define co-processor control register number and relevant operations.

Influence on flag bit: No influence

Exception: Illegal instruction exception

Instruction**format:**

31 30 26 25 21 20 16 15 12 11 0

1	1	1	1	1	1	0	CPID	RX	0	0	1	1	User-define
---	---	---	---	---	---	---	------	----	---	---	---	---	-------------

CPWGR – Write transfer to general-purpose register of co-processor

Unified instruction

Grammar	Operation	Compiling result
cpwgr rx, <cpid, func>	Write contents in the general-purpose register of host processor into the general-purpose register of co-processor; co-processor number and co-processor general-purpose register number are user defined.	Only 32-bit instructions exist. cpwgr32 rx, <cpid, func>

Description: This instruction executes user defined write operation of co-processor general-purpose register. The code space of 12 low bits is reserved for user defined application. Among them, bits 8-11 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define co-processor general-purpose register number and relevant operations.

Influence on flag bit: No influence

Exception: Illegal instruction exception

32-bit instruction

Operation: Write contents in the general-purpose register of host processor into the general-purpose register of co-processor; co-processor number and co-processor general-purpose register number are user defined.

Grammar: cpwgr32 rx, <cpid, func>

Description: This instruction executes user defined write operation of co-processor general-purpose register. The code space of 12 low bits is reserved for user defined application. Among them, bits 8-11 are agreed as co-processor numbers and used to assign co-processor of pre-operation. As for the remaining bits, users will define co-processor general-purpose register number and relevant operations.

Influence on flag No influence

bit:

Exception: Illegal instruction exception

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
1 1 1 1 1 1	CPRZ	RX	0 0 0 1	User-define	

DECF – C=0 SUBTRACT IMMEDIATE

**Unified
instruction**

Grammar	Operation	Compiling result
decf rz, rx, imm5	if C==0, then $RZ \leftarrow RX - \text{zero_extend(imm5)};$ else $RZ \leftarrow RZ;$	Only 32-bit instructions exist. decf32 rz, rx, imm5

Description: If the condition bit C is 0, zero-extend the 5-bit immediate operand to 32 bits, subtract this 32-bit value from RX value, and save the result in RZ; otherwise, keep the values of RZ and RX unchanged.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0-31.

Exception: None

32-bit instruction

Operation: if C==0, then
 $RZ \leftarrow RX - \text{zero_extend(imm5)};$
 else
 $RZ \leftarrow RZ;$

Grammar: decf32 rz, rx, imm5

Description: If the condition bit C is 0, zero-extend the 5-bit immediate operand to 32 bits, subtract this 32-bit value from RX value, and save the result in RZ; otherwise, keep the values of RZ and RX unchanged.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 1 0 0	IMM5
---	-----------	----	----	-------------	-----------	------



DECGT – Set C bit when greater than zero in subtraction

Unified instruction

Grammar	Operation	Compiling result
decgt rz, rx, imm5	RZ \leftarrow RX - zero_extend(IMM5); If RZ > 0, then C \leftarrow 1; else C \leftarrow 0;	Only 32-bit instructions exist. decgt32 rz, rx, imm5

- Description:** Zero-extend the 5-bit immediate operand to 32 bits and save the result of subtracting this 32-bit value from RX value in RZ. The subtraction result is considered as signed number of complement form. If the result is greater than zero, set the condition bit C; otherwise, clear the condition bit C.
- Influence on flag bit:** If the subtraction result is greater than zero, set the condition bit C; otherwise, clear the condition bit C.
- Restriction:** The range of immediate operand is 0-31.
- Exception:** None

32-bit instruction

- Operation:** RZ \leftarrow RX - zero_extend(IMM5);
If RZ > 0, then
 C \leftarrow 1;
else
 C \leftarrow 0;
- Grammar:** decgt32 rz, rx, imm5
- Description:** Zero-extend the 5-bit immediate operand to 32 bits and save the result of subtracting this 32-bit value from RX value in RZ. The subtraction result is considered as signed number of complement form. If the result is greater than zero, set the condition bit C; otherwise, clear the condition bit C.
- Influence on flag bit:** If the subtraction result is greater than zero, set the condition bit C; otherwise, clear the condition bit C.

Restriction: The range of immediate operand is 0-31.

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 IMM5 RX 0 0 0 1 0 0 0 0 0 0 1 RZ						

DECLT – Set C bit when smaller than zero in subtraction

Unified instruction

Grammar	Operation	Compiling result
declt rz, rx, imm5	$RZ \leftarrow RX - \text{zero_extend(IMM5)};$ If $RZ < 0$, then $C \leftarrow 1;$ else $C \leftarrow 0;$	Only 32-bit instructions exist. declt32 rz, rx, imm5

- Description:** Zero-extend the 5-bit immediate operand to 32 bits and save the result of subtracting this 32-bit value from RX value in RZ. The subtraction result is considered as signed number of complement form. If the result is smaller than zero, set the condition bit C; otherwise, clear the condition bit C.
- Influence on flag bit:** If the subtraction result is smaller than zero, set the condition bit C; otherwise, clear the condition bit C.
- Restriction:** The range of immediate operand is 0-31.
- Exception:** None

32-bit instruction

- Operation:** $RZ \leftarrow RX - \text{zero_extend(IMM5)};$
 If $RZ < 0$, then
 $C \leftarrow 1;$
 else
 $C \leftarrow 0;$
- Grammar:** declt32 rz, rx, imm5
- Description:** Zero-extend the 5-bit immediate operand to 32 bits and save the result of subtracting this 32-bit value from RX value in RZ. The subtraction result is considered as signed number of complement form. If the result is smaller than zero, set the condition bit C; otherwise, clear the condition bit C.
- Influence on flag bit:** If the subtraction result is smaller than zero, set the condition bit C; otherwise, clear the condition bit C.

Restriction: The range of immediate operand is 0-31.

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 IMM5 RX 0 0 0 1 0 0 0 0 0 1 0 RZ						

DECNE – Set C bit when not equal to zero in subtraction

Unified instruction

Grammar	Operation	Compiling result
decne rz, rx, imm5	RZ \leftarrow RX - zero_extend(IMM5); If RZ \neq 0, then C \leftarrow 1; else C \leftarrow 0;	Only 32-bit instructions exist. decne32 rz, rx, imm5

Description: Zero-extend the 5-bit immediate operand to 32 bits and save the result of subtracting this 32-bit value from RX value in RZ. If the result is not equal to zero, set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: If the subtraction result is not equal to zero, set the condition bit C; otherwise, clear the condition bit C.

Restriction: The range of immediate operand is 0-31.

Exception: None

32-bit instruction

Operation: RZ \leftarrow RX - zero_extend(IMM5);
If RZ \neq 0, then
 C \leftarrow 1;
else
 C \leftarrow 0;

Grammar: decne32 rz, rx, imm5

Description: Zero-extend the 5-bit immediate operand to 32 bits and save the result of subtracting this 32-bit value from RX value in RZ. If the result is not equal to zero, set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: If the subtraction result is not equal to zero, set the condition bit C; otherwise, clear the condition bit C.

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction**format:**

31 30 26 25 21 20 16 15 109 5 4 0

1	1 0 0 0 1	IMM5	RX	0 0 0 1 0 0	0 0 1 0 0	RZ
---	-----------	------	----	-------------	-----------	----



DECT – C=1 subtract immediate

Unified instruction

Grammar	Operation	Compiling result
dect rz, rx, imm5	if C==1, then $RZ \leftarrow RX - \text{zero_extend(imm5)};$ else $RZ \leftarrow RZ;$	Only 32-bit instructions exist. dect32 rz, rx, imm5

- Description:** If the condition bit C is 1, zero-extend the 5-bit immediate operand to 32 bits, subtract this 32-bit value from RX value, and save the result in RZ; otherwise, keep the values of RZ and RX unchanged.
- Influence on flag bit:** No influence
- Restriction:** The range of immediate operand is 0-31.
- Exception:** None

32-bit instruction

- Operation:** if C==1, then
 $RZ \leftarrow RX - \text{zero_extend(imm5)};$
 else
 $RZ \leftarrow RZ;$
- Grammar:** dect32 rz, rx, imm5
- Description:** If the condition bit C is 1, zero-extend the 5-bit immediate operand to 32 bits, subtract this 32-bit value from RX value, and save the result in RZ; otherwise, keep the values of RZ and RX unchanged.
- Influence on flag bit:** No influence
- Restriction:** The range of immediate operand is 0-31.
- Exception:** None
- Instruction format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 1 0 0 0	IMM5
---	-----------	----	----	-------------	-----------	------



DIVS – Divide signed

Unified instruction

Grammar	Operation	Compiling result
divs rz, rx, ry	Divide signed $RZ = RX / RY$	Only 32-bit instructions exist. divs32 rz, rx, ry

Description: This instruction divides RX value of register by RY value of register, and saves the quotient in RZ. The values of RX, RY and RZ are considered as 32-bit signed numbers.

Attention: There is no definition for the result of dividing 0x80000000 by 0xffffffff.

**Influence on flag
bit:** No influence

Exception: Division by zero exception

32-bit instruction

Operation: Divide signed
 $RZ = RX / RY$

Grammar: divs32 rz, rx, ry

Description: This instruction divides RX value of register by RY value of register, and saves the quotient in RZ. The values of RX, RY and RZ are considered as 32-bit signed numbers.

Attention: There is no definition for the result of dividing 0x80000000 by 0xffffffff.

**Influence on flag
bit:** No influence

Exception: Division by zero exception

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	RY	RX	1 0 0 0 0 0	0 0 0 1 0	RZ
---	-----------	----	----	-------------	-----------	----



DIVU – Divide unsigned

**Unified
instruction**

Grammar	Operation	Description
divu rz, rx, ry	Divide unsigned $RZ = RX / RY$	Only 32-bit instructions exist. divu32 rz, rx, ry

Description: This instruction divides RX value of register by RY value of register, and saves the quotient in RZ. The values of RX, RY and RZ are considered as 32-bit unsigned numbers.

Influence on flag bit: No influence

Exception: Division by zero exception

32-bit instruction

Operation: Divide unsigned
 $RZ = RX / RY$

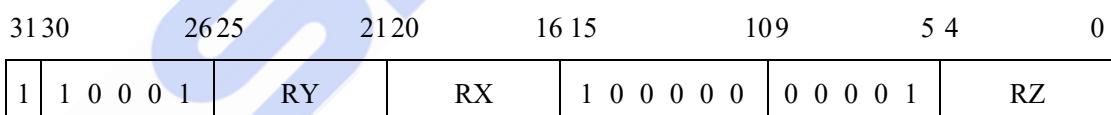
Grammar: divu32 rz, rx, ry

Description: This instruction divides RX value of register by RY value of register, and saves the quotient in RZ. The values of RX, RY and RZ are considered as 32-bit unsigned numbers.

Influence on flag bit: No influence

Exception: Division by zero exception

**Instruction
format:**



DOZE – Enter low power consumption doze mode

Unified instruction

Grammar	Operation	Compiling result
doze	Enter low power consumption doze mode	Only 32-bit instructions exist. doze32

Description: This instruction makes the processor enter low power consumption doze mode and wait for an interrupt to exit from this mode. At this time, CPU clock is stopped and corresponding peripheral equipment is also stopped.

Influence on flag bit: No influence

Exception: Privilege violation exception

32-bit instruction

Operation: Enter low power consumption doze mode

Grammar: doze32

Attribute: Privileged instruction

Description: This instruction makes the processor enter low power consumption doze mode and wait for an interrupt to exit from this mode. At this time, CPU clock is stopped and corresponding peripheral equipment is also stopped.

Influence on flag bit: No influence

Exception: Privilege violation exception

Instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 1 0 0	0 0 0 0 1	0 0 0 0 0	

FF0 – Fast find 0

Unified instruction

Grammar	Operation	Compiling result
ff0 rz, rx	RZ ← find_first_0(RX);	Only 32-bit instructions exist. ff0.32 rz, rx

Description: Find the first bit that is 0 in RX and return the search result to RZ. The search order is from the highest bit to the lowest bit of RX. If the highest bit (RX[31]) of RX is 0, return the value of 0 to RZ. If no bit of 0 exists in RX, return the value of 32 to RZ.

Influence on flag No influence

bit:

Exception: None

32-bit instruction

Operation: RZ ← find_first_0(RX);

Grammar: ff0.32 rz, rx

Description: Find the first bit that is 0 in RX and return the search result to RZ. The search order is from the highest bit to the lowest bit of RX. If the highest bit (RX[31]) of RX is 0, return the value of 0 to RZ. If no bit of 0 exists in RX, return the value of 32 to RZ.

Influence on flag No influence

bit:

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	0	0	0	0	0	1	RZ

FF1 – Fast find 1

Unified instruction

Grammar	Operation	Compiling result
ff1 rz, rx	RZ ← find_first_1(RX);	Only 32-bit instructions exist. ff1.32 rz, rx

Description: Find the first bit that is 1 in RX and return the search result to RZ. The search order is from the highest bit to the lowest bit of RX. If the highest bit (RX[31]) of RX is 1, return the value of 0 to RZ. If no bit of 1 exists in RX, return the value of 32 to RZ.

Influence on flag No influence

bit:

Exception: None

32-bit instruction

Operation: RZ ← find_first_1(RX);

Grammar: ff1.32 rz, rx

Description: Find the first bit that is 1 in RX and return the search result to RZ. The search order is from the highest bit to the lowest bit of RX. If the highest bit (RX[31]) of RX is 1, return the value of 0 to RZ. If no bit of 1 exists in RX, return the value of 32 to RZ.

Influence on flag No influence

bit:

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	0	0	0	0	0	1	0

GRS – Sign generation

Unified instruction

Grammar	Operation	Compiling result
grs rz, label	$RZ \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1);$	Only 32-bit instructions exist. grs32 rz, label
grs rz, imm32		grs32 rz, imm32

Description: Generate the value of sign and this value is determined by the location of label or 32-bit immediate operand (IMM32). The value of sign is gained by adding the current PC to the value of sign-extending the 18-bit relative offset shifting left by 1 bit to 32 bits. The effective range of sign value is the address space of $\pm 256KB$.

Influence on flag No influence

bit:

Exception: None

32-bit

instruction

Operation: $RZ \leftarrow PC + \text{sign_extend}(\text{offset} \ll 1);$

Grammar: grs rz, label

grs rz, imm32

Description: Generate the value of sign and this value is determined by the location of label or 32-bit immediate operand (IMM32). The value of sign is gained by adding the current PC to the value of sign-extending the 18-bit relative offset shifting left by 1 bit to 32 bits. The effective range of sign value is the address space of $\pm 256KB$.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	18 17	0
1 1 0 0 1 1	RZ	0 1 1		Offset

IDLY – Ban interrupt identification

Unified instruction

Grammar	Operation	Compiling result
idly n	Ban interrupt identification for n instructions	Only 32-bit instructions exist. idly32 n

Description: After idly, interrupt identification is banned for n instructions, thus an uninterruptible instruction sequence is executed in multitask environment.

Influence on flag bit: The flag bit C is cleared when idly instruction is executed. If exception happens to the n instructions after idly instruction is executed (including tracking or breakpoint exception), C bit is set as 1 and the interrupt instruction sequence can be observed.

Restriction: The instructions after idly instruction can only be arithmetic and logical instructions of single clock period, ld, st or branch instruction. In order to minimize some potential interrupt influences, other instructions should not be adopted; otherwise, we cannot guarantee that the instruction will not be interrupted. If there is another idly instruction in the instruction sequence after idly, it will be ignored. However, rte, rfi, doze, wait and stop instructions can stop idly instruction sequence.
Idly instruction is not allowed to appear in a cycle with less than 8 instructions.

Exception: None

Remark: If idly counter stays in a non-zero state, interrupt will be shielded. If a breakpoint exception or tracking exception happens in idly instruction sequence, C bit will be set as 1 and operation failure will happen to the sequence. In the process of exception handling, interrupt shielding becomes invalid, thus the counter will be cleared.

The idly counter remains unchanged in the debugging process of using HAD debugging port. Once the processor transforms into normal operation from debugging mode, count will be continued.

Note: n is decided by IMM5. When IMM5 is smaller than or equal to 3,

n is 4; when IMM5 is greater than 3, n is (IMM5+1).

IMM5:

00000 – n=4

00001 – n=4

00010 – n=4

00011 – n=4

00100 – n=5

.....

11111 – n=32

32-bit instruction

Operation:	Ban interrupt identification for n instructions disable_int_in_following(n);
Grammar:	idly32 n
Description:	After idly, interrupt identification is banned for n instructions, thus an uninterruptible instruction sequence is executed in multitask environment.
Influence on flag bit:	The flag bit C is cleared when idly instruction is executed. If exception happens to the n instructions after idly instruction is executed (including tracking or breakpoint exception), C bit is set as 1 and the interrupt instruction sequence can be observed.
Restriction:	The instructions after idly instruction can only be arithmetic and logical instructions of single clock period, ld, st or branch instruction. In order to minimize some potential interrupt influences, other instructions should not be adopted; otherwise, we cannot guarantee that the instruction will not be interrupted. If there is another idly instruction in the instruction sequence after idly, it will be ignored. However, rte, rfi, doze, wait and stop instructions can stop idly instruction sequence. Idly instruction is not allowed to appear in a cycle with less than 8 instructions.
Exception:	None
Remark:	If idly counter stays in a non-zero state, interrupt will be shielded. If a breakpoint exception or tracking exception happens in idly instruction sequence, C bit will be set as 1 and operation failure will happen to the sequence. In the process of exception handling, interrupt shielding becomes invalid, thus the counter will be cleared.

The idly counter remains unchanged in the debugging process of using HAD debugging port. Once the processor transforms into normal operation from debugging mode, count will be continued.

Note: n is decided by IMM5. When IMM5 is smaller than or equal to 3, n is 4; when IMM5 is greater than 3, n is (IMM5+1).

IMM5:

00000 – n=4

00001 – n=4

00010 – n=4

00011 – n=4

00100 – n=5

.....

11111 – n=32

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0 IMM5 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0						

INCF – C=0 add immediate

**Unified
instruction**

Grammar	Operation	Compiling result
incf rz, rx, imm5	if C==0, then $RZ \leftarrow RX + \text{zero_extend(IMM5)};$ else $RZ \leftarrow RZ;$	Only 32-bit instructions exist. incf32 rz, rx, imm5

Description: If the condition bit C is 0, zero-extend the 5-bit immediate operand to 32 bits, add this 32-bit value to RX value, and save the result in RZ; otherwise, keep the values of RZ and RX unchanged.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0-31.

Exception: None

32-bit instruction

Operation: if C==0, then
 $RZ \leftarrow RX + \text{zero_extend(IMM5)};$
 else
 $RZ \leftarrow RZ;$

Grammar: incf32 rz, rx, imm5

Description: If the condition bit C is 0, zero-extend the 5-bit immediate operand to 32 bits, add this 32-bit value to RX value, and save the result in RZ; otherwise, keep the values of RZ and RX unchanged.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 0 1	IMM5
---	-----------	----	----	-------------	-----------	------



INCT – C=1 add immediate

**Unified
instruction**

Grammar	Operation	Compiling result
inct rz, rx, imm5	if C==1, then $RZ \leftarrow RX + \text{zero_extend(IMM5)};$ else $RZ \leftarrow RZ;$	Only 32-bit instructions exist. inct32 rz, rx, imm5

Description: If the condition bit C is 1, zero-extend the 5-bit immediate operand to 32 bits, add this 32-bit value to RX value, and save the result in RZ; otherwise, keep the values of RZ and RX unchanged.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

32-bit instruction

Operation: if C==1, then
 $RZ \leftarrow RX + \text{zero_extend(IMM5)};$
 else
 $RZ \leftarrow RZ;$

Grammar: inct32 rz, rx, imm5

Description: If the condition bit C is 1, zero-extend the 5-bit immediate operand to 32 bits, add this 32-bit value to RX value, and save the result in RZ; otherwise, keep the values of RZ and RX unchanged.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 1 0	IMM5
---	-----------	----	----	-------------	-----------	------



INS – Bit insert

Unified instruction

Grammar	Operation	Compiling result
ins rz, rx, msb, lsb	RZ[MSB:LSB] ← RX[MSB-LSB:0]	Only 32-bit instructions exist. ins32 rz, rx, msb, lsb

Description: Insert a section of consecutive low bits of RX into a section of consecutive bits of RZ (RZ[MSB:LSB]) appointed by 2 5-bit immediate operands (MSB,LSB), and keep other bits of RZ unchanged; the consecutive low bit width of RX is assigned by MSB and LSB (i.e. RX[MSB-LSB:0]). If MSB is equal to 31 and LSB is equal to zero, RZ value is the same with RX value. If MSB is equal to LSB, then MSB (i.e. LSB) bit of RZ is the lowest bit of RX, and other bits remain unchanged. If MSB is smaller than LSB, behavior of this instruction cannot be predicted.

Influence on flag No influence

bit:

Restriction: The range of MSB is 0-31, the range of LSB is 0-31, and MSB should be greater than or equal to LSB.

Exception: None

32-bit instruction

Operation: RZ[MSB:LSB] ← RX[MSB-LSB:0]

Grammar: ins32 rz, rx, msb, lsb

Description: Insert a section of consecutive low bits of RX into a section of consecutive bits of RZ (RZ[MSB:LSB]) appointed by 2 5-bit immediate operands (MSB,LSB), and keep other bits of RZ unchanged; the consecutive low bit width of RX is assigned by MSB and LSB (i.e. RX[MSB-LSB:0]). If MSB is equal to 31 and LSB is equal to zero, RZ value is the same with RX value. If MSB is equal to LSB, then MSB (i.e. LSB) bit of RZ is the lowest bit of RX, and other bits remain unchanged. If MSB is smaller than LSB, behavior of this instruction

cannot be predicted.

Influence on flag No influence

bit:

Restriction: The range of MSB is 0-31, the range of LSB is 0-31, and MSB should be greater than or equal to LSB.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RZ	RX	0 1 0 1 1 1	SIZE	LSB	

SIZE field – Assign the width of inserted bit.

Attention: The binary operand SIZE is equal to MSB-LSB.

00000 – 1

00001 – 2

.....

11111 – 32

LSB field – Assign the bit that ends the insertion.

00000 – 0 bit

00001 – 1 bit

.....

11111 – 31 bits

IPOP – Interrupt pop

**Unified
instruction**

Grammar	Operation	Compiling result
ipop	<p>Load the interrupted general-purpose register site {R0~R3, R12, R13} from the stack pointer register, and then update the stack pointer register to the top of stack storage;</p> $\{R0\sim R3, R12, R13\} \leftarrow MEM[SP] \sim MEM[SP+20];$ $SP \leftarrow SP+24;$	Only 16-bit instructions exist. ipop;

Description: Load the interrupted general-purpose register site {R0~R3, R12, R13} from the stack pointer register, and then update the stack pointer register to the top of stack storage. Adopt direct addressing mode of stack pointer register.

Influence on flag bit: No influence

Exception: Access error exception and unaligned exception

16-bit instruction

Operation: Load the interrupted general-purpose register site {R0~R3, R12, R13} from the stack pointer register, and then update the stack pointer register to the top of stack storage;

$$\{R0\sim R3, R12, R13\} \leftarrow MEM[SP]\sim MEM[SP+20];$$

$$SP \leftarrow SP+24;$$

Grammar: ipop16

Description: Load the interrupted general-purpose register site {R0~R3, R12, R13} from the stack pointer register, and then update the stack pointer register to the top of stack storage. Adopt direct addressing mode of stack pointer register.

Influence on flag bit: No influence

Exception: Access error exception and unaligned exception

Instruction

format:

15 14	10 9	8	7	5	4	0
0	0 0 1 0 1	0	0	0 1 1	0 0 0 1 1	

IPUSH – Interrupt pop

Unified instruction

Grammar	Operation	Compiling result
ipush	<p>Store the interrupted general-purpose register site {R0~R3, R12, R13} to the stack storage, and then update the stack pointer register to the top of stack storage;</p> <p>MEM[SP-4]~MEM[SP-24] $\leftarrow \{R13, R12, R3~R0\};$ $SP \leftarrow SP-24;$</p>	<p>Only 16-bit instructions exist.</p> <p>ipush;</p>

Description: Store the interrupted general-purpose register site {R0~R3, R12, R13} to the stack storage, and then update the stack pointer register to the top of stack storage. Adopt direct addressing mode of stack pointer register.

Influence on flag bit: No influence

Exception: Access error exception and unaligned exception

16-bit instruction

Operation: Store the interrupted general-purpose register site {R0~R3, R12, R13} to the stack storage, and then update the stack pointer register to the top of stack storage;

MEM[SP-4]~MEM[SP-24] $\leftarrow \{R13, R12, R3~R0\};$
 $SP \leftarrow SP-24;$

Grammar: ipush16

Description: Store the interrupted general-purpose register site {R0~R3, R12, R13} to the stack storage, and then update the stack pointer register to the top of stack storage. Adopt direct addressing mode of stack pointer register.

Influence on flag bit: No influence

Exception: Access error exception and unaligned exception

Instruction

format:

15	14	10	9	8	7	5	4	0
0	0	0	1	0	1	0	0	1



IXH – Index half-word

**Unified
instruction**

Grammar	Operation	Compiling result
ixh rz, rx, ry	$RZ \leftarrow RX + (RY \ll 1)$	Only 32-bit instructions exist. ixh32 rz, rx, ry

Description: Make RY value shift left by one bit, add it to RX value, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

**32-bit
instruction**

Operation: $RZ \leftarrow RX + (RY \ll 1)$

Grammar: ixh32 rz, rx, ry

Description: Make RY value shift left by one bit, add it to RX value, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RY	RX	0	0	0	0	1

IXW – Index word

**Unified
instruction**

Grammar	Operation	Compiling result
ixw rz, rx, ry	$RZ \leftarrow RX + (RY \ll 2)$	Only 32-bit instructions exist. ixw32 rz, rx, ry

Description: Make RY value shift left by two bits, add it to RX value, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

32-bit

instruction

Operation: $RZ \leftarrow RX + (RY \ll 2)$

Grammar: ixw32 rz, rx, ry

Description: Make RY value shift left by two bits, add it to RX value, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 RY RX 0 0 0 0 1 0 0 0 0 1 0 RZ						

IXD – Index double word

**Unified
instruction**

Grammar	Operation	Compiling result
ixd rz, rx, ry	$RZ \leftarrow RX + (RY \ll 3)$	Only 32-bit instructions exist. ixd32 rz, rx, ry

Description: Make RY value shift left by three bits, add it to RX value, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

32-bit

instruction

Operation: $RZ \leftarrow RX + (RY \ll 3)$

Grammar: ixd32 rz, rx, ry

Description: Make RY value shift left by three bits, add it to RX value, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RY	RX	0	0	0	0	RZ

JMP – Register jump

Unified instruction

Grammar	Operation	Compiling result
jmp rx	Jump to the position appointed by register $PC \leftarrow RX \& 0xffffffff$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$), then jmp16 rx; else jmp32 rx;

Description: The program jumps to the position appointed by register RX and the lowest bit of RX is ignored. The jump range of JMP instruction is the whole address space of 4GB.

Influence on flag bit: No influence

Exception: None

16-bit instruction

Operation: Jump to the position appointed by register
 $PC \leftarrow RX \& 0xffffffff$

Grammar: jmp16 rx

Description: The program jumps to the position appointed by register RX and the lowest bit of RX is ignored. The jump range of JMP instruction is the whole address space of 4GB.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

15 14	10 9	6 5	2 1 0
0 1 1 1 1 0	0 0 0 0	RX	0 0

32-bit

instruction

Operation: Jump to the position appointed by register
 $PC \leftarrow RX \& 0xffffffff$

Grammar: jmp32 rx

Description: The program jumps to the position appointed by register RX and the lowest bit of RX is ignored. The jump range of JMP instruction is the whole address space of 4GB.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 0 1 1 0	RX	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	

JMPI – Jump indirect

Unified instruction

Grammar	Operation	Compiling result
jmpi label	The program jumps to the position appointed by storage $PC \leftarrow MEM[(PC + \text{zero_extend}(offset \ll 2)) \& 0xffffffff]$	Only 32-bit instructions exist. jmpi32 label

Description: The program jumps to label position and label is loaded by storage. The storage address is gained by adding the current PC to the value of unsigned extending the 16-bit relative offset shifting left by 2 bits to 32 bits, and compulsively clearing the two lowest bits. The jump range of JMPI instruction is the whole address space of 4GB.

Influence on flag bit: No influence

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: The program jumps to the position appointed by storage
 $PC \leftarrow MEM[(PC + \text{zero_extend}(offset \ll 2)) \& 0xffffffff]$

Grammar: jmpi32 label

Description: The program jumps to label position and label is loaded by storage. The storage address is gained by adding the current PC to the value of unsigned extending the 16-bit relative offset shifting left by 2 bits to 32 bits, and compulsively clearing the two lowest bits. The jump range of JMPI instruction is the whole address space of 4GB.

Influence on flag bit: No influence

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	1 0 1 1 0	0 0 0 0 0		Offset

JMPIX – Register index jump

**Unified
instruction**

Grammar	Operation	Compiling result
jmpix rx, imm	Jump to the position appointed by register index $PC \leftarrow SVBR + (RX \& 0xff) * IMM$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$), then jmpix16 rx; else jmpix32 rx;

Description: The program jumps to the position of $SVBR + RX[7:0] * IMM$, $IMM \in \{16, 24, 32, 40\}$. The 24th high bit of RX is ignored.

Influence on flag bit: No influence

Exception: None

**16-bit
instruction**

Operation: Jump to the position appointed by register index
 $PC \leftarrow SVBR + (RX \& 0xff) * IMM$

Grammar: jmpix16 rx, imm

Description: The program jumps to the position of $SVBR + RX[7:0] * IMM$, $IMM \in \{16, 24, 32, 40\}$. The 24th high bit of RX is ignored.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

15 14	11 10	8 7	3	0
-------	-------	-----	---	---

0	0 1 1 1	RX	1 1 1 0 0 0	IMM2
---	---------	----	-------------	------

IMM2 field – Assign the value of immediate operand.

Attention: The corresponding relation between IMM2 value of binary coding and IMM value in jump instruction is as follows:

2'b00 – *16

2'b01 – *24

2'b10 – *32

2'b11 – *40

32-bit

instruction

Operation: Jump to the position appointed by register index

$PC \leftarrow SVBR + (RX \& 0xff) * IMM$

Grammar: jmpix32 rx, imm

Description: The program jumps to the position of $SVBR + RX[7:0] * IMM$, $IMM \in \{16, 24, 32, 40\}$. The 24th high bit of RX is ignored.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	2 1 0
1 1 1 0 1 0	0 1 1 1 1 RX	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 IMM2		

IMM2 field – Assign the value of immediate operand.

Attention: The corresponding relation between IMM2 value of binary coding and IMM value in jump instruction is as follows:

2'b00 – *16

2'b01 – *24

2'b10 – *32

2'b11 – *40

JSR – Register jump to subprogram

Unified instruction

Grammar	Operation	Compiling result
jsr rx	<p>Link and jump to the subprogram position appointed by register</p> $R15 \leftarrow PC + 4,$ $PC \leftarrow RX \& 0xffffffff$	<p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of register.</p> <pre>if(x<16), then jsr16 rx; else jsr32 rx;</pre>

Description: This instruction saves the return address (PC of the next instruction, i.e. PC+4 at present) of the subprogram in link register R15, the program is executed after jumping to the subprogram position appointed by contents of register RX, and the lowest bit of RX is ignored. The jump range of JSR instruction is the whole address space of 4GB.

Influence on flag bit: No influence

Exception: None

16-bit instruction

Operation: Link and jump to the subprogram position appointed by register

$$R15 \leftarrow PC + 4, PC \leftarrow RX \& 0xffffffff$$

Grammar: jsr16 rx

Description: This instruction saves the return address (PC of the next instruction, i.e. PC+4 at present) of the subprogram in link register R15, the program is executed after jumping to the subprogram position appointed by contents of register RX, and the lowest bit of RX is ignored. The jump range of JSR instruction is the whole address space of 4GB.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1	0
0 1 1 1 1 0	1 1 1 1	RX	0 1	

32-bit

instruction

Operation: Link and jump to the subprogram position appointed by register

$R15 \leftarrow PC + 4, PC \leftarrow RX \& 0xffffffff$

Grammar: jsr32 rx

Description: This instruction saves the return address (PC of the next instruction, i.e. $PC+4$ at present) of the subprogram in link register R15, the program is executed after jumping to the subprogram position appointed by contents of register RX, and the lowest bit of RX is ignored. The jump range of JSR instruction is the whole address space of 4GB.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 0 1 1 1	RX	0 0	

JSRI – Jump to subprogram indirect

Unified instruction

Grammar	Operation	Compiling result
jsri label	The program jumps to the subprogram position appointed by storage $R15 \leftarrow \text{next PC},$ $\text{PC} \leftarrow \text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xfffffffffc]$	Only 32-bit instructions exist. jsri32 label;

Description: This instruction saves the return address (PC of the next instruction) of the subprogram in link register R15, the program is executed after jumping to the label position, and label is loaded by storage. The jump range of JSR instruction is the whole address space of 4GB. The storage address is gained by adding the current PC to the value of unsigned extending the 16-bit relative offset shifting left by 2 bits to 32 bits, and compulsively clearing the two lowest bits. The jump range of JSRI instruction is the whole address space of 4GB.

Influence on flag bit: No influence

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: The program jumps to the subprogram position appointed by storage

 $R15 \leftarrow \text{PC} + 4, \text{PC} \leftarrow \text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xfffffffffc]$

Grammar: jsri32 label

Description: This instruction saves the return address (PC of the next instruction, i.e. PC+4 at present) of the subprogram in link register R15, the program is executed after jumping to the label position, and label is loaded by storage. The jump range of JSR instruction is the whole address space of

4GB. The storage address is gained by adding the current PC to the value of unsigned extending the 16-bit relative offset shifting left by 2 bits to 32 bits, and compulsively clearing the two lowest bits. The jump range of JSRI instruction is the whole address space of 4GB.

Influence on flag No influence

bit:

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	1 0 1 1 1	0 0 0 0 0		Offset

LD.B – Load unsigned and extended byte

Unified instruction

Grammar	Operation	Compiling result
ld.b rz,(rx, disp)	$RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset})])$	Compiled into 16-bit or 32-bit instructions according to the range of offset and register. if (disp<32)and(x<7) and (z<7), then ld16.b rz, (rx, disp); else ld32.b rz, (rx, disp);

Description: Save the byte loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset to 32 bits. The address space of LD.B instruction is +4KB.
 Attention: The offset DISP is the offset of binary operand.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

16-bit instruction

Operation: Load byte from storage to register, extend unsigned
 $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset})])$

Grammar: ld16.b rz, (rx, disp)

Description: Save the byte loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 5-bit relative offset to 32 bits. The address space of LD16.B instruction is +32B.
 Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

15 14	11 10	8 7	5 4	0
-------	-------	-----	-----	---

1	0 0 0 0	RX	RZ	Offset
---	---------	----	----	--------

32-bit

instruction

Operation: Load byte from storage to register, extend unsigned

$$RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset})])$$

Grammar: ld32.b rz, (rx, disp)

Description: Save the byte loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset to 32 bits. The address space of LD32.B instruction is +4KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
-------	-------	-------	-------	-------	---

1	1 0 1 1 0	RZ	RX	0 0 0 0	Offset
---	-----------	----	----	---------	--------

LD.BS – Load signed and extended byte

Unified instruction

Grammar	Operation	Compiling result
ld.bs rz, (rx, disp)	$RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset})])$	Only 32-bit instructions exist. ld32.bs rz, (rx, disp)

Description: Save byte loaded from storage in register RZ after sign-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset to 32 bits. The address space of LD.BS instruction is +4KB.
 Attention: The offset DISP is the offset of binary operand.

**Influence on flag
bit:** No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load byte from storage to register, extend signed
 $RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset})])$

Grammar: ld32.bs rz, (rx, disp)

Description: Save byte loaded from storage in register RZ after sign-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset to 32 bits. The address space of LD32.BS instruction is +4KB.
 Attention: The offset DISP is the offset of binary operand.

**Influence on flag
bit:** No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction**format:**

3130 2625 2120 1615 1211 0

1 1 0 1 1 0	RZ	RX	0 1 0 0	Offset
---------------	----	----	---------	--------

LD.D – Load double word

Unified instruction

Grammar	Operation	Compiling result
ld.d rz, (rx, disp)	$RZ \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2)]$ $RZ + 1 \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2) + 0x4]$	Only 32-bit instructions exist. ld32.d rz, (rx, disp);

Description: Load double word from storage to register RZ and RZ + 1. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 2 bits to 32 bits. The address space of LD.D instruction is +16KB.
 Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load double word from storage to register
 $RZ \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2)]$
 $RZ + 1 \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2) + 0x4]$

Grammar: ld32.d rz, (rx, disp)

Description: Load double word from storage to register RZ and RZ + 1. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 2 bits to 32 bits. The address space of LD32.D instruction is +16KB.
 Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30 26 25 21 20 16 15 12 11 0

1	1 0 1 1 0	RZ	RX	0 0 1 1	Offset
---	-----------	----	----	---------	--------

LD.H – Load unsigned and extended half-word

**Unified
instruction**

Grammar	Operation	Compiling result
ld.h rz, (rx, disp)	$RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 1)])$	Compiled into 16-bit or 32-bit instructions according to the range of offset and register. if (disp<64)and(x<7) and (z<7), then ld16.h rz, (rx, disp); else ld32.h rz, (rx, disp);

Description: Save half-word loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 1 bit to 32 bits. The address space of LD.H instruction is +8KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 1 bit.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

16-bit instruction

Operation: Load half-word from storage to register, extend unsigned
 $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 1)])$

Grammar: ld16.h rz, (rx, disp)

Description: Save half-word loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 5-bit relative offset shifting left by 1 bit to 32 bits. The address space of LD16.H

instruction is +64B.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 1 bit.

Influence on flag bit: No influence

Restriction: The range of register is r0-r7.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

15	14	11	10	8	7	5	4	0
1	0 0 0 1	RX	RZ	Offset				

32-bit instruction

Operation: Load half-word from storage to register, extend unsigned
 $RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 1)])$

Grammar: ld32.h rz, (rx, disp)

Description: Save half-word loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 1 bit to 32 bits. The address space of LD32.H instruction is +8KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 1 bit.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31	30	26	25	21	20	16	15	12	11	0
1	1 0 1 1 0	RZ	RX	0 0 0 1	Offset					

LD.HS – Load signed and extended half-word

Unified instruction

Grammar	Operation	Compiling result
ld.hs rz, (rx, disp)	$RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 1)])$	Only 32-bit instructions exist. ld32.hs rz, (rx, disp)

Description: Save half-word loaded from storage in register RZ after sign-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 1 bit to 32 bits. The address space of LD.HS instruction is +8KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 1 bit.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load half-word from storage to register, extend signed
 $RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 1)])$

Grammar: ld32.hs rz, (rx, disp)

Description: Save half-word loaded from storage in register RZ after signed extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 1 bit to 32 bits. The address space of LD32.HS instruction is +8KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 1 bit.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30 26 25 21 20 16 15 12 11 0

1	1 0 1 1 0	RZ	RX	0 1 0 1	Offset
---	-----------	----	----	---------	--------

LD.W – Load word

Unified instruction

Grammar	Operation	Compiling result
ld.w rz, (rx, disp)	$RZ \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2)]$	Compiled into 16-bit or 32-bit instructions according to the range of offset and register. if ($x=sp$) and ($z<7$) and ($\text{disp} < 1024$), ld16.w rz, (sp, disp); else if ($\text{disp}<128$) and ($x<7$) and ($z<7$), ld16.w rz, (rx, disp); else ld32.w rz, (rx, disp);

Description: Load word from storage to register RZ. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value obtained of unsigned extending the 12-bit relative offset shifting left by 2 bits to 32 bits. The address space of LD.W instruction is +16KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

16-bit instruction

Operation: Load word from storage to register

$RZ \leftarrow \text{MEM}[RX + \text{sign_extend}(\text{offset} \ll 2)]$

Grammar: ld16.w rz, (rx, disp)

ld16.w rz, (sp, disp)

Description: Load word from storage to register RZ. Adopt the addressing mode of register and immediate operand offset. When RX is SP, the effective address of storage is gained by adding the base register RX to the value

of unsigned extending the 8-bit relative offset shifting left by 2 bits to 32 bits. When rx is other register, the effective address of storage is gained by adding the base register RX to the value of unsigned extending the 5-bit relative offset shifting left by 2 bits to 32 bits. The address space of LD16.W instruction is +1KB.

Attention: The offset DISP is gained after the binary operand IMM5 shifts left by 2 bits. When the base register RX is SP, the offset DISP is gained after the binary operand {IMM3, IMM5} shifts left by 2 bits.

Influence on flag bit: No influence

Restriction: The range of register is r0-r7.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction format:

ld16.w rz, (rx, disp)

15	14	11 10	8	7	5	4	0
1	0 0 1 0	RX	RZ	IMM5			

ld16.w rz, (sp, disp)

15	14	11 10	8	7	5	4	0
1	0 0 1 1	IMM3	RZ	IMM5			

32-bit instruction

Operation: Load word from storage to register

$$RZ \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2)]$$

Grammar: ld32.w rz, (rx, disp)

Description: Load word from storage to register RZ. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 2 bits to 32 bits. The address space of LD32.W instruction is +16KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30 26 25 21 20 16 15 12 11 0

1	1 0 1 1 0	RZ	RX	0 0 1 0	Offset
---	-----------	----	----	---------	--------

LDCPR – Load word to co-processor

Unified instruction

Grammar	Operation	Compiling result
ldcpr <cpid, cprz>, (rx, offset)	CPRZ \leftarrow MEM[RX + zero_extend(offset << 2)]	Only 32-bit instructions exist. ldcpr32 <cpid, cprz>, (rx, offset)

Description: Load word from storage to general-purpose register of co-processor CPRZ. Adopt the addressing mode of register and immediate operand offset. Bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. 12 low bits are user defined.

**Influence on flag
bit:**

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load word from storage to general-purpose register of co-processor
CPRZ \leftarrow MEM[RX + zero_extend(offset << 2)]

Grammar: ldcpr32 <cpid, cprz>, (rx, offset)

Description: Load word from storage to general-purpose register of co-processor CPRZ. Adopt the addressing mode of register and immediate operand offset. Bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. 12 low bits are user defined.

**Influence on flag
bit:**

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

**Instruction
format:**

31 30	26 25	21 20	16 15	12 11	0
-------	-------	-------	-------	-------	---

1	1	1	0	1	0	0	CPIID	RX	0	1	1	0	User-define
---	---	---	---	---	---	---	-------	----	---	---	---	---	-------------



LDEX.W – Load word exclusive

Unified instruction

Grammar	Operation	Compiling result
ldex.w rz, (rx, disp)	$RZ \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2)]$	Only 32-bit instructions exist. ldex32.w rz, (rx, disp)

Description: Load word from storage to general-purpose register RZ. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 2 bits to 32 bits. The address space of LDEX.W instruction is +16KB.

This instruction matches STEX.W and it is used for atom operation of “read storage – modify – write storage” in multi-core communication.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load word from storage to general-purpose register
 $RZ \leftarrow \text{MEM}[RX + \text{zero_extend}(\text{offset} \ll 2)]$

Grammar: ldex32.w rz, (rx, disp)

Description: Load word from storage to general-purpose register RZ. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 2 bits to 32 bits. The address space of LDEX32.W instruction is +16KB.

This instruction matches STEX32.W and it is used for atom operation of “read storage – modify – write storage” in multi-core communication.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30 26 25 21 20 16 15 12 11 0

1	1 0 1 1 0	RZ	RX	0 1 1 1	Offset
---	-----------	----	----	---------	--------

LDM – Load consecutive multiword

**Unified
instruction**

Grammar	Operation	Compiling result
ldm ry-rz, (rx)	<p>Load multiple consecutive words from storage to a group of consecutive register files</p> <pre>dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ Rdst ← MEM[addr]; dst ← dst + 1; addr ← addr + 4; }</pre>	<p>Only 32-bit instructions exist.</p> <p>ldm32 ry-rz, (rx);</p>

Description: Load multiple consecutive words from storage to a group of consecutive register files starting from register RY. In another word, load the first word in the address appointed by storage to register RY; load the second word to register RY+1, and the like; load the last word to register RZ. The effective address of storage is decided by the contents of base register RX.

Influence on flag bit: No influence

Restriction: RZ should be greater than and equal to RY.

The base register RX should not be included within the range of RY-RZ; otherwise, the result will be unpredictable.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

**32-bit
instruction
Operation:**

Load multiple consecutive words from storage to a group of consecutive register files

```
dst ← Y; addr ← RX;
for (n = 0; n <= IMM5; n++){
```

```

Rdst ← MEM[addr];
dst ← dst + 1;
addr ← addr + 4;
}

```

Grammar: ldm32 ry-rz, (rx)

Description: Load multiple consecutive words from storage to a group of consecutive register files starting from register RY. In another word, load the first word in the address appointed by storage to register RY; load the second word to register RY+1, and the like; load the last word to register RZ. The effective address of storage is decided by the contents of base register RX.

Influence on No influence

flag bit:

Restriction: RZ should be greater than and equal to RY.

The base register RX should not be included within the range of RY-RZ; otherwise, the result will be unpredictable.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 1 0 0	RY	RX	0 0 0 1 1 1	0 0 0 0 1	IMM5	

IMM5 field – Assign the number of destination registers, IMM5 = Z – Y.

00000 – 1 destination register

00001 – 2 destination registers

.....

11111 – 32 destination registers

LDQ – Load consecutive quad word#

Unified instruction

Grammar	Operation	Compiling result
ldq r4-r7, (rx)	<p>Load four consecutive words from storage to registers R4-R7</p> <p>dst \leftarrow 4; addr \leftarrow RX;</p> <pre>for (n = 0; n <= 3; n++) { Rdst \leftarrow MEM[addr]; dst \leftarrow dst + 1; addr \leftarrow addr + 4; }</pre>	<p>Only 32-bit instructions exist.</p> <p>ldq32 r4-r7, (rx);</p>

Description: Load 4 consecutive words from storage to register file [R4, R7] (including boundary) successively. In another word, load the first word in the address appointed by storage to register R4, load the second word to register R5, load the third word to register R6, and load the fourth word to register R7. The effective address of storage is decided by the contents of base register RX.

Attention: This instruction is the pseudo instruction of ldm r4-r7, (rx).

Influence on flag bit: No influence

Restriction: The base register RX should not be included within the range of R4-R7; otherwise, the result will be unpredictable.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load four consecutive words from storage to registers R4-R7

```
dst  $\leftarrow$  4; addr  $\leftarrow$  RX;
for (n = 0; n <= 3; n++) {
    Rdst  $\leftarrow$  MEM[addr];
    dst  $\leftarrow$  dst + 1;
```

addr ← addr + 4;

}

Grammar: ldq32 r4-r7, (rx)

Description: Load 4 consecutive words from storage to register file [R4, R7] (including boundary) successively. In another word, load the first word in the address appointed by storage to register R4, load the second word to register R5, load the third word to register R6, and load the fourth word to register R7. The effective address of storage is decided by the contents of base register RX.

Attention: This instruction is the pseudo instruction of ldm32 r4-r7, (rx).

Influence on flag No influence

bit:

Restriction: The base register RX should not be included within the range of R4-R7; otherwise, the result will be unpredictable.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 1 0 0	0 0 1 0 0	RX	0 0 0 1 1 1	0 0 0 0 1	0 0 0 1 1	

LDR.B – Load unsigned and extended byte in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
ldr.b rz, (rx, ry << 0)	Load byte from storage to register, extend unsigned	Only 32-bit instructions exist.
ldr.b rz, (rx, ry << 1)		ldr32.b rz, (rx, ry << 0)
ldr.b rz, (rx, ry << 2)	RZ \leftarrow zero_extend(MEM[RX + RY << IMM2])	ldr32.b rz, (rx, ry << 1)
ldr.b rz, (rx, ry << 3)		ldr32.b rz, (rx, ry << 2)
		ldr32.b rz, (rx, ry << 3)

Description: Save byte loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load byte from storage to register, extend unsigned

$$RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + RY << IMM2])$$

Grammar:
 ldr32.b rz, (rx, ry << 0)
 ldr32.b rz, (rx, ry << 1)
 ldr32.b rz, (rx, ry << 2)
 ldr32.b rz, (rx, ry << 3)

Description: Save byte loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

ldr32.b rz, (rx, ry << 0)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 0 0 0	0 0 0 0 1	RZ						

ldr32.b rz, (rx, ry << 1)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 0 0 0	0 0 0 1 0	RZ						

ldr32.b rz, (rx, ry << 2)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 0 0 0	0 0 1 0 0	RZ						

ldr32.b rz, (rx, ry << 3)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 0 0 0	0 1 0 0 0	RZ						

LDR.BS – Load signed and extended byte in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
ldr.bs rz, (rx, ry << 0)	Load byte from storage to register, extend signed	Only 32-bit instructions exist.
ldr.bs rz, (rx, ry << 1)		ldr32.bs rz, (rx, ry << 0)
ldr.bs rz, (rx, ry << 2)	RZ \leftarrow sign_extend(MEM[RX + RY << IMM2])	ldr32.bs rz, (rx, ry << 1)
ldr.bs rz, (rx, ry << 3)		ldr32.bs rz, (rx, ry << 2)
		ldr32.bs rz, (rx, ry << 3)

Description: Save byte loaded from storage in register RZ after sign-extension to 32 bits. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load byte from storage to register, extend signed

$$RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + RY << IMM2])$$

Grammar: ldr32.bs rz, (rx, ry << 0)
 ldr32.bs rz, (rx, ry << 1)
 ldr32.bs rz, (rx, ry << 2)
 ldr32.bs rz, (rx, ry << 3)

Description: Save byte loaded from storage in register RZ after sign-extension to 32 bits. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

ldr32.bs rz, (rx, ry<< 0)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 1 0 0	0 0 0 0 1	RZ					

ldr32.bs rz, (rx, ry<< 1)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 1 0 0	0 0 0 1 0	RZ					

ldr32.bs rz, (rx, ry<< 2)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 1 0 0	0 0 1 0 0	RZ					

ldr32.bs rz, (rx, ry<< 3)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 1 0 0	0 1 0 0 0	RZ					

LDR.H – Load unsigned and extended half-word in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
ldr.h rz, (rx, ry << 0)	Load half-word from storage to register, extend unsigned	Only 32-bit instructions exist.
ldr.h rz, (rx, ry << 1)		ldr32.h rz, (rx, ry << 0)
ldr.h rz, (rx, ry << 2)	RZ \leftarrow zero_extend(MEM[RX + RY << IMM2])	ldr32.h rz, (rx, ry << 1)
ldr.h rz, (rx, ry << 3)		ldr32.h rz, (rx, ry << 2)
		ldr32.h rz, (rx, ry << 3)

Description: Save half-word loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit

instruction

Operation: Load half-word from storage to register, extend unsigned

$$RZ \leftarrow \text{zero_extend}(\text{MEM}[RX + RY << IMM2])$$

Grammar:
 ldr32.h rz, (rx, ry << 0)
 ldr32.h rz, (rx, ry << 1)
 ldr32.h rz, (rx, ry << 2)
 ldr32.h rz, (rx, ry << 3)

Description: Save half-word loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to

the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

ldr32.h rz,(rx, ry << 0)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0		RY		RX	0 0 0 0 0 1		0 0 0 0 1		RZ	

ldr32.h rz,(rx, ry << 1)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0		RY		RX	0 0 0 0 0 1		0 0 0 1 0		RZ	

ldr32.h rz,(rx, ry << 2)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0		RY		RX	0 0 0 0 0 1		0 0 1 0 0		RZ	

ldr32.h rz,(rx, ry << 3)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0		RY		RX	0 0 0 0 0 1		0 1 0 0 0		RZ	

LDR.HS – Load signed and extended half-word in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
ldr.hs rz, (rx, ry << 0)	Load half-word from storage to register, extend signed	Only 32-bit instructions exist.
ldr.hs rz, (rx, ry << 1)		ldr32.hs rz, (rx, ry << 0)
ldr.hs rz, (rx, ry << 2)	RZ \leftarrow sign_extend(MEM[RX + RY << IMM2])	ldr32.hs rz, (rx, ry << 1)
ldr.hs rz, (rx, ry << 3)		ldr32.hs rz, (rx, ry << 2)
		ldr32.hs rz, (rx, ry << 3)

Description: Save half-word loaded from storage in register RZ after sign-extension to 32 bits. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit

instruction

Operation: Load half-word from storage to register, extend signed

$$RZ \leftarrow \text{sign_extend}(\text{MEM}[RX + RY << IMM2])$$

Grammar:
 ldr32.hs rz, (rx, ry << 0)
 ldr32.hs rz, (rx, ry << 1)
 ldr32.hs rz, (rx, ry << 2)
 ldr32.hs rz, (rx, ry << 3)

Description: Save half-word loaded from storage in register RZ after sign-extension to 32 bits. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to

the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

ldr32.hs rz, (rx, ry << 0)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0		RY		RX	0 0 0 1 0 1		0 0 0 0 1		RZ	

ldr32.hs rz, (rx, ry << 1)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0		RY		RX	0 0 0 1 0 1		0 0 0 1 0		RZ	

ldr32.hs rz, (rx, ry << 2)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0		RY		RX	0 0 0 1 0 1		0 0 1 0 0		RZ	

ldr32.hs rz, (rx, ry << 3)

31	30	26	25	21	20	16	15	109	5	4	0
1	1 0 1 0 0		RY		RX	0 0 0 1 0 1		0 1 0 0 0		RZ	

LDR.W – Load word in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
ldr.w rz, (rx, ry << 0)	Load word from storage to register	Only 32-bit instructions exist.
ldr.w rz, (rx, ry << 1)	$RZ \leftarrow \text{MEM}[RX + RY << \text{IMM2}]$	
ldr.w rz, (rx, ry << 2)		ldr32.w rz, (rx, ry << 0)
ldr.w rz, (rx, ry << 3)		ldr32.w rz, (rx, ry << 1) ldr32.w rz, (rx, ry << 2) ldr32.w rz, (rx, ry << 3)

Description: Load word from storage to register RZ. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit

instruction

Operation: Load word from storage to register

$RZ \leftarrow \text{MEM}[RX + RY << \text{IMM2}]$

Grammar:

ldr32.w rz, (rx, ry << 0)
ldr32.w rz, (rx, ry << 1)
ldr32.w rz, (rx, ry << 2)
ldr32.w rz, (rx, ry << 3)

Description: Load word from storage to register RZ. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

ldr32.w rz, (rx, ry << 0)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 0 1 0	0 0 0 0 1	RZ						

ldr32.w rz, (rx, ry << 1)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 0 1 0	0 0 0 1 0	RZ						

ldr32.w rz, (rx, ry << 2)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 0 1 0	0 0 1 0 0	RZ						

ldr32.w rz, (rx, ry << 3)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0	RY	RX	0 0 0 0 1 0	0 1 0 0 0	RZ						

LRS.B – Load byte sign

Unified instruction

Grammar	Operation	Compiling result
lrs.b rz, [label]	Load byte from storage to register $RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset})])$	Only 32-bit instructions exist. lrs32.b rz, [label]

Description: Load the byte sign in the place where label is located, and save it in destination register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding base register R28 to the value of unsigned extending the 18-bit relative offset to 32 bits. The address space of LRS.B instruction is +256KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load byte sign from storage to register, extend unsigned
 $RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset})])$

Grammar: lrs32.b rz, [label]

Description: Load the byte sign in the place where label is located, and save it in destination register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding base register R28 to the value of unsigned extending the 18-bit relative offset to 32 bits. The address space of LRS.B instruction is +256KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

**Instruction
format:**

31 30	26 25	21 20	18 17	0
1 1 0 0 1 1	RZ	0 0 0	Offset	

LRS.H – Load half-word sign

Unified instruction

Grammar	Operation	Compiling result
lrs.h rz, [label]	$RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset} \ll 1)])$	Only 32-bit instructions exist. lrs32.h rz, [label]

Description: Load the half-word sign in the place where label is located, and save it in destination register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding base register R28 to the value of unsigned extending the 18-bit relative offset shifting left by 1 bit to 32 bits. The address space of LRS.H instruction is +512KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load half-word sign from storage to register, extend unsigned
 $RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset} \ll 1)])$

Grammar: lrs32.h rz, [label]

Description: Load the half-word sign in the place where label is located, and save it in destination register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding base register R28 to the value of unsigned extending the 18-bit relative offset shifting left by 1 bit to 32 bits. The address space of LRS.H instruction is +512KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction**format:**

3130 2625 21 20 18 17 0

1	1 0 0 1 1	RZ	0 0 1	Offset
---	-----------	----	-------	--------

LRS.W – Load word sign

Unified instruction

Grammar	Operation	Compiling result
lrs.w rz, [label]	$RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset} \ll 2)])$	Only 32-bit instructions exist. lrs32.w rz, [label]

Description: Load the word sign in the place where label is located, and save it in destination register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding base register R28 to the value of unsigned extending the 18-bit relative offset shifting left by 2 bits to 32 bits. The address space of LRS.W instruction is +1024KB.
 Attention: The offset DISP is the offset of binary operand.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load word sign from storage to register, extend unsigned
 $RZ \leftarrow \text{zero_extend}(\text{MEM}[R28 + \text{zero_extend}(\text{offset} \ll 2)])$

Grammar: lrs32.w rz, [label]

Description: Load the word sign in the place where label is located, and save it in destination register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding base register R28 to the value of unsigned extending the 18-bit relative offset shifting left by 2 bits to 32 bits. The address space of LRS.W instruction is +1024KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction**format:**

31 30 26 25 21 20 18 17 0

1	1 0 0 1 1	RZ	0 1 0	Offset
---	-----------	----	-------	--------

LRW – Memory read-in

**Unified
instruction**

Grammar	Operation	Compiling result
lrw rz, label	Load word from storage to register	Compiled into corresponding 16-bit or 32-bit instructions according to the range of load
lrw rz, imm32	$RZ \leftarrow \text{zero_extend}(\text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffffc])$	if($\text{offset} < 1020B$), then lrw16 label; lrw16 imm32; else lrw32 label; lrw32 imm32;

Description: Load the word in the place where label is located or 32-bit immediate operand (IMM32) to destination register RZ. The storage address is gained by adding PC to the relative offset shifting left by 2 bits, unsigned extending it to 32 bits, and compulsively clearing the two lowest bits. The load range of LRW instruction is the whole address space of 4GB.

Influence on flag bit: No influence

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

**16-bit
instruction----1**

Operation: Load word from storage to register
 $RZ \leftarrow \text{zero_extend}(\text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffffc])$

Grammar: lrw16 rz, label
 lrw16 rz, imm32

Description: Load the word in the place where label is located or 32-bit immediate operand (IMM32) to destination register RZ. The storage address is

gained by adding PC to the 8-bit relative offset shifting left by 2 bits, unsigned extending it to 32 bits, and compulsively clearing the two lowest bits. The load range of LRW instruction is the whole address space of 4GB.

Attention: The relative offset is equal to the binary code $\{1, \sim\{\text{IMM2}, \text{IMM5}\}\}$.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7; the range of relative offset is 0x80-0xfe.

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

15	14	11	10	9	8	7	5	4	0
0	0	0	0	0	0	IMM2	RZ	IMM5	

16-bit

instruction----2

Operation: Load word from storage to register

$RZ \leftarrow \text{zero_extend}(\text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffffc])$

Grammar: lrw16 rz, label

lrw16 rz, imm32

Description: Load the word in the place where label is located or 32-bit immediate operand (IMM32) to destination register RZ. The storage address is gained by adding PC to the 8-bit relative offset shifting left by 2 bits, unsigned extending it to 32 bits, and compulsively clearing the two lowest bits. The load range of LRW instruction is the whole address space of 4GB.

Attention: The relative offset is equal to the binary code $\{0, \{\text{IMM2}, \text{IMM5}\}\}$.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7; the range of relative offset is 0x0-0x7f.

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

15 14	11 10 9 8 7	5 4	0
0 0 0 1 0 0 IMM2 RZ IMM5			

32-bit

instruction

Operation: Load word from storage to register

$$RZ \leftarrow \text{zero_extend}(\text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffff])$$

Grammar: lrw32 rz, label

lrw32 rz, imm32

Description: Load the word in the place where label is located or 32-bit immediate operand (IMM32) to destination register RZ. The storage address is gained by adding PC to the 16-bit relative offset shifting left by 2 bits, unsigned extending it to 32 bits, and compulsively clearing the two lowest bits. The load range of LRW instruction is the whole address space of 4GB.

Influence on flag No influence

bit:

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0 1 0 1 0 0 RZ Offset				

LSL – Logical shift left

**Unified
instruction**

Grammar	Operation	Compiling result
lsl rz, rx	$RZ \leftarrow RZ << RX[5:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then lsl16 rz, rx; else lsl32 rz, rx, rx;
lsl rz, rx, ry	$RZ \leftarrow RX << RY[5:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x == z$) and ($y < 16$) and ($z < 16$), then lsl16 rz, ry else lsl32 rz, rx, ry

Description: For lsl rz, rx, perform a logical left shift on RZ value (the original value shifts left and 0 will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 31, RZ will be cleared;
For lsl rz, rx, ry, perform a logical left shift on RX value (the original value shifts left and 0 will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RY (RY[5:0]). If the value of RY[5:0] is greater than 31, RZ will be cleared.

Influence on flag No influence

bit:

Exception: None

**16-bit
instruction**

Operation: $RZ \leftarrow RZ << RX[5:0]$

Grammar:	lsl16 rz, rx
Description:	Perform a logical left shift on RZ value (the original value shifts left and 0 will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 31, RZ will be cleared.
Influence on flag	No influence
bit:	
Restriction:	The range of register is r0-r15.
Exception:	None
Instruction	
format:	

15 14	10 9	6 5	2 1 0
0 1 1 1 0 0	RZ	RX	0 0

32-bit instruction

Operation:	$RZ \leftarrow RX \ll RY[5:0]$
Grammar:	lsl32 rz, rx, ry
Description:	Perform a logical left shift on RX value (the original value shifts left and 0 will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RY (RY[5:0]). If the value of RY[5:0] is greater than 31, RZ will be cleared.
Influence on flag	No influence
bit:	
Exception:	None
Instruction	
format:	

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 0 0 0 1	RZ	

LSLC – Logical shift left immediate to C

Unified instruction

Grammar	Operation	Compiling result
lslc rz, rx, oimm5	$RZ \leftarrow RX \ll OIMM5, C \leftarrow RX[32 - OIMM5]$	Only 32-bit instructions exist. lslc32 rz, rx, oimm5

Description: Perform a logical left shift on RX value (the original value shifts left and 0 will shift to the right side), save the end bit shifting out in condition bit C, and save the shifting result in RZ; the range of left shift is decided by the value of 5-bit immediate operand with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, then the condition bit C is the lowest bit of RX and RZ will be cleared.

Influence on flag bit: $C \leftarrow RX[32 - OIMM5]$

Restriction: The range of immediate operand is 1-32.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow RX \ll OIMM5, C \leftarrow RX[32 - OIMM5]$

Grammar: lslc32 rz, rx, oimm5

Description: Perform a logical left shift on RX value (the original value shifts left and 0 will shift to the right side), save the end bit shifting out in condition bit C, and save the shifting result in RZ; the range of left shift is decided by the value of 5-bit immediate operand with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, then the condition bit C is the lowest bit of RX and RZ will be cleared.

Attention: The binary operand IMM5 is equal to OIMM5 – 1.

Influence on flag bit: $C \leftarrow RX[32 - OIMM5]$

Restriction: The range of immediate operand is 1-32.

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 1	0 0 0 0 1	RZ

IMM5 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM5, the shifting value OIMM5 requires offset 1.

00000 – shift by 1 bit

00001 – shift by 2 bits

.....

11111 – shift by 32 bits

LSLI – Logical shift left immediate

Unified instruction

Grammar	Operation	Compiling result
lsli rz, rx, imm5	$RZ \leftarrow RX \ll IMM5$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 8$) and ($z < 8$), then lsli16 rz, rx, imm5 else lsli32 rz, rx, imm5

- Description:** Perform a logical left shift on RX value (the original value shifts left and 0 will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value remains unchanged.
- Influence on flag bit:** No influence
- Restriction:** The range of immediate operand is 1-31.
- Exception:** None

16-bit instruction

- Operation:** $RZ \leftarrow RX \ll IMM5$
- Grammar:** lsli16 rz, rx, imm5
- Description:** Perform a logical left shift on RX value (the original value shifts left and 0 will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value remains unchanged.
- Influence on flag bit:** No influence
- Restriction:** The range of register is r0-r7;
The range of immediate operand is 1-31.
- Exception:** None

Instruction

format:

15 14 11 10 8 7 5 4 0

0	1 0 0 0	RX	RZ	IMM5
---	---------	----	----	------

32-bit instruction

Operation: $RZ \leftarrow RX \ll IMM5$

Grammar: lsli32 rz, rx, imm5

Description: Perform a logical left shift on RX value (the original value shifts left and 0 will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value is the same with RX value.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 1-31.

Exception: None

Instruction

format:

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 0	0 0 0 0 1	RZ
---	-----------	------	----	-------------	-----------	----

LSR – Logical shift right

Unified instruction

Grammar	Operation	Compiling result
lsr rz, rx	$RZ \leftarrow RZ >> RX[5:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 16$) and ($x < 16$), then lsr16 rz, rx; else lsr32 rz, rx;
lsr rz, rx, ry	$RZ \leftarrow RX >> RY[5:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x == z$) and ($z < 16$) and ($y < 16$), then lsr16 rz, ry; else lsr32 rz, rx, ry;

Description: For lsr rz, rx, perform a logical right shift on RZ value (the original value shifts right and 0 will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 31, RZ will be cleared;
For lsr rz, rx, ry, perform a logical right shift on RX value (the original value shifts right and 0 will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RY (RY[5:0]). If the value of RY[5:0] is greater than 31, RZ will be cleared.

Influence on flag No influence

bit:

Exception: None

16-bit instruction

Operation: $RZ \leftarrow RZ >> RX[5:0]$

Grammar: lsr16 rz, rx

Description: Perform a logical right shift on RZ value (the original value shifts right and 0 will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 31, RZ will be cleared.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 1 0 0	RZ	RX	0 1

32-bit instruction

Operation: $RZ \leftarrow RX >> RY[5:0]$

Grammar: lsr32 rz, rx, ry

Description: Perform a logical right shift on RX value (the original value shifts right and 0 will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RY (RY[5:0]). If the value of RY[5:0] is greater than 31, RZ will be cleared.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 0 0 1 0	RZ	

LSRC – Logical shift right immediate to C

Unified instruction

Grammar	Operation	Compiling result
lsrc rz, rx, oimm5	$RZ \leftarrow RX >> OIMM5,$ $C \leftarrow RX[OIMM5 - 1]$	Only 32-bit instructions exist. lsrc32 rz, rx, oimm5

Description: Perform a logical right shift on RX value (the original value shifts right and 0 will shift to the left side), save the end bit shifting out in condition bit C, and save the shifting result in RZ; the range of right shift is decided by the value of 5-bit immediate operand with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, then the condition bit C is the highest bit of RX and RZ will be cleared.

Influence on flag bit: $C \leftarrow RX[OIMM5 - 1]$

Restriction: The range of immediate operand is 1-32.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow RX >> OIMM5, C \leftarrow RX[OIMM5 - 1]$

Grammar: lsrc32 rz, rx, oimm5

Description: Perform a logical right shift on RX value (the original value shifts right and 0 will shift to the left side), save the end bit shifting out in condition bit C, and save the shifting result in RZ; the range of right shift is decided by the value of 5-bit immediate operand with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, then the condition bit C is the highest bit of RX and RZ will be cleared.

Attention: The binary operand IMM5 is equal to OIMM5 – 1.

Influence on flag bit: $C \leftarrow RX[OIMM5 - 1]$

Restriction: The range of immediate operand is 1-32.

Exception: None

Instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	IMM5	RX	0 1 0 0 1 1	0 0 0 1 0	RZ	

IMM5 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM5, the shifting value OIMM5 requires offset 1.

00000 – shift by 1 bit

00001 – shift by 2 bits

.....

11111 – shift by 32 bits

LSRI – Logical shift right immediate

Unified instruction

Grammar	Operation	Compiling result
lsri rz, rx, imm5	$RZ \leftarrow RX \gg IMM5$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 8$) and ($z < 8$), then lsri16 rz, rx, imm5 else lsri32 rz, rx, imm5

Description: Perform a logical right shift on RX value (the original value shifts right and 0 will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value remains unchanged or is the same with RX value.

Influence on flag bit: No influence

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

16-bit instruction

Operation: $RZ \leftarrow RX \gg IMM5$

Grammar: lsri16 rz, rx, imm5

Description: Perform a logical right shift on RX value (the original value shifts right and 0 will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value remains unchanged.

Influence on flag bit: No influence

bit:

Restriction: The range of register is r0-r7; the range of immediate operand is 1-31.

Exception: None

Instruction

format:

15 14	11 10	8 7	5 4	0
0 0 0 1	RX	RZ	IMM5	

32-bit
instruction

Operation: $RZ \leftarrow RX \gg IMM5$

Grammar: lsri32 rz, rx, imm5

Description: Perform a logical right shift on RX value (the original value shifts right and 0 will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value is the same with RX value.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction
format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	IMM5	RX	0 1 0 0 1 0	0 0 0 1 0	RZ	

MFCR – Read transfer from control register

Unified instruction

Grammar	Operation	Compiling result
mfcr rz, cr<x, sel>	Transfer contents in control register to general-purpose register $RZ \leftarrow CR<X, sel>$	Only 32-bit instructions exist. mfcr32 rz, cr<x, sel>

Attribute: Privileged instruction

Description: Transfer contents in control register $CR<x, sel>$ to general-purpose register RZ .

Influence on flag No influence

bit:

Exception: Privilege violation exception

32-bit instruction

Operation: Transfer contents in control register to general-purpose register
 $RZ \leftarrow CR<X, sel>$

Grammar: mfcr32 rz, cr<x, sel>

Attribute: Privileged instruction

Description: Transfer contents in control register $CR<x, sel>$ to general-purpose register RZ .

Influence on flag No influence

bit:

Exception: Privilege violation exception

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	0	sel	CRX	0	1	1	0	0

MFHI – Read transfer from high bit of accumulator

**Unified
instruction**

Grammar	Operation	Compiling result
mfhi rz	Transfer contents in high-bit accumulator register to general-purpose register RZ ← HI	Only 32-bit instructions exist. mfhi32 rz

Description: Transfer contents in the 32-high-bit register HI of 64-bit accumulator to general-purpose register RZ.

Influence on flag bit: No influence

Exception: None

32-bit

instruction

Operation: Transfer contents in high-bit accumulator register to general-purpose register

RZ ← HI

Grammar: mfhi32 rz

Description: Transfer contents in the 32-high-bit register HI of 64-bit accumulator to general-purpose register RZ.

Influence on flag bit: No influence

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	0	0	0	0	1	1	1

MFHIS – Read transfer saturate from high bit of accumulator

Unified instruction

Grammar	Operation	Compiling result
mfhis rz	<p>Transfer contents in high-bit accumulator register to general-purpose register after getting the saturation value</p> <p>$RZ \leftarrow \text{saturate(HI)}$</p>	<p>Only 32-bit instructions exist.</p> <p>mfhis32 rz</p>

Description: Transfer contents in the 32-high-bit register HI of 64-bit accumulator to general-purpose register RZ after getting the saturation value. See the descriptions about guard bit in the processor manual for details about saturation operation.

Influence on flag bit: No influence

Exception: None

32-bit instruction

Operation: Transfer contents in high-bit accumulator register to general-purpose register after getting the saturation value

$RZ \leftarrow \text{saturate(HI)}$

Grammar: mfhis32 rz

Description: Transfer contents in the 32-high-bit register HI of 64-bit accumulator to general-purpose register RZ after getting the saturation value. See the descriptions about guard bit in the processor manual for details about saturation operation.

Influence on flag bit: No influence

Exception: None

Instruction format:

31 30	2625	2120	16 15	109	5 4	0
1	1 0 0 0 1	0 0 0 0 0	0 0 0 0 0	1 0 0 1 1 0	0 0 0 0 1	RZ

CSKY

MFL0 – Read transfer from low bit of accumulator

Unified instruction

Grammar	Operation	Compiling result
mflo rz	Transfer contents in low-bit accumulator register to general-purpose register $RZ \leftarrow LO$	Only 32-bit instructions exist. mflo32 rz

Description: Transfer contents in the 32-low-bit register LO of 64-bit accumulator to general-purpose register RZ.

Influence on flag bit: No influence

Exception: None

32-bit

instruction

Operation: Transfer contents in low-bit accumulator register to general-purpose register

$RZ \leftarrow LO$

Grammar: mflo32 rz

Description: Transfer contents in the 32-low-bit register LO of 64-bit accumulator to general-purpose register RZ.

Influence on flag bit: No influence

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	0	0	0	0	1	0	0

MFLOS – Read transfer saturate from low bit of accumulator

Unified instruction

Grammar	Operation	Compiling result
mflos rz	<p>Transfer contents in low-bit accumulator register to general-purpose register after getting the saturation value</p> $RZ \leftarrow \text{saturate}(LO)$	<p>Only 32-bit instructions exist.</p> <p>mflos32 rz</p>

Description: Transfer contents in the 32-low-bit register LO of 64-bit accumulator to general-purpose register RZ after getting the saturation value. See the descriptions about guard bit in the processor manual for details about saturation operation.

Influence on flag bit: No influence

Exception: None

32-bit instruction

Operation: Transfer contents in low-bit accumulator register to general-purpose register after getting the saturation value

$$RZ \leftarrow \text{saturate}(LO)$$

Grammar: mflos32 rz

Description: Transfer contents in the 32-low-bit register LO of 64-bit accumulator to general-purpose register RZ after getting the saturation value. See the descriptions about guard bit in the processor manual for details about saturation operation.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31 30	2625	2120	16 15	109	5 4	0
1	1 0 0 0 1	0 0 0 0 0	0 0 0 0 0	1 0 0 1 1 0	0 0 1 0 0	RZ

CSKY

MOV – Move#

Unified

instruction

Grammar	Operation	Compiling result
mov rz, rx	RZ \leftarrow RX	Always compiled into 16-bit instruction. mov16 rz, rx

Description: Copy the value in RX to destination register RZ.

Influence on flag No influence

bit:

Exception: None

16-bit

instruction

Operation: RZ \leftarrow RX

Grammar: mov16 rz, rx

Description: Copy the value in RX to destination register RZ.

Attention: The register index range of this instruction is r0-r31.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

15	14	10	9	6	5	2	1	0	
0	1	1	0	1	1	RZ	RX	1	1

32-bit

instruction

Operation: RZ \leftarrow RX

Grammar: mov32 rz, rx

Description: Copy the value in RX to destination register RZ.

Attention: This instruction is the pseudo instruction of lsli32 rz, rx, 0x0.

Influence on No influence

flag bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 0 0 0 0 0 RX 0 1 0 0 1 0 0 0 0 0 1 RZ						

MOVF – C=0 move#

**Unified
instruction**

Grammar	Operation	Compiling result
movf rz, rx	if C==0, then RZ ← RX; else RZ ← RZ;	Only 32-bit instructions exist. movf32 rz, rx

Description: If C is 0, copy the value of RX to destination register RZ. Otherwise, keep the value of RZ unchanged.

Attention: This instruction is the pseudo instruction of incf rz, rx, 0x0.

Influence on flag bit: No influence

Exception: None

**32-bit
instruction**

Operation: if C==0, then
 RZ ← RX;
else
 RZ ← RZ;

Grammar: movf32 rz, rx

Description: If C is 0, copy the value of RX to destination register RZ. Otherwise, keep the value of RZ unchanged.

Attention: This instruction is the pseudo instruction of incf32 rz, rx, 0x0.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 0 1	0 0 0 0 0
---	-----------	----	----	-------------	-----------	-----------



MOVI – Move immediate

Unified instruction

Grammar	Operation	Compiling result
movi16 rz, imm16	$RZ \leftarrow \text{zero_extend(}IMM16\text{)};$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of immediate operand and register. if ($imm16 < 256$) and ($z < 7$), then movi16 rz, imm8; else movi32 rz, imm16;

Description: Zero-extend the 16-bit immediate operand to 32 bits, and transfer it to destination register RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

16-bit instruction

Operation: $RZ \leftarrow \text{zero_extend(}IMM8\text{)};$

Grammar: movi16 rz, imm8

Description: Zero-extend the 8-bit immediate operand to 32 bits, and transfer it to destination register RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r7; the range of immediate operand is 0-255.

Exception: None



32-bit instruction

Operation: $RZ \leftarrow \text{zero_extend(}IMM16\text{)};$

Grammar: movi32 rz, imm16

Description: Zero-extend the 16-bit immediate operand to 32 bits, and transfer it to destination register RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	1 0 0 0 0	RZ		IMM16

MOVIH – Move immediate high

Unified instruction

Grammar	Operation	Compiling result
movih rz, imm16	$RZ \leftarrow \text{zero_extend}(\text{IMM16}) \ll 16$	Only 32-bit instructions exist. movih32 rz, imm16

Description: Zero-extend the 16-bit immediate operand to 32 bits, perform a logical left shift by 16 bits, and transfer the result to destination register RZ. This instruction can generate any 32-bit immediate operand by cooperating with ori rz, rz, imm16 instruction.

**Influence on flag
bit:** No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow \text{zero_extend}(\text{IMM16}) \ll 16$

Grammar: movih32 rz, imm16

Description: Zero-extend the 16-bit immediate operand to 32 bits, perform a logical left shift by 16 bits, and transfer the result to destination register RZ. This instruction can generate any 32-bit immediate operand by cooperating with ori32 rz, rz, imm16 instruction.

**Influence on flag
bit:** No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	0
1	1 1 0 1 0	1	0 0 0 1	RZ		IMM16		

MOVT – C=1 move#

**Unified
instruction**

Grammar	Operation	Compiling result
movt rz, rx	if C==1, then RZ ← RX; else RZ ← RZ;	Only 32-bit instructions exist. movt32 rz, rx

Description: If C is 1, copy the value of RX to destination register RZ. Otherwise, keep the value of RZ unchanged.

Attention: This instruction is the pseudo instruction of inct rz, rx, 0x0.

Influence on flag bit: No influence

Exception: None

**32-bit
instruction**

Operation: if C==1, then
 RZ ← RX;
else
 RZ ← RZ;

Grammar: movt32 rz, rx

Description: If C is 1, copy the value of RX to destination register RZ. Otherwise, keep the value of RZ unchanged.

Attention: This instruction is the pseudo instruction of inct32 rz, rx, 0x0.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 1 0	0 0 0 0 0
---	-----------	----	----	-------------	-----------	-----------



MTCR – Write transfer to control register

Unified instruction

Grammar	Operation	Compiling result
mtc r rx, cr<z, sel>	Transfer contents in general-purpose register to control register CR<Z, sel> ← RX	Only 32-bit instructions exist. mtcr32 rx, cr<z, sel>

Attribute: Privileged instruction

Description: Transfer contents in general-purpose register RX to control register CR<z, sel>.

Influence on flag bit: If the target control register is not PSR, this instruction will not affect the flag bit.

Exception: Privilege violation exception

32-bit instruction

Operation: Transfer contents in general-purpose register to control register CR<Z, sel> ← RX

Grammar: mtcr32 rx, cr<z, sel>

Attribute: Privileged instruction

Description: Transfer contents in general-purpose register RX to control register CR<z, sel>.

Influence on flag bit: If the target control register is not PSR, this instruction will not affect the flag bit.

Exception: Privilege violation exception

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0	sel	RX	0 1 1 0 0 1	0 0 0 0 1	CRZ	

MTHI – Write transfer to high bit of accumulator

**Unified
instruction**

Grammar	Operation	Compiling result
mthi rx	Transfer contents in general-purpose register to high-bit accumulator register HI ← RX	Only 32-bit instructions exist. mthi32 rx

Description: Transfer contents in general-purpose register RX to the 32-high-bit register HI of 64-bit accumulator.

Influence on flag bit: No influence

Exception: None

**32-bit
instruction**

Operation: Transfer contents in general-purpose register to high-bit accumulator register
HI ← RX

Grammar: mthi32 rx

Description: Transfer contents in general-purpose register RX to the 32-high-bit register HI of 64-bit accumulator.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	0	0	0	0	0	1	0

MTLO – Write transfer to low bit of accumulator

**Unified
instruction**

Grammar	Operation	Compiling result
mtlo rx	Transfer contents in general-purpose register to low-bit accumulator register LO ← RX	Only 32-bit instructions exist. mtlo32 rx

Description: Transfer contents in general-purpose register RX to the 32-low-bit register LO of 64-bit accumulator.

Influence on flag bit: No influence

Exception: None

**32-bit
instruction**

Operation: Transfer contents in general-purpose register to low-bit accumulator register

LO ← RX

Grammar: mtlo32 rx

Description: Transfer contents in general-purpose register RX to the 32-low-bit register LO of 64-bit accumulator.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	0	0	0	0	0	1	0

MULS – Multiply signed

Unified instruction

Grammar	Operation	Compiling result
muls rx, ry	Multiply two signed numbers and put the result in accumulator $\{HI, LO\} \leftarrow RX \times RY$	Only 32-bit instructions exist. muls32 rx, ry

Description: Multiply the contents in general-purpose register RX and RY, and put the result in the 64-bit accumulator. The 32 high bits are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as signed numbers.

Influence on flag bit: Overflow bit is cleared

Exception: None

32-bit

instruction

Operation: Multiply two signed numbers and put the result in accumulator
 $\{HI, LO\} \leftarrow RX \times RY$

Grammar: muls32 rx, ry

Description: Multiply the contents in general-purpose register RX and RY, and put the result in the 64-bit accumulator. The 32 high bits are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as signed numbers.

Influence on flag bit: Overflow bit is cleared

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 RY RX 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0						

MULSA – Multiply-accumulate signed

Unified instruction

Grammar	Operation	Compiling result
mulsa rx, ry	Multiply two signed numbers, add the product to value in accumulator, and put the result in accumulator $\{HI, LO\} \leftarrow \{HI, LO\} + RX \times RY$	Only 32-bit instructions exist. mulsa32 rx, ry

Description: Multiply the contents in general-purpose register RX and RY, add the product to value in the 64-bit accumulator, and put the result in the accumulator. The 32 high bits of the result are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as signed numbers.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

32-bit instruction

Operation: Multiply two signed numbers, add the product to value in accumulator, and put the result in accumulator
 $\{HI, LO\} \leftarrow \{HI, LO\} + RX \times RY$

Grammar: mulsa32 rx, ry

Description: Multiply the contents in general-purpose register RX and RY, add the product to value in the 64-bit accumulator, and put the result in the accumulator. The 32 high bits of the result are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as signed numbers.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

**Instruction
format:**

3130	2625	2120	1615	109	54	0
1 1 0 0 0 1	RY	RX	1 0 0 0 1 1	0 0 0 1 0	0 0 0 0 0	



MULSH – 16-bit multiply signed

**Unified
instruction**

Grammar	Operation	Compiling result
mulsh rz, rx	Multiply two 16-bit signed numbers and put the result in general-purpose register $RZ \leftarrow RX[15:0] \times RZ[15:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($y < 16$), then mulsh16 rz, rx; else mulsh32 rz, rx;
mulsh rz, rx, ry	Multiply two 16-bit signed numbers and put the result in general-purpose register $RZ \leftarrow RX[15:0] \times RY[15:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($y == z$) and ($x < 16$) and ($z < 16$), then mulsh16 rz, rx; else mulsh32 rz, rx, ry;

Description: Multiply the 16 low bits of general-purpose register RX and 16 low bits of RZ/RY, and put the result in general-purpose register RZ. All contents in the registers are considered as signed numbers. The sign bit of source register is the 15th bit and the sign bit of destination register is the 31st bit.

Influence on flag bit: No influence

Exception: None

**16-bit
instruction**

Operation: Multiply two 16-bit signed numbers and put the result in general-purpose register

$$RZ \leftarrow RX[15:0] \times RZ[15:0]$$

Grammar: mulsh16 rz, rx

Description: Multiply the 16 low bits of general-purpose register RX and 16 low bits

of RZ, and put the 32-bit result in general-purpose register RZ. All contents in register RX and register RZ are considered as signed numbers. The sign bit of source operand is the 15th bit and the sign bit of result is the 31st bit.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 1 1 1 RZ RX 0 1			

32-bit

instruction

Operation: Multiply two 16-bit signed numbers and put the result in general-purpose register

$$RZ \leftarrow RX[15:0] \times RY[15:0]$$

Grammar: mulsh32 rz, rx, ry

Description: Multiply the 16 low bits of general-purpose register RX and 16 low bits of RY, and put the result in general-purpose register RZ. All contents in register RX, RY and RZ are considered as signed numbers. The sign bit of source register RX and RY is the 15th bit, and the sign bit of destination register RZ is the 31st bit.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 RY RX 1 0 0 1 0 0 0 0 0 0 1 RZ						

MULSHA – 16-bit multiply-accumulate signed

Unified instruction

Grammar	Operation	Compiling result
mulsha rx, ry	<p>Multiply two 16-bit signed numbers, add the product to low-bit value in accumulator, and put the result in low bit of accumulator</p> $LO \leftarrow LO + RX[15:0] \times RY[15:0]$	<p>Only 32-bit instructions exist.</p> <p>mulsha32 rx, ry</p>

Description: Multiply the 16 low bits of general-purpose register RX and 16 low bits of RY, add the 32-bit result to the value in 32-low-bit register LO of 64-bit accumulator, and put the result in low-bit accumulator register LO. All contents in register RX and RY as well as low-bit accumulator register LO are considered as signed numbers. The sign bit of source operand in register RX and RY is the 15th bit and the sign bit of source operand and result in low-bit accumulator register LO is the 31st bit. This instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

32-bit instruction

Operation: Multiply two 16-bit signed numbers, add the product to low-bit value in accumulator, and put the result in low bit of accumulator

$$LO \leftarrow LO + RX[15:0] \times RY[15:0]$$

Grammar: mulsha32 rx, ry

Description: Multiply the 16 low bits of general-purpose register RX and 16 low bits of RY, add the 32-bit result to the value in 32-low-bit register LO of 64-bit accumulator, and put the result in low-bit accumulator register LO. All contents in register RX and RY as well as low-bit accumulator register LO are considered as signed numbers. The sign bit of source operand in register RX and RY is the 15th bit and the sign bit of source

operand and result in low-bit accumulator register LO is the 31st bit.

This instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag Due to the overflow, the overflow bit will be set as 1
bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	1 0 0 1 0 0	0 0 0 1 0	0 0 0 0 0	

MULSHS – 16-bit multiply-subtract signed

Unified instruction

Grammar	Operation	Compiling result
mulshs rx, ry	<p>Subtract the value after multiplying two 16-bit signed numbers from the low-bit value in accumulator, and put the result in low bit of accumulator</p> $LO \leftarrow LO - RX[15:0] \times RY[15:0]$	<p>Only 32-bit instructions exist.</p> <p>mulshs32 rx, ry</p>

Description: Subtract the value after multiplying the 16 low bits of general-purpose register RX and 16 low bits of RY from the value in 32-low-bit register LO of 64-bit accumulator, and put the result in low-bit accumulator register LO. All contents in register RX and RY as well as low-bit accumulator register LO are considered as signed numbers. The sign bit of source operand in register RX and RY is the 15th bit and the sign bit of source operand and result in low-bit accumulator register LO is the 31st bit.

This instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

32-bit instruction

Operation: Subtract the value after multiplying two 16-bit signed numbers from the low-bit value in accumulator, and put the result in low bit of accumulator

$$LO \leftarrow LO - RX[15:0] \times RY[15:0]$$

Grammar: mulshs32 rx, ry

Description: Subtract the value after multiplying the 16 low bits of general-purpose register RX and 16 low bits of RY from the value in 32-low-bit register LO of 64-bit accumulator, and put the result in low-bit accumulator register LO. All contents in register RX and RY as well as low-bit

accumulator register LO are considered as signed numbers. The sign bit of source operand in register RX and RY is the 15th bit and the sign bit of source operand and result in low-bit accumulator register LO is the 31st bit.

This instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag Due to the overflow, the overflow bit will be set as 1 bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	1 0 0 1 0 0	0 0 1 0 0	0 0 0 0 0	

MULSS – Multiply-subtract signed

Unified instruction

Grammar	Operation	Compiling result
mulss rx, ry	<p>Subtract the value after multiplying two signed numbers from the value in accumulator, and put the result in accumulator</p> $\{HI, LO\} \leftarrow \{HI, LO\} - RX \times RY$	<p>Only 32-bit instructions exist.</p> <p>mulss32 rx, ry</p>

Description: Subtract the value after multiplying general-purpose register RX and RY from the value in 64-bit accumulator, and put the result in accumulator. The 32 high bits of the result are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as signed numbers.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

32-bit instruction

Operation: Subtract the value after multiplying two signed numbers from the value in accumulator, and put the result in accumulator

$$\{HI, LO\} \leftarrow \{HI, LO\} - RX \times RY$$

Grammar: mulss32 rx, ry

Description: Subtract the value after multiplying general-purpose register RX and RY from the value in 64-bit accumulator, and put the result in accumulator. The 32 high bits of the result are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as signed numbers.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

Instruction

format:

3130	2625	2120	16 15	109	5 4	0
1	1 0 0 0 1	RY	RX	1 0 0 0 1 1	0 0 1 0 0	0 0 0 0 0

MULSW – 16x32 multiply signed

Unified instruction

Grammar	Operation	Compiling result
mulsw rz, rx, ry	Multiply 16-bit signed number and 32-bit signed number, and put the 32 high bits of result in general-purpose register $RZ \leftarrow (RX[15:0] \times RY[31:0])[47:16]$	Only 32-bit instructions exist. mulsw32 rx, ry

Description: Multiply the 16 low bits of general-purpose register RX and contents of register RY, and put the 32 high bits of result in general-purpose register RZ. All values in general-purpose register RX and RY as well as RZ are considered as signed numbers. The sign bit of source operand in register RX is the 15th bit of register and the sign bit of source operand in register RY and result in register RZ is the 31st bit of register.

Influence on flag bit: No influence

Exception: None

32-bit instruction

Operation: Multiply 16-bit signed number and 32-bit signed number, and put the 32 high bits of result in general-purpose register
 $RZ \leftarrow (RX[15:0] \times RY[31:0])[47:16]$

Grammar: mulsw32 rx, ry

Description: Multiply the 16 low bits of general-purpose register RX and contents of register RY, and put the 32 high bits of result in general-purpose register RZ. All values in general-purpose register RX and RY as well as RZ are considered as signed numbers. The sign bit of source operand in register RX is the 15th bit of register and the sign bit of source operand in register RY and result in register RZ is the 31st bit of register.

Influence on flag bit: No influence

Exception: None

Instruction**format:**

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 0 0 1	RY	RX	1 0 0 1 0 1	0 0 0 0 1	RZ
---	-----------	----	----	-------------	-----------	----

MULSWA – 16x32 multiply-accumulate signed

Unified instruction

Grammar	Operation	Compiling result
mulswa rx, ry	Multiply 16-bit signed number and 32-bit signed number, add the 32 high bits of result to the value of low bit in accumulator, and put the result in low bit of accumulator $LO \leftarrow LO + (RX[15:0] \times RY[31:0])[47:16]$	Only 32-bit instructions exist. mulswa32 rx, ry

Description: Multiply the 16 low bits of general-purpose register RX and contents of RY, add the 32 high bits of result to the value in 32-low-bit register LO of 64-bit accumulator, and put the result in low-bit accumulator register LO. All values in register RX and RY as well as low-bit accumulator register LO are considered as signed numbers. The sign bit of source operand in register RX is the 15th bit of register and the sign bit of source operand and result in register RY and low-bit accumulator register LO is the 31st bit of register. This instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

32-bit instruction

Operation: Multiply 16-bit signed number and 32-bit signed number, add the 32 high bits of result to the value of low bit in accumulator, and put the result in low bit of accumulator

$$LO \leftarrow LO + (RX[15:0] \times RY[31:0])[47:16]$$

mulswa32 rx, ry

Description: Multiply the 16 low bits of general-purpose register RX and contents of RY, add the 32 high bits of result to the value in 32-low-bit register LO of 64-bit accumulator, and put the result in low-bit accumulator register LO.

All values in register RX and RY as well as low-bit accumulator register LO are considered as signed numbers. The sign bit of source operand in register RX is the 15th bit of register and the sign bit of source operand and result in register RY and low-bit accumulator register LO is the 31st bit of register.

This instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

Instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	1 0 0 1 0 1	0 0 1 0 0	0 0 0 0 0	

MULSWS – 16x32 multiply-subtract signed

Unified instruction

Grammar	Operation	Compiling result
mulsws rx, ry	<p>Subtract the 32 high bits of the result after multiplying 16-bit signed number and 32-bit signed number from the value of low bit in accumulator, and put the result in low bit of accumulator</p> $LO \leftarrow LO - (RX[15:0] \times RY[31:0])[47:16]$	<p>Only 32-bit instructions exist.</p> <p>mulsws32 rx, ry</p>

Description: Subtract the 32 high bits of the result after multiplying the 16 low bits of general-purpose register RX and contents of RY from the value in 32-low-bit register LO of 64-bit accumulator, and put the result in low-bit accumulator register LO. All contents in register RX and RY as well as low-bit accumulator register LO are considered as signed numbers. The sign bit of source operand in register RX is the 15th bit of register and the sign bit of source operand and result in register RY and low-bit accumulator register LO is the 31st bit of register.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

32-bit instruction

Operation: Subtract the 32 high bits of the result after multiplying 16-bit signed number and 32-bit signed number from the value of low bit in accumulator, and put the result in low bit of accumulator

$$LO \leftarrow LO - (RX[15:0] \times RY[31:0])[47:16]$$

Grammar: mulsws32 rx, ry

Description: Subtract the 32 high bits of the result after multiplying the 16 low bits of general-purpose register RX and contents of RY from the value in 32-low-bit register LO of 64-bit accumulator, and put the result in low-bit accumulator register LO. All contents in register RX and RY as well as

low-bit accumulator register LO are considered as signed numbers. The sign bit of source operand in register RX is the 15th bit of register and the sign bit of source operand and result in register RY and low-bit accumulator register LO is the 31st bit of register.

Influence on flag Due to the overflow, the overflow bit will be set as 1
bit:

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	1 0 0 1 0 1	0 1 0 0 0	0 0 0 0 0	

MULT – Multiply

Unified instruction

Grammar	Operation	Compiling result
mult rz, rx	Multiply two numbers, and put the 32 low bits of the result in general-purpose register $RZ \leftarrow RX \times RZ$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then mult16 rz, rx; else mult32 rz, rx;
mult rz, rx, ry	Multiply two numbers, and put the 32 low bits of the result in general-purpose register $RZ \leftarrow RX \times RY$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($y == z$) and ($x < 16$) and ($z < 16$), then mult16 rz, rx; else mult32 rz, rx, ry;

Description: Multiply the contents of two source registers, put the 32 low bits of the result in destination register, and abandon the 32 high bits of the result. The result is the same no matter whether the source operand is considered as signed number or unsigned number.

Influence on flag bit: No influence

Exception: None

16-bit instruction

Operation: Multiply two numbers, and put the 32 low bits of the result in general-purpose register
 $RZ \leftarrow RX \times RZ$

Grammar: mult16 rz, rx

Description: Multiply the contents of general-purpose register RX and RZ, put the 32

low bits of the result in general-purpose register RZ, and abandon the 32 high bits of the result. The result is the same no matter whether the source operand is considered as signed number or unsigned number.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15	14	10	9	6	5	2	1	0
0	1 1 1 1 1	RZ	RX	0 0				

32-bit

instruction

Operation: Multiply two numbers, and put the 32 low bits of the result in general-purpose register

$$RZ \leftarrow RX \times RY$$

Grammar: mult32 rz, rx, ry

Description: Multiply the contents of general-purpose register RX and RY, put the 32 low bits of the result in general-purpose register RZ, and abandon the 32 high bits of the result. The result is the same no matter whether the source operand is considered as signed number or unsigned number.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1	RY	RX	1 0 0 0 0 1	0 0 0 0 1	RZ						

MULU – Multiply unsigned

Unified instruction

Grammar	Operation	Compiling result
mulu rx, ry	Multiply two unsigned numbers, and put the result in accumulator $\{HI, LO\} \leftarrow RX \times RY$	Only 32-bit instructions exist. mulu32 rx, ry

Description: Multiply the contents in general-purpose register RX and RY, and put the result in the 64-bit accumulator. The 32 high bits are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as unsigned numbers.

Influence on flag bit: Overflow bit is cleared

Exception: None

32-bit instruction

Operation: Multiply two unsigned numbers, and put the result in accumulator
 $\{HI, LO\} \leftarrow RX \times RY$

Grammar: mulu32 rx, ry

Description: Multiply the contents in general-purpose register RX and RY, and put the result in the 64-bit accumulator. The 32 high bits are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as unsigned numbers.

Influence on flag bit: Overflow bit is cleared

Exception: None

Instruction format:

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 1	RY	RX	1 0 0 0 1 0	0 0 0 0 1	0 0 0 0 0
---	-----------	----	----	-------------	-----------	-----------

CSKY 中文

MULUA – Multiply-accumulate unsigned

Unified instruction

Grammar	Operation	Compiling result
mulua rx, ry	Multiply two unsigned numbers, add the product to value in accumulator, and put the result in accumulator $\{HI, LO\} \leftarrow \{HI, LO\} + RX \times RY$	Only 32-bit instructions exist. mulua32 rx, ry

Description: Multiply the contents in general-purpose register RX and RY, add the product to value in the 64-bit accumulator, and put the result in accumulator. The 32 high bits of the result are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as unsigned numbers.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

32-bit instruction

Operation: Multiply two unsigned numbers, add the product to value in accumulator, and put the result in accumulator
 $\{HI, LO\} \leftarrow \{HI, LO\} + RX \times RY$

Grammar: mulua32 rx, ry

Description: Multiply the contents in general-purpose register RX and RY, add the product to value in the 64-bit accumulator, and put the result in accumulator. The 32 high bits of the result are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as unsigned numbers.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

Instruction format:

3130	2625	2120	1615	109	54	0
1 1 0 0 0 1	RY	RX	1 0 0 0 1 0	0 0 0 1 0	0 0 0 0 0	



MULUS – Multiply-subtract unsigned

Unified instruction

Grammar	Operation	Compiling result
mulus rx, ry	<p>Subtract the value after multiplying two unsigned numbers from the value in accumulator, and put the result in accumulator</p> $\{HI, LO\} \leftarrow \{HI, LO\} - RX \times RY$	<p>Only 32-bit instructions exist.</p> <p>mulus32 rx, ry</p>

Description: Subtract the value after multiplying general-purpose register RX and RY from the value in 64-bit accumulator, and put the result in accumulator. The 32 high bits of the result are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as unsigned numbers.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

32-bit instruction

Operation: Subtract the value after multiplying two unsigned numbers from the value in accumulator, and put the result in accumulator

$$\{HI, LO\} \leftarrow \{HI, LO\} - RX \times RY$$

Grammar: mulus32 rx, ry

Description: Subtract the value after multiplying general-purpose register RX and RY from the value in 64-bit accumulator, and put the result in accumulator. The 32 high bits of the result are stored in HI and the 32 low bits are stored in LO. All values in general-purpose register RX and RY as well as 64-bit accumulator are considered as unsigned numbers.

Influence on flag bit: Due to the overflow, the overflow bit will be set as 1

Exception: None

Instruction

format:

3130	2625	2120	16 15	109	5 4	0
1	1 0 0 0 1	RY	RX	1 0 0 0 1 0	0 0 1 0 0	0 0 0 0 0

MVC – C bit move

**Unified
instruction**

Grammar	Operation	Compiling result
mvc rz	RZ \leftarrow C	Only 32-bit instructions exist. mvc32 rz;

Description: Transfer the condition bit C to the lowest bit of RZ, and clear other bits of RZ.

Influence on flag bit: No influence

Exception: None

**32-bit
instruction**

Operation: RZ \leftarrow C

Grammar: mvc32 rz

Description: Transfer the condition bit C to the lowest bit of RZ, and clear other bits of RZ.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	0	0	0	0	0	1	RZ

MVCV – C bit reverse move

**Unified
instruction**

Grammar	Operation	Compiling result
mvcv rz	$RZ \leftarrow (\neg C)$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 16$), then mvcv16 rz; else mvcv32 rz;

Description: Transfer the condition bit C to the lowest bit of RZ after negation, and clear other bits of RZ.

Influence on flag bit: No influence

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow (\neg C)$

Grammar: mvcv16 rz

Description: Transfer the condition bit C to the lowest bit of RZ after negation, and clear other bits of RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1	0
-------	------	-----	-----	---

0	1 1 0 0 1	RZ	0 0 0 0	1 1
---	-----------	----	---------	-----

32-bit

instruction

Operation: RZ \leftarrow (!C)

Grammar: mvcv32 rz

Description: Transfer the condition bit C to the lowest bit of RZ after negation, and clear other bits of RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 1	1 0 0 0 0	RZ	

MVTC – Copy overflow bit to C bit

Unified instruction

Grammar	Operation	Compiling result
mvtc	C ← V	Only 32-bit instructions exist. mvtc32;

Description: Copy the overflow bit of DCSR(CR <14,0>) to C bit.

Influence on flag bit: C bit is set according to the overflow bit

Exception: None

32-bit

instruction

Operation: C ← V

Grammar: mvtc32

Description: Copy the overflow bit of DCSR(CR <14,0>) to C bit.

Influence on flag bit: C bit is set according to the overflow bit

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	0	0	0	0	1	0	0

NIE – Interrupt nesting enable

**Unified
instruction**

Grammar	Operation	Compiling result
nie	<p>Store the interrupted control register site {EPSR, EPC} to the stack storage, then update the stack pointer register to the top of stack storage, and initiate PSR.IE and PSR.EE;</p> <p>MEM[SP-4] ← EPC; MEM[SP-8] ← EPSR; SP ← SP-8; PSR({EE,IE}) ← 1</p>	Only 16-bit instructions exist. nie;

Attribute:	Privileged instruction
Description:	Store the interrupted control register site {EPSR, EPC} to the stack storage, then update the stack pointer register to the top of stack storage, and initiate the interrupt and exception enable bit PSR.IE and PSR.EE. Adopt direct addressing mode of stack pointer register.
Influence on flag bit:	No influence
Exception:	Access error exception, unaligned exception and privilege violation exception

16-bit instruction

Operation:	Store the interrupted control register site {EPSR, EPC} to the stack storage, then update the stack pointer register to the top of stack storage, and initiate PSR.IE and PSR.EE;
	MEM[SP-4] ← EPC; MEM[SP-8] ← EPSR; SP ← SP-8; PSR({EE,IE}) ← 1
Grammar:	nie16
Attribute:	Privileged instruction

Description: Store the interrupted control register site {EPSR, EPC} to the stack storage, then update the stack pointer register to the top of stack storage, and initiate the interrupt and exception enable bit PSR.IE and PSR.EE. Adopt direct addressing mode of stack pointer register.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, and access error exception

Instruction

format:

15	14	10	9	8	7	5	4	0
0	0	0	1	0	1	0	0	0

NIR – Interrupt nesting return

**Unified
instruction**

Grammar	Operation	Compiling result
nir	<p>Load the interrupted control register site {EPSR, EPC} from the stack storage, and then update the stack pointer register to the top of stack storage; return from interrupt</p> $\text{EPSR} \leftarrow \text{MEM}[\text{SP}]$ $\text{EPC} \leftarrow \text{MEM}[\text{SP}+4];$ $\text{SP} \leftarrow \text{SP}+8;$ $\text{PSR} \leftarrow \text{EPSR};$ $\text{PC} \leftarrow \text{EPC}$	<p>Only 16-bit instructions exist.</p> <p>nir;</p>

Attribute:	Privileged instruction
Description:	<p>Load the interrupted site {EPSR, EPC} from the stack storage, and then update the stack pointer register to the top of stack storage; restore PC value to the value in control register EPC and restore PSR value to EPSR value; the instruction is executed from the new PC address.</p> <p>Adopt direct addressing mode of stack pointer register.</p>
Influence on flag bit:	No influence
Exception:	Access error exception, unaligned exception and privilege violation exception

16-bit instruction	
Operation:	<p>Load the interrupted control register site {EPSR, EPC} from the stack storage, and then update the stack pointer register to the top of stack storage; return from interrupt</p> $\text{EPSR} \leftarrow \text{MEM}[\text{SP}]$ $\text{EPC} \leftarrow \text{MEM}[\text{SP}+4];$ $\text{SP} \leftarrow \text{SP}+8;$ $\text{PSR} \leftarrow \text{EPSR};$ $\text{PC} \leftarrow \text{EPC}$

Grammar:	nir16
Attribute:	Privileged instruction
Description:	<p>Load the interrupted site {EPSR, EPC} from the stack storage, and then update the stack pointer register to the top of stack storage; restore PC value to the value in control register EPC and restore PSR value to EPSR value; the instruction is executed from the new PC address.</p> <p>Adopt direct addressing mode of stack pointer register.</p>
Influence on flag	No influence
bit:	
Exception:	Unaligned access exception, unaligned access exception, and access error exception
Instruction format:	

15	14	10	9	8	7	5	4	0
0	0	0	1	0	1	0	0	0

NOR – Bitwise NOT-OR

**Unified
instruction**

Grammar	Operation	Compiling result
nor rz, rx	$RZ \leftarrow !(RZ RX)$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then nor16 rz, rx; else nor32 rz, rz, rx;
nor rz, rx, ry	$RZ \leftarrow !(RX RY)$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($y == z$) and ($x < 16$) and ($z < 16$), then nor16 rz, rx else nor32 rz, rx, ry

Description: Perform a bitwise OR of the values of RX and RY/RZ, then perform a bitwise NOT, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

**16-bit
instruction**

Operation: $RZ \leftarrow !(RZ | RX)$

Grammar: nor16 rz, rx

Description: Perform a bitwise OR of the values of RZ and RX, then perform a bitwise NOT, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 1 1	RZ	RX	1 0

32-bit

instruction

Operation: $RZ \leftarrow !(RX \mid RY)$

Grammar: nor32 rz, rx, ry

Description: Perform a bitwise OR of the values of RX and RY, then perform a bitwise NOT, and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 1 0 0 1	0 0 1 0 0	RZ	

NOT – Bitwise NOT#

Unified instruction

Grammar	Operation	Compiling result
not rz	$RZ \leftarrow !(RZ)$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 16$), then not16 rz; else not32 rz, rz;
not rz, rx	$RZ \leftarrow !(RX)$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x == z$) and ($z < 16$), then not16 rz; else not32 rz, rx;

Description: Perform a bitwise NOT of RZ/RX value and save the result in RZ.

Attention: This instruction is the pseudo instruction of nor rz, rz and nor rz, rx, rx.

Influence on flag bit: No influence

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow !(RZ)$

Grammar: not16 rz

Description: Perform a bitwise NOT of RZ value and save the result in RZ.

Attention: This instruction is the pseudo instruction of nor16 rz, rz.

Influence on flag bit: No influence

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 1 1	RZ	RZ	1 0

32-bit

instruction

Operation: $RZ \leftarrow !(RX)$

Grammar: not32 rz, rx

Description: Perform a bitwise NOT of RX value and save the result in RZ.

Attention: This instruction is the pseudo instruction of nor32 rz, rx, rx.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RX	RX	0 0 1 0 0 1	0 0 1 0 0	RZ	

OR – Bitwise OR

**Unified
instruction**

Grammar	Operation	Compiling result
or rz, rx	$RZ \leftarrow RZ RX$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then or16 rz, rx ; else or32 rz, rz, rx;
or rz, rx, ry	$RZ \leftarrow RX RY$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($y == z$) and ($x < 16$) and ($z < 16$), then or16 rz, rx else or32 rz, rx, ry

Description: Perform a bitwise OR of the values of RX and RY/RZ, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow RZ | RX$

Grammar: or16 rz, rx

Description: Perform a bitwise OR of the values of RZ and RX, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 1 1	RZ	RX	0 0

32-bit

instruction

Operation: $RZ \leftarrow RX \mid RY$

Grammar: or rz, rx, ry

Description: Perform a bitwise OR of the values of RX and RY, and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 1 0 0 1	0 0 0 0 1	RZ	

ORI – Bitwise OR immediate

Unified instruction

Grammar	Operation	Compiling result
ori rz, rx, imm16	$RZ \leftarrow RX \mid \text{zero_extend(IMM16)}$	Only 32-bit instructions exist. ori32 rz, rx, imm16

Description: Zero-extend the 16-bit immediate operand to 32 bits, perform a bitwise OR with RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

32-bit instruction

Operation: $RZ \leftarrow RX \mid \text{zero_extend(IMM16)}$

Grammar: ori32 rz, rx, imm16

Description: Zero-extend the 16-bit immediate operand to 32 bits, perform a bitwise OR with RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	0
1	1	1	0	1	1	RZ	RX	IMM16

PLDR – Prefetch read data

Unified instruction

Grammar	Operation	Compiling result
pldr (rx, disp)	Prefetch read data MEM[RX + zero_extend(offset << 2)]	Only 32-bit instructions exist. pldr32 (rx, disp)

Description: This instruction aims to accelerate the data loading behavior. Before data loading, PLDR instruction is started to read the data line into Cache; when Load instruction is executed, it is aimed at D-Cache, so the data loading efficiency is increased.
The effective address of this instruction is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag bit: No influence

Exception: None

32-bit instruction

Operation: Prefetch read data
Grammar: pldr32 (rx, disp)

Description: This instruction aims to accelerate the data loading behavior. Before data loading, PLDR instruction is started to read the data line into Cache; when Load instruction is executed, D-Cache will be targeted, so the data loading efficiency is increased.

The effective address of this instruction is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag bit: No influence

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
1 1 0 1 1 0	0 0 0 0 0	RX	0 1 1 0		Offset

PLDW – Prefetch write data

Unified instruction

Grammar	Operation	Compiling result
pldw (rx, disp)	Prefetch write data MEM[RX + zero_extend(offset << 2)]	Only 32-bit instructions exist. pldw32 (rx, disp)

Description: This instruction aims to accelerate the data storage behavior under write allocation Cache strategy. Under write allocation Cache strategy, a backfilling operation will be caused to Cache if the store instruction Cache is not targeted. PLDW instruction will read data to be stored into Cache before store instruction; when Store instruction is executed, D-Cache will be targeted, so the data storage efficiency is increased. The effective address of this instruction is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag bit: No influence

Exception: None

32-bit instruction

Operation: Prefetch write data

Grammar: pldw32 (rx, disp)

Description: This instruction aims to accelerate the data storage behavior under write allocation Cache strategy. Under write allocation Cache strategy, a backfilling operation will be caused to Cache if the store instruction Cache is not targeted. PLDW instruction will read data to be stored into Cache before store instruction; when Store instruction is executed, D-Cache will be targeted, so the data storage efficiency is increased. The effective address of this instruction is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting

left by two bits to 32 bits. Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
1 1 0 1 1 1 0 0 0 0 0 RX 0 1 1 0 Offset					

POP – Pop

**Unified
instruction**

Grammar	Operation	Compiling result
pop reglist	<p>Load multiple consecutive words from stack storage to a group of consecutive register files, then update the stack register to the top of stack storage, and return from the subprogram;</p> <pre>dst ← {reglist}; addr ← SP; foreach (reglist){ Rdst ← MEM[addr]; dst ← next {reglist}; addr ← addr + 4; } sp ← addr; PC ← R15 & 0xffffffff;</pre>	<p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of register</p> <pre>if ({reglist}<16), then pop16 reglist; else pop32 reglist;</pre>

Description: Load multiple consecutive words from stack storage to a group of consecutive register files, update the stack pointer register, and realize the function of returning from the subprogram. In another word, the program jumps to the position appointed by link register R15 and the lowest bit of link register is ignored. Adopt the direct addressing mode of stack register.

**Influence on flag
bit:** No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Load multiple consecutive words from stack storage to a group of consecutive register files, then update the stack register to the top of stack storage, and return from the subprogram

```

dst ← {reglist}; addr ← SP;
foreach ( reglist ){
    Rdst ← MEM[addr];
    dst ← next {reglist};
    addr ← addr + 4;
}
sp ← addr;
PC ← R15 & 0xffffffff;

```

Grammar: pop16 reglist

Description: Load multiple consecutive words from stack storage to a group of consecutive register files, update the stack pointer register, and realize the function of returning from the subprogram. In another word, the program jumps to the position appointed by link register R15 and the lowest bit of link register is ignored. Adopt the direct addressing mode of stack pointer register.

Influence on flag No influence

bit:

Restriction: The range of register is r4 – r11, r15.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

15	14	10	9	8	7	6	5	4	3	0
0	0	0	1	0	1	0	0	R15	LIST1	

LIST1 field – Assign whether registers r4-r11 are in the register list.

0000 – r4-r11 are not in the register list

0001 – r4 is in the register list

0010 – r4-r5 are in the register list

0011 – r4-r6 are in the register list

.....

1000 – r4-r11 are in the register list

R15 field – Assign whether register r15 is in the register list.

0 – r15 is not in the register list

1 – r15 is in the register list

32-bit instruction

Operation: Load multiple consecutive words from stack storage to a group of consecutive register files

```
dst ← {reglist}; addr ← SP;
foreach ( reglist ){
    Rdst ← MEM[addr];
    dst ← next {reglist};
    addr ← addr + 4;
}
```

```
sp ← addr;
PC ← R15 & 0xffffffff;
```

Grammar: pop32 reglist

Description: Load multiple consecutive words from stack storage to a group of consecutive register files, update the stack pointer register, and realize the function of returning from the subprogram. In another word, the program jumps to the position appointed by link register R15 and the lowest bit of link register is ignored. Adopt the direct addressing mode of stack pointer register.

Influence on flag No influence

bit:

Restriction: The range of register is r4 – r11, r15, r16 - r17, r28.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

31	30	26	25	21	20	16	15	12	11	10	9	8	7	6	5	4	3	0	
1	1	1	0	1	0	1	1	1	1	0	0	0	0	0	0	R28	LIST2	R15	LIST1

LIST1 field – Assign whether registers r4-r11 are in the register list.

0000 – r4-r11 are not in the register list

0001 – r4 is in the register list

0010 – r4-r5 are in the register list

0011 – r4-r6 are in the register list

.....

1000 – r4-r11 are in the register list

R15 field – Assign whether register r15 is in the register list.

0 – r15 is not in the register list

1 – r15 is in the register list

LIST2 field – Assign whether registers r16-r17 are in the register list.

000 – r16-r19 are not in the register list

001 – r16 is in the register list

010 – r16-r17 are in the register list

R28 field – Assign whether register r28 is in the register list.

0 – r28 is not in the register list

1 – r28 is in the register list

PSRCLR – Clear PSR bit

Unified instruction

Grammar	Operation	Compiling result
psrclr ee, ie, fe, af Or the operand can be any combination of ee, ie, fe and af.	Clear a certain bit or several bits of status register $\text{PSR}(\{\text{EE, IE, FE, AF}\}) \leftarrow 0$	Only 32-bit instructions exist. psrclr32 ee, ie, fe, af

Attribute: Privileged instruction

Description: The selected PSR bit is cleared (1 means that it is selected). The 5-bit immediate operand IMM5 is used to code the control bit to be cleared, and the corresponding relation is as follows:

Various bits of immediate operand IMM5	Corresponding PSR control bit
Imm5[0]	AF
Imm5[1]	FE
Imm5[2]	IE
Imm5[3]	EE
Imm5[4]	Retain

Influence on No influence

flag bit:

Exception: Privilege violation exception

32-bit

instruction

Operation: Clear a certain bit or several bits of status register

$\text{PSR}(\{\text{EE, IE, FE, AF}\}) \leftarrow 0$

Grammar: psrclr32 ee, ie, fe, af

Or the operand can be any combination of ee, ie, fe and af.

Attribute: Privileged instruction

Description: The selected PSR bit is cleared (1 means that it is selected). The 5-bit immediate operand IMM5 is used to code the control bit to be cleared, and the corresponding relation is as follows:

Various bits of immediate operand	Corresponding PSR control bit
-----------------------------------	-------------------------------

IMM5	
Imm5[0]	AF
Imm5[1]	FE
Imm5[2]	IE
Imm5[3]	EE
Imm5[4]	Retain

Influence on No influence

flag bit:

Exception: Privilege violation exception

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	0	IMM5	0	0	0	0	1	0

PSRSET – Set PSR bit

Unified instruction

Grammar	Operation	Compiling result
psrset ee, ie, fe, af Or the operand can be any combination of ee, ie, fe and af.	Set several bits of status register $\text{PSR}(\{\text{EE, IE, FE, AF}\}) \leftarrow 1$	Only 32-bit instructions exist. psrset32 ee, ie, fe, af

Attribute: Privileged instruction

Description: The selected PSR bit is set (1 means that it is selected). The 5-bit immediate operand IMM5 is used to code the control bit to be cleared, and the corresponding relation is as follows:

Various bits of immediate operand IMM5	Corresponding PSR control bit
Imm5[0]	AF
Imm5[1]	IE
Imm5[2]	EE
Imm5[3]	EE
Imm5[4]	Retain

Influence on No influence

flag bit:

Exception: Privilege violation exception

32-bit

instruction

Operation: Set several bits of status register

$\text{PSR}(\{\text{EE, IE, FE, AF}\}) \leftarrow 1$

Grammar: psrset32 ee, ie, fe, af

Or the operand can be any combination of ee, ie, fe and af.

Attribute: Privileged instruction

Description: The selected PSR bit is set (1 means that it is selected). The 5-bit immediate operand IMM5 is used to code the control bit to be cleared, and the corresponding relation is as follows:

Various bits of immediate operand	Corresponding PSR control bit
-----------------------------------	-------------------------------

IMM5	
Imm5[0]	AF
Imm5[1]	FE
Imm5[2]	IE
Imm5[3]	EE
Imm5[4]	Retain

Influence on No influence

flag bit:

Exception: Privilege violation exception

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0	IMM5	0 0 0 0 0	0 1 1 1 0 1	0 0 0 0 1	0 0 0 0 0	

PUSH – Push

**Unified
instruction**

Grammar	Operation	Compiling result
push reglist	<p>Store words in register list to stack storage, and update stack register to the top of stack storage;</p> <pre>src ← {reglist}; addr ← SP; foreach (reglist){ addr ← addr - 4; MEM[addr] ← Rsrc; src ← next {reglist}; } sp ← addr;</pre>	<p>Compiled into corresponding 16-bit or 32-bit instructions according to the range of register</p> <pre>if ({reglist}<16), then push16 reglist; else push32 reglist;</pre>

Description: Store words in register list to stack storage, and update stack register to the top of stack storage. Adopt the direct addressing mode of stack register.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Store words in register list to stack storage

```
src ← {reglist}; addr ← SP;
foreach ( reglist ){
    MEM[addr] ← Rsrc;
    src ← next {reglist};
    addr ← addr - 4;
}
```

Grammar:

Description: Store words in register list to stack storage, and update stack register to

the top of stack storage. Adopt the direct addressing mode of stack register.

Influence on flag No influence

bit:

Restriction: The range of register is r4 – r11, r15.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

15	14	10	9	8	7	6	5	4	3	0
0	0	0	1	0	1	0	1	1	0	R15 LIST1

LIST1 field – Assign whether registers r4-r11 are in the register list.

0000 – r4-r11 are not in the register list

0001 – r4 is in the register list

0010 – r4-r5 are in the register list

0011 – r4-r6 are in the register list

.....

1000 – r4-r11 are in the register list

R15 field – Assign whether register r15 is in the register list.

0 – r15 is not in the register list

1 – r15 is in the register list

32-bit instruction

Operation: Load multiple consecutive words from stack storage to a group of consecutive register files

```
src ← {reglist}; addr ← SP;
```

```
foreach ( reglist ){
```

```
    MEM[addr] ← Rsrc;
```

```
    src ← next {reglist};
```

```
    addr ← addr - 4;
```

}

sp ← addr

Grammar: push32 reglist

Description: Store words in register list to stack storage, and update stack register to the top of stack storage. Adopt the direct addressing mode of stack register.

Influence on flag No influence

bit:

Restriction: The range of register is r4 – r11, r15, r16 - r17, r28.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

31	30	26	25	21	20	16	15	12	11	10	9	8	7	6	5	4	3	0		
1	1	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	R28	LIST2	R15	LIST1

LIST1 field – Assign whether registers r4-r11 are in the register list.

0000 – r4-r11 are not in the register list

0001 – r4 is in the register list

0010 – r4-r5 are in the register list

0011 – r4-r6 are in the register list

.....

1000 – r4-r11 are in the register list

R15 field – Assign whether register r15 is in the register list.

0 – r15 is not in the register list

1 – r15 is in the register list

LIST2 field – Assign whether registers r16-r17 are in the register list.

000 – r16-r19 are not in the register list

001 – r16 is in the register list

010 – r16-r17 are in the register list

R28 field – Assign whether register r28 is in the register list.

0 – r28 is not in the register list

1 – r28 is in the register list



REVB – Byte-reverse

**Unified
instruction**

Grammar	Operation	Compiling result
revb rz, rx	RZ[31:24] ← RX[7:0]; RZ[23:16] ← RX[15:8]; RZ[15:8] ← RX[23:16]; RZ[7:0] ← RX[31:24];	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if (x<16) and (z<16), then revb16 rz, rx; else revb32 rz, rx;

Description: Get the reverse order of RX value according to the byte, keep bit order inside the byte unchanged, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

**16-bit
instruction**

Operation: RZ[31:24] ← RX[7:0];
RZ[23:16] ← RX[15:8];
RZ[15:8] ← RX[23:16];
RZ[7:0] ← RX[31:24];

Grammar: revb16 rz, rx

Description: Get the reverse order of RX value according to the byte, keep bit order inside the byte unchanged, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

**Instruction
format:**

15 14	10 9	6 5	2 1 0
-------	------	-----	-------

0	1 1 1 1 0	RZ	RX	1 0
---	-----------	----	----	-----

32-bit

instruction

Operation: RZ[31:24] ← RX[7:0];
 RZ[23:16] ← RX[15:8];
 RZ[15:8] ← RX[23:16];
 RZ[7:0] ← RX[31:24];

Grammar: revb32 rz, rx

Description: Get the reverse order of RX value according to the byte, keep bit order inside the byte unchanged, and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 0 0 0	0 0 1 0 0	RZ

REVH – Half-word byte-reverse

Unified instruction

Grammar	Operation	Compiling result
revh rz, rx	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← RX[15:8];	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if (x<16) and (z<16), then revh16 rz, rx; else revh32 rz, rx;

Description: Get the reverse order of RX value within half-word according to the byte. In another word, two bytes in the high half-word and two bytes in the low half-word are exchanged. Keep the bit order between two half-words and the bit order inside the byte unchanged, and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

16-bit instruction

Operation: RZ[31:24] ← RX[23:16];
RZ[23:16] ← RX[31:24];
RZ[15:8] ← RX[7:0];
RZ[7:0] ← RX[15:8];

Grammar: revh16 rz, rx

Description: Get the reverse order of RX value within half-word according to the byte. In another word, two bytes in the high half-word and two bytes in the low half-word are exchanged. Keep the bit order between two half-words and the bit order inside the byte unchanged, and save the result in RZ.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 1 1 0	RZ	RX	1 1

32-bit

instruction

Operation: RZ[31:24] ← RX[23:16];

RZ[23:16] ← RX[31:24];

RZ[15:8] ← RX[7:0];

RZ[7:0] ← RX[15:8];

Grammar: revh32 rz, rx

Description: Get the reverse order of RX value within half-word according to the byte.

In another word, two bytes in the high half-word and two bytes in the low half-word are exchanged. Keep the bit order between two half-words and the bit order inside the byte unchanged, and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0 1	0 0 0 0 0	RX	0 1 1 0 0 0	0 1 0 0 0	RZ	

RFI – Return from fast interrupt

Unified instruction

Grammar	Operation	Compiling result
rfi	Return from fast interrupt PC ← FPC, PSR ← FPSR	Only 32-bit instructions exist. rfi32

Attribute: Privileged instruction

Description: Restore PC value to value saved in control register FPC and restore PSR value to value saved in FPSR; the instruction is executed from the new PC address.

Influence on flag No influence

bit:

Exception: Privilege violation exception

32-bit instruction

Operation: Return from fast interrupt
PC ← FPC, PSR ← FPSR

Grammar: rfi32

Attribute: Privileged instruction

Description: Restore PC value to value saved in control register FPC and restore PSR value to value saved in FPSR; the instruction is executed from the new PC address.

Influence on flag No influence

bit:

Exception: Privilege violation exception

**Instruction
format:**

31 30

26 25

21 20

16 15

10 9

5 4

0

1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 0 0 1	0 0 0 0 1	0 0 0 0 0
---	-----------	-----------	-----------	-------------	-----------	-----------



ROTL – Rotate left

**Unified
instruction**

Grammar	Operation	Compiling result
rotl rz, rx	$RZ \leftarrow RZ <<< RX[5:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then rotl16 rz, rx ; else rotl32 rz, rz, rx;
rotl rz, rx, ry	$RZ \leftarrow RX <<< RY[5:0]$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x == z$) and ($y < 16$) and ($z < 16$), then rotl16 rz, ry else rotl32 rz, rx, ry

Description: For rotl rz, rx, perform a ring left shift on RZ value (the original value shifts left and the bit shifting out from left side will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 31, RZ will be cleared;

For lsl rz, rx, ry, perform a ring left shift on RX value (the original value shifts left and the bit shifting out from left side will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RY (RY[5:0]). If the value of RY[5:0] is greater than 31, RZ will be cleared.

Influence on flag No influence

bit:

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow RZ <<< RX[5:0]$

Grammar: rotl16 rz, rx

Description: Perform a ring left shift on RZ value (the original value shifts left and the bit shifting out from left side will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 31, RZ will be cleared.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction
format:

15	14	10	9	6	5	2	1	0
0	1 1 1 0 0	RZ	RX	1 1				

32-bit
instruction

Operation: $RZ \leftarrow RX <<< RY[5:0]$

Grammar: rotl32 rz, rx, ry

Description: Perform a ring left shift on RX value (the original value shifts left and the bit shifting out from left side will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RY (RY[5:0]). If the value of RY[5:0] is greater than 31, RZ will be cleared.

Influence on flag No influence

bit:

Exception: None

Instruction
format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 1 0 0 0	RZ						

ROTLI – Rotate left immediate

**Unified
instruction**

Grammar	Operation	Compiling result
rotli rz, rx, imm5	$RZ \leftarrow RX \lll IMM5$	rotli32 rz, rx, imm5;

Description: Perform a ring left shift on RX value (the original value shifts left and the bit shifting out from left side will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value is the same with RX value.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0-31.

Exception: None

32-bit

instruction

Operation: $RZ \leftarrow RX \lll IMM5$

Grammar: rotli32 rz, rx, imm5

Description: Perform a ring left shift on RX value (the original value shifts left and the bit shifting out from left side will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero, RZ value is the same with RX value.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0-31.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 0	0 1 0 0 0	RZ

CSKY 中文

RSUB – Reverse subtract#

**Unified
instruction**

Grammar	Operation	Compiling result
rsub rz, rx, ry	$RZ \leftarrow RY - RX$	Only 32-bit instructions exist. rsub32 rz, rx, ry

Description: Subtract RX value from RY value and save the result in RZ.

Attention: This instruction is the pseudo instruction of subu rz, ry, rx.

Influence on flag bit: No influence

Exception: None

32-bit

instruction

Operation: $RZ \leftarrow RY - RX$

Grammar: rsub32 rz, rx, ry

Description: Subtract RX value from RY value and save the result in RZ.

Attention: This instruction is the pseudo instruction of subu32 rz, ry, rx.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RX	RY	0	0	0	0	RZ

RTS – Return from subprogram#

Unified instruction

Grammar	Operation	Compiling result
rts	The program jumps to the position appointed by link register $PC \leftarrow R15 \& 0xffffffffe$	Always compiled into 16-bit instruction. rts16

Description: The program jumps to the position appointed by link register R15 and the lowest bit of link register is ignored. The jump range of RTS16 instruction is the whole address space of 4GB.
 This instruction is used to realize the function of returning from subprogram.
 Attention: This instruction is the pseudo instruction of jmp r15.

Influence on flag No influence

bit:

Exception: None

16-bit instruction

Operation: The program jumps to the position appointed by link register
 $PC \leftarrow R15 \& 0xffffffffe$

Grammar: rts16

Description: The program jumps to the position appointed by link register R15 and the lowest bit of link register is ignored. The jump range of RTS16 instruction is the whole address space of 4GB.
 This instruction is used to realize the function of returning from subprogram.
 Attention: This instruction is the pseudo instruction of jmp16 r15.

Influence on flag No influence

bit:

Exception: None

**Instruction
format:**

15 14	10 9	6 5	2 1 0
0 1 1 1 1 0	0 0 0 0	1 1 1 1	0 0

32-bit

instruction

Operation: The program jumps to the position appointed by link register

$$PC \leftarrow R15 \& 0xffffffff$$

Grammar: rts32

Description: The program jumps to the position appointed by link register R15 and the lowest bit of link register is ignored. The jump range of RTS instruction is the whole address space of 4GB.

This instruction is used to realize the function of returning from subprogram.

Attention: This instruction is the pseudo instruction of jmp32 r15.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	0
1 1 1 0 1 0	0 0 1 1 0	0 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	

RTE – Return from abnormal and normal interrupt

Unified instruction

Grammar	Operation	Compiling result
rte	Return from abnormal and normal interrupt PC ← EPC, PSR ← EPSR	Only 32-bit instructions exist. rte32

Attribute: Privileged instruction

Description: Restore PC value to value saved in control register EPC and restore PSR value to value saved in EPSR; the instruction is executed from the new PC address.

Influence on flag bit: No influence

Exception: Privilege violation exception

32-bit instruction

Operation: Return from abnormal and normal interrupt
PC ← EPC, PSR ← EPSR

Grammar: rte32

Attribute: Privileged instruction

Description: Restore PC value to value saved in control register EPC and restore PSR value to value saved in EPSR; the instruction is executed from the new PC address.

Influence on flag bit: No influence

Exception: Privilege violation exception

Instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	0	0	0	0	1	0	0	0

SCE – Set conditional execution

Unified instruction

Grammar	Operation	Compiling result
sce cond	Set the conditional execution bits of the following 4 instructions	Only 32-bit instructions exist. sce32 cond

Description: SCE instruction is used to set the conditional execution bits of the following 4 instructions. The operand COND is a 4-bit binary immediate operand. The lowest bit refers to the condition bit of the first instruction after sce instruction; the second lowest bit refers to the condition bit of the second instruction after sce instruction, and the like. If the condition bit is 1, it means that normal execution is realized when C is 1; if the condition bit is 0, it means that normal execution is realized when C is 0. If C bit does not meet the condition bit, conditional execution instruction will not generate any influence. The value used to judge C bit is subject to the value when sce instruction is executed.

Influence on flag bit: When exception or interrupt happens to the following 4 instructions, conditional execution bit will be saved in EPSR or FPSR.

Restriction: Instructions that can be set by sce instruction only include arithmetic operation instruction, multiply-divide instructions, and byte, half-word and word load/store instructions of immediate operand addressing mode; besides, these instructions should not affect condition bit C. Instructions that cannot be set by sce instruction include but are not limited to: branch jump instruction, load/store instruction of register addressing mode, load/store instruction of double word, load/store instruction of multiword, co-processor instruction, privileged instruction, special function instruction, floating point instruction, and vector multimedia instruction.

The operand is a 4-bit binary immediate operand.

Exception: None

Remark: The instruction sequence is:

sce 0101

mov r1, r0

```
mov r3, r2
mov r5, r4
mov r7, r6
```

The condition bit of sce instruction is 0101. If C bit is 0 when sce instruction is executed, the second and fourth mov instructions meet the execution condition, and the result is written into register 3 and register 7; the first and third mov instructions do not meet the execution condition, and the result will not be written into destination register.

32-bit instruction

Operation: Set the conditional execution bits of the following 4 instructions

```
set_condition_execution(COND);
```

Grammar: sce32 cond

Description: SCE instruction is used to set the conditional execution bits of the following 4 instructions. The operand COND is a 4-bit binary immediate operand. The lowest bit refers to the condition bit of the first instruction after sce instruction; the second lowest bit refers to the condition bit of the second instruction after sce instruction, and the like. If the condition bit is 1, it means that normal execution is realized when C is 1; if the condition bit is 0, it means that normal execution is realized when C is 0. If C bit does not meet the condition bit, conditional execution instruction will not generate any influence. The value used to judge C bit is subject to the value when sce instruction is executed.

Influence on flag bit: When exception or interrupt happens to the following 4 instructions, conditional execution bit will be saved in EPSR or FPSR.

Restriction: Instructions that can be set by sce instruction only include arithmetic operation instruction, multiply-divide instructions, and byte, half-word and word load/store instructions of immediate operand addressing mode; besides, these instructions should not affect condition bit C.

Instructions that cannot be set by sce instruction include but are not limited to: branch jump instruction, load/store instruction of register addressing mode, load/store instruction of double word, load/store instruction of multiword, co-processor instruction, privileged instruction, special function instruction, floating point instruction, and vector multimedia instruction.

The operand is a 4-bit binary immediate operand.

Exception: None

Instruction

format:

Remark: The instruction sequence is:

sce32 0101

mov32 r1, r0

mov32 r3, r2

mov32 r5, r4

mov32 r7, r6

The condition bit of sce instruction is 0101. If C bit is 0 when sce instruction is executed, the second and fourth mov instructions meet the execution condition, and the result is written into register 3 and register 7; the first and third mov instructions do not meet the execution condition, and the result will not be written into destination register.

31 30	26 25 24	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0 0 COND 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0						

SE – Send event

**Unified
instruction**

Grammar	Operation	Compiling result
se	Send events to the external part of processor	Only 32-bit instructions exist. se32

Description: As a communication mechanism between processor core and peripheral equipment, it can be used for multi-core communication.

Influence on flag bit: No influence

Exception: None

**32-bit
instruction**

Operation: Send events to the external part of processor

Grammar: se32

Description: As a communication mechanism between processor core and peripheral equipment, it can be used for multi-core communication.

Influence on flag bit: No influence

Exception: None

**Instruction
format:**

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SEXT – Extract bit and extend signed

Unified instruction

Grammar	Operation	Compiling result
sext rz, rx, msb, lsb	$RZ \leftarrow \text{sign_extend}(RX[\text{MSB:LSB}])$	Only 32-bit instructions exist. sext32 rz, rx, msb, lsb

Description:	Extract a section of consecutive bits of RX (RX[MSB:LSB]) appointed by 2 5-bit immediate operands (MSB,LSB), sign-extend it to 32 bits, and save the result in RZ. If MSB is equal to 31 and LSB is equal to zero, RZ value is the same with RX value. If MSB is equal to LSB, RZ value is the result after one-bit sign-extension of RX[MSB] (i.e. RX[LSB]). If MSB is smaller than LSB, behavior of this instruction is unpredictable.
Influence on flag bit:	No influence
Restriction:	The range of MSB is 0-31, the range of LSB is 0-31, and MSB should be greater than or equal to LSB.
Exception:	None

32-bit instruction

Operation:	$RZ \leftarrow \text{sign_extend}(RX[\text{MSB:LSB}])$
Grammar:	sext32 rz, rx, msb, lsb
Description:	Extract a section of consecutive bits of RX (RX[MSB:LSB]) appointed by 2 5-bit immediate operands (MSB,LSB), sign-extend it to 32 bits, and save the result in RZ. If MSB is equal to 31 and LSB is equal to zero, RZ value is the same with RX value. If MSB is equal to LSB, RZ value is the result after one-bit sign-extension of RX[MSB] (i.e. RX[LSB]). If MSB is smaller than LSB, behavior of this instruction is unpredictable.
Influence on flag bit:	No influence
Restriction:	The range of MSB is 0-31, the range of LSB is 0-31, and MSB should be greater than or equal to LSB.
Exception:	None

Instruction

format:

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 0 0 1	LSB	RX	0 1 0 1 1 0	MSB	RZ
---	-----------	-----	----	-------------	-----	----

MSB field – Assign the starting bit of extraction.

LSB field – Assign the end bit of extraction.

00000 – 0

00000 – 0 bit

00001 – 1

00001 – 1 bit

.....

.....

11111 – 31

11111 – 31 bits

SEXTB – Extract byte and extend signed#

**Unified
instruction**

Grammar	Operation	Compiling result
sextb rz, rx	$RZ \leftarrow \text{sign_extend}(RX[7:0]);$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 16$) and ($x < 16$), then sextb16 rz, rx; else sextb32 rz, rx;

Description: Sign-extend low bytes of RX (RX[7:0]) to 32 bits, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow \text{sign_extend}(RX[7:0]);$

Grammar: sextb16 rz, rx

Description: Sign-extend low bytes of RX (RX[7:0]) to 32 bits, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

**Instruction
format:**

15	14	10	9	6	5	2	1	0
0	1	1	1	0	1	RZ	RX	1 0

32-bit

instruction

Operation: $RZ \leftarrow \text{sign_extend}(RX[7:0]);$

Grammar: sextb32 rz, rx

Description: Sign-extend low bytes of RX (RX[7:0]) to 32 bits, and save the result in RZ.

Attention: This instruction is the pseudo instruction of sext32 rz, rx,
0x7, 0x0.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	0 0 0 0 0	RX	0 1 0 1 1 0	0 0 1 1 1	RZ	

SEXTH – Extract half-word and extend signed#

**Unified
instruction**

Grammar	Operation	Compiling result
sexth rz, rx	$RZ \leftarrow \text{sign_extend}(RX[15:0]);$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 16$) and ($x < 16$), then sexth16 rz, rx; else sexth32 rz, rx;

Description: Sign-extend low half-word of RX (RX[15:0]) to 32 bits, and save the result in RZ.

Influence on flag bit: No influence

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow \text{sign_extend}(RX[15:0]);$

Grammar: sexth16 rz, rx

Description: Sign-extend low half-word of RX (RX[15:0]) to 32 bits, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

**Instruction
format**

15	14	10	9	6	5	2	1	0
0	1	1	1	0	1	RZ	RX	1 1

32-bit instruction

Operation: $RZ \leftarrow \text{sign_extend}(RX[15:0]);$

Grammar: sexth32 rz, rx

Description: Sign-extend low half-word of RX (RX[15:0]) to 32 bits, and save the result in RZ.

Attention: This instruction is the pseudo instruction of sext32 rz, rx,
0x15, 0x0.

Influence on flag No influence

bit:

Exception: None

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1	0	0	0	1	0	0	0	0	0	RX	0	1	0	1	1	0	0	1	1	1	1	RZ
---	---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	---	---	---	---	----

SRS.B – Store byte sign

Unified instruction

Grammar	Operation	Compiling result
srs.b rz, [label]	Store the lowest byte sign in register to storage $\text{MEM}[\text{R28} + \text{zero_extend}(\text{offset})] \leftarrow \text{RZ}[7:0]$	Only 32-bit instructions exist. srs32.b rz, [label]

Description: Store the lowest byte sign in register RZ to the position of label. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 18-bit relative offset to 32 bits. The address space of SRS.B instruction is +256KB.
Attention: The offset DISP is the offset of binary operand.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Store the lowest byte sign in register to storage

$\text{MEM}[\text{R28} + \text{zero_extend}(\text{offset})] \leftarrow \text{RZ}[7:0]$

Grammar: srs32.b rz, [label]

Description: Store the lowest byte sign in register RZ to the position of label. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 18-bit relative offset to 32 bits. The address space of SRS.B instruction is +256KB.
Attention: The offset DISP is the offset of binary operand.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable

exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	21 20	18 17	0
1 1 0 0 1 1	RZ	1 0 0		Offset

SRS.H – Store half-word sign

Unified instruction

Grammar	Operation	Compiling result
srs.h rz, [label]	Store the lowest half-word sign in register to storage $MEM[R28 + \text{zero_extend}(\text{offset} \ll 1)] \leftarrow RZ[7:0]$	Only 32-bit instructions exist. srs32.h rz, [label]

Description: Store the lowest half-word sign in register RZ to the position of label. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 18-bit relative offset shifting left by one bit to 32 bits. The address space of SRS.H instruction is +512KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Store the lowest half-word sign in register to storage

$MEM[R28 + \text{zero_extend}(\text{offset} \ll 1)] \leftarrow RZ[7:0]$

Grammar: srs32.h rz, [label]

Description: Store the lowest half-word sign in register RZ to the position of label. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 18-bit relative offset shifting left by one bit to 32 bits. The address space of SRS.H instruction is +512KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	21 20	18 17	0
1 1 0 0 1 1	RZ	1 0 1		Offset

SRS.W – Store word sign

Unified instruction

Grammar	Operation	Compiling result
srs.w rz, [label]	Store the lowest word sign in register to storage $MEM[R28 + \text{zero_extend}(\text{offset} \ll 2)] \leftarrow RZ[7:0]$	Only 32-bit instructions exist. srs32.w rz, [label]

Description: Store the lowest word sign in register RZ to the position of label. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 18-bit relative offset shifting left by two bits to 32 bits. The address space of SRS.W instruction is +1024KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Store the lowest word sign in register to storage
 $MEM[R28 + \text{zero_extend}(\text{offset} \ll 2)] \leftarrow RZ[7:0]$

Grammar: srs32.w rz, [label]

Description: Store the lowest word sign in register RZ to the position of label. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 18-bit relative offset shifting left by two bits to 32 bits. The address space of SRS.W instruction is +1024KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

**Instruction
format:**

31 30	26 25	21 20	18 17	0
1 1 0 0 1 1	RZ	1 1 0	Offset	

SRTE – Return from safe state

Unified instruction

Grammar	Operation	Compiling result
srte	Return from safe state PC \leftarrow SPC, PSR \leftarrow SPSR	Only 32-bit instructions exist. srte32

Attribute: Privileged instruction

Description: Restore PC value to value saved in control register SPC and restore PSR value to value saved in SPSR; the instruction is executed from the new PC address.

Influence on flag No influence

bit:

Exception: Privilege violation exception

32-bit instruction

Operation: Return from safe state
PC \leftarrow SPC, PSR \leftarrow SPSR

Grammar: srte32

Attribute: Privileged instruction

Description: Restore PC value to value saved in control register SPC and restore PSR value to value saved in SPSR; the instruction is executed from the new PC address.

Influence on flag No influence

bit:

Exception: Privilege violation exception

Instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	0	0	0	0	1	1	1	1

ST.B – Store byte

Unified instruction

Grammar	Operation	Compiling result
st.b rz, (rx, disp)	<p>Store the lowest byte in register to storage</p> $\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset})] \leftarrow \text{RZ}[7:0]$	<p>Compiled into 16-bit or 32-bit instructions according to the range of offset and register.</p> <p>if (disp<32) and (x<7) and (z<7), then</p> <pre>st16.b rz, (rx, disp); else st32.b rz, (rx, disp);</pre>

Description: Store the lowest byte in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset to 32 bits. The address space of ST.B instruction is +4KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Store the lowest byte in register to storage

$$\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset})] \leftarrow \text{RZ}[7:0]$$

Grammar: st16.b rz, (rx, disp)

Description: Store the lowest byte in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 5-bit relative offset to 32 bits. The address space of ST16.B instruction is +32B.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

15 14	11 10	8 7	5 4	0
-------	-------	-----	-----	---

1	0 1 0 0	RX	RZ	IMM5
---	---------	----	----	------

32-bit instruction

Operation: Store the lowest byte in register to storage

$$\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset})] \leftarrow \text{RZ}[7:0]$$

Grammar: st32.b rz, (rx, disp)

Description: Store the lowest byte in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset to 32 bits. The address space of ST32.B instruction is +4KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
-------	-------	-------	-------	-------	---

1	1 0 1 1 1	RZ	RX	0 0 0 0	Offset
---	-----------	----	----	---------	--------

ST.D – Store double word

Unified instruction

Grammar	Operation	Compiling result
st.d rz, (rx, disp)	Store double word in register to storage $\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset} \ll 2)] \leftarrow \text{RZ}[31:0]$ $\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset} \ll 2) + 0x4] \leftarrow \text{RZ} + 1[31:0]$	Only 32-bit instructions exist. st32.d rz, (rx, disp);

Description: Store double word in register RZ and RZ + 1 to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. The address space of ST.D instruction is +16KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

**Influence on flag
bit:**

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

32-bit instruction

Operation: Store double word in register to storage

$$\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset} \ll 2)] \leftarrow \text{RZ}[31:0]$$

$$\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset} \ll 2) + 0x4] \leftarrow \text{RZ} + 1[31:0]$$

Grammar: st32.d rz, (rx, disp)

Description: Store double word in register RZ and RZ + 1 to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. The address space of ST32.D instruction is +16KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
1 1 0 1 1 1	RZ	RX	0 0 1 1	Offset	

ST.H – Store half-word

Unified instruction

Grammar	Operation	Compiling result
st.h rz, (rx, disp)	<p>Store the lowest byte in register to storage</p> <p>MEM[RX + zero_extend(offset<< 1)] ← RZ[15:0]</p>	<p>Compiled into 16-bit or 32-bit instructions according to the range of offset and register.</p> <p>if (disp<64)and(x<7)and(z<7), then st16.h rz, (rx, disp); else st32.h rz, (rx, disp);</p>

Description: Store low half-word in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by one bit to 32 bits. The address space of ST.H instruction is +8KB.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Store low half-word in register to storage

MEM[RX + zero_extend(offset << 1)] ← RZ[15:0]

Grammar: st16.h rz, (rx, disp)

Description: Store low half-word in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 5-bit relative offset shifting left by one bit to 32 bits. The address space of ST16.H instruction is +64B.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 1 bit.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

15 14	11 10	8 7	5 4	0
-------	-------	-----	-----	---

1	0 1 0 1	RX	RZ	IMM5
---	---------	----	----	------

32-bit instruction

Operation: Store low half-word in register to storage

$\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset} \ll 1)] \leftarrow \text{RZ}[15:0]$

Grammar: st32.h rz, (rx, disp)

Description: Store low half-word in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by one bit to 32 bits. The address space of ST32.H instruction is +8KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 1 bit.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
-------	-------	-------	-------	-------	---

1	1 0 1 1 1	RZ	RX	0 0 0 1	Offset
---	-----------	----	----	---------	--------

ST.W – Store word

Unified instruction

Grammar	Operation	Compiling result
st.w rz, (rx, disp)	Store word in register to storage $MEM[RX + \text{zero_extend}(\text{offset} \ll 2)] \leftarrow RZ[31:0]$	Compiled into 16-bit or 32-bit instructions according to the range of offset and register. if ($x=sp$) and ($z \leq 7$) and ($\text{disp} < 1024$), st16.w rz, (sp, disp); else if ($\text{disp} \leq 128$) and ($x \leq 7$) and ($z \leq 7$), st16.w rz, (rx, disp); else st32.w rz, (rx, disp);

Description: Store word in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. The address space of ST.W instruction is +16KB.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

16-bit instruction

Operation: Store word in register to storage

$MEM[RX + \text{zero_extend}(\text{offset} \ll 2)] \leftarrow RZ[31:0]$

Grammar:
st16.w rz, (rx, disp)
st16.w rz, (sp, disp)

Description: Store word in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. When $rx=sp$, the effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by two bits to 32 bits. When rx is other register, the effective address of

storage is gained by adding the base register RX to the value of unsigned extending the 5-bit relative offset shifting left by two bits to 32 bits. The address space of ST16.W instruction is +1KB.

Attention: The offset DISP is gained after the binary operand IMM5 shifts left by two bits. When the base register RX is SP, the offset DISP is gained after the binary operand {IMM3, IMM5} shifts left by two bits.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

st16.w rz, (rx, disp)

15	14	11 10	8 7	5 4	0
----	----	-------	-----	-----	---

1	0 1 1 0	RX	RZ	IMM5	
---	---------	----	----	------	--

st16.w rz, (sp, disp)

15	14	11 10	8 7	5 4	0
----	----	-------	-----	-----	---

1	0 1 1 1	IMM3	RZ	IMM5	
---	---------	------	----	------	--

32-bit instruction

Operation: Store word in register to storage

$\text{MEM}[\text{RX} + \text{zero_extend}(\text{offset} \ll 2)] \leftarrow \text{RZ}[31:0]$

Grammar: st32.w rz, (rx, disp)

Description: Store word in register RZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. The address space of ST32.W instruction is +16KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
1 1 0 1 1 1	RZ	RX	0 0 1 0	Offset	

STCPR – Store word in co-processor

Unified instruction

Grammar	Operation	Compiling result
stcpr <cpid, cprz>, (rx, offset)	Store word in general-purpose register of co-processor to storage MEM[RX + sign_extend(offset << 2)] ← CPRZ	Only 32-bit instructions exist. stcpr32 <cpid, cprz>, (rx, offset)

Description: Store word in general-purpose register of co-processor CPRZ to storage. Adopt the addressing mode of register and immediate operand offset. Bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. 12 low bits are user defined.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Operation: Store word in general-purpose register of co-processor to storage
MEM[RX + sign_extend(offset << 2)] ← CPRZ

Grammar: stcpr32 <cpid, cprz>, (rx, offset)

Description: Store word in general-purpose register of co-processor CPRZ to storage. Adopt the addressing mode of register and immediate operand offset. Bits 24-21 are agreed as co-processor numbers and used to assign co-processor of pre-operation. 12 low bits are user defined.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction format:

31 30	26 25	21 20	16 15	12 11	0
-------	-------	-------	-------	-------	---

1	1 1 0 1 0	0	CVID	RX	0 1 1 1	User-define
---	-----------	---	------	----	---------	-------------



STEX.W – Store word exclusive

Unified instruction

Grammar	Operation	Compiling result
stex.w rz, (rx, disp)	If exclusive storage of word succeeds, then $MEM[RX + sign_extend(offset \ll 2)] \leftarrow RZ;$ $RZ \leftarrow 1;$ else $RZ \leftarrow 0;$	Only 32-bit instructions exist. stex32.w rz, (rx, disp)

Description: Store word in general-purpose register RZ to storage. If exclusive storage succeeds, the source register RZ returns to 1; if the source register returns to zero, it means failure of exclusive storage. STEX.W adopts the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. The address space of STEX.W instruction is +16KB.

This instruction matches LDEX.W and it is used for atom operation of “read storage – modify – write storage” in multi-core communication.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

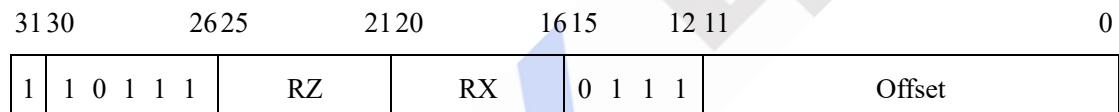
Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

32-bit instruction

Operation: If exclusive storage of word succeeds, then
 $MEM[RX + sign_extend(offset \ll 2)] \leftarrow RZ;$
 $RZ \leftarrow 1;$
 else
 $RZ \leftarrow 0;$

Grammar:	stex32.w rz, (rx, disp)
Description:	<p>Store word in general-purpose register RZ to storage. If exclusive storage succeeds, the source register RZ returns to 1; if the source register returns to zero, it means failure of exclusive storage.</p> <p>STEX32.W adopts the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by two bits to 32 bits. The address space of STEX32.W instruction is +16KB.</p> <p>This instruction matches LDEX32.W and it is used for atom operation of “read storage – modify – write storage” in multi-core communication.</p> <p>Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.</p>
Influence on flag bit:	No influence
Exception:	Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception
Instruction format:	



STM – Store consecutive multiword

Unified instruction

Grammar	Operation	Compiling result
stm ry-rz, (rx)	<p>Store contents in a group of consecutive register files to a group of consecutive storage addresses successively</p> <pre>src ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ MEM[addr] ← Rsrc; src ← src + 1; addr ← addr + 4; }</pre>	<p>Only 32-bit instructions exist.</p> <p>stm32 ry-rz, (rx)</p>

Description: Store contents in a group of consecutive register files starting from RY to a group of consecutive storage addresses successively. In another word, store contents in register RY to the address of the first word in the address appointed by storage; store the contents in register RY+1 to the address of the second word in the address appointed by storage, and the like; store the contents in register RZ to the address of the last word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Influence on flag bit: No influence

Restriction: RZ should be greater than or equal to RY.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

32-bit

instruction

Operation:

Store contents in a group of consecutive register files to a group of consecutive storage addresses successively

```

src ← Y; addr ← RX;
for (n = 0; n <= IMM5; n++) {
    MEM[addr] ← Rsrc;
    src ← src + 1;
    addr ← addr + 4;
}

```

Grammar: stm32 ry-rz, (rx)

Description: Store contents in a group of consecutive register files starting from RY to a group of consecutive storage addresses successively. In another word, store contents in register RY to the address of the first word in the address appointed by storage; store the contents in register RY+1 to the address of the second word in the address appointed by storage, and the like; store the contents in register RZ to the address of the last word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Influence on flag No influence

bit:

Restriction: RZ should be greater than or equal to RY.

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	1	0	1	RY	RX	0	0	0	1	1

IMM5 field – Assign the number of destination registers, IMM5 = Z – Y.

00000 – 1 destination register

00001 – 2 destination registers

.....

11111 – 32 destination registers

STOP – Enter low power consumption stop mode

Unified instruction

Grammar	Operation	Compiling result
stop	Enter low power consumption stop mode	Only 32-bit instructions exist. stop32

- Description:** This instruction makes the processor enter low power consumption mode and wait for an interrupt to exit from this mode. At this time, CPU clock is stopped and corresponding peripheral equipment is also stopped.
- Influence on flag bit:** No influence
- Exception:** Privilege violation exception

32-bit instruction

- Operation:** Enter low power consumption stop mode
- Grammar:** stop32
- Attribute:** Privileged instruction
- Description:** This instruction makes the processor enter low power consumption mode and wait for an interrupt to exit from this mode. At this time, CPU clock is stopped and corresponding peripheral equipment is also stopped.
- Influence on flag bit:** No influence
- Exception:** Privilege violation exception
- Instruction format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 0 1 0	0 0 0 0 1	0 0 0 0 0	0

STQ – Store consecutive quad word#

**Unified
instruction**

Grammar	Operation	Compiling result
stq r4-r7, (rx)	<p>Store words in registers R4-R7 to a group of consecutive storage addresses successively</p> <pre>src ← 4; addr ← RX; for (n = 0; n <= 3; n++){ MEM[addr] ← Rsrc; src ← src + 1; addr ← addr + 4; }</pre>	<p>Only 32-bit instructions exist.</p> <p>stq32 r4-r7, (rx);</p>

Description: Store words in register file [R4,R7] (including boundary) to a group of consecutive storage addresses successively. In other word, store contents in register R4 to the address of the first word in the address appointed by storage; store contents in register R5 to the address of the second word in the address appointed by storage; store contents in register R6 to the address of the third word in the address appointed by storage; store contents in register R7 to the address of the fourth word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Attention: This instruction is the pseudo instruction of stm r4-r7, (rx).

**Influence on flag
bit:**

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

32-bit instruction

Operation: Store words in registers R4-R7 to a group of consecutive storage addresses successively

```
src ← 4; addr ← RX;
for (n = 0; n <= 3; n++){
```

```

    MEM[addr] ← Rsrc;
    src ← src + 1;
    addr ← addr + 4; }

```

Grammar: stq32 r4-r7, (rx)

Description: Store words in register file [R4,R7] (including boundary) to a group of consecutive storage addresses successively. In other word, store contents in register R4 to the address of the first word in the address appointed by storage; store contents in register R5 to the address of the second word in the address appointed by storage; store contents in register R6 to the address of the third word in the address appointed by storage; store contents in register R7 to the address of the fourth word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Attention: This instruction is the pseudo instruction of stm r4-r7, (rx).

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 1 0 1 0 0 1 0 0 RX 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1						

STR.B – Store byte in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
str.b rz, (rx, ry << 0)	Store the lowest byte in register to storage	Only 32-bit instructions exist.
str.b rz, (rx, ry << 1)	MEM[RX + RY << IMM2] ← RZ[7:0]	str32.b rz, (rx, ry << 0)
str.b rz, (rx, ry << 2)		str32.b rz, (rx, ry << 1)
str.b rz, (rx, ry << 3)		str32.b rz, (rx, ry << 2)
		str32.b rz, (rx, ry << 3)

- Description:** Store the lowest byte in register RZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value after offset register RY shifts left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.
- Influence on flag bit:** No influence
- Exception:** Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

32-bit instruction

- Operation:** Store the lowest byte in register to storage
 $MEM[RX + RY << IMM2] \leftarrow RZ[7:0]$
- Grammar:** str32.b rz, (rx, ry << 0)
 str32.b rz, (rx, ry << 1)
 str32.b rz, (rx, ry << 2)
 str32.b rz, (rx, ry << 3)
- Description:** Store the lowest byte in register RZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value after offset register RY shifts left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

str32.b rz, (rx, ry << 0)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 1	RY	RX	0 0 0 0 0 0	0 0 0 0 1	RZ						

str32.b rz, (rx, ry << 1)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 1	RY	RX	0 0 0 0 0 0	0 0 0 1 0	RZ						

str32.b rz, (rx, ry << 2)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 1	RY	RX	0 0 0 0 0 0	0 0 1 0 0	RZ						

str32.b rz, (rx, ry << 3)

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 1	RY	RX	0 0 0 0 0 0	0 1 0 0 0	RZ						

STR.H – Store half-word in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
str.h rz, (rx, ry << 0)	Store low half-word in register to storage	Only 32-bit instructions exist.
str.h rz, (rx, ry << 1)	MEM[RX + RY << IMM2] ← RZ[15:0]	str32.h rz, (rx, ry << 0)
str.h rz, (rx, ry << 2)		str32.h rz, (rx, ry << 1)
str.h rz, (rx, ry << 3)		str32.h rz, (rx, ry << 2)
		str32.h rz, (rx, ry << 3)

Description: Store low half-word in register RZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value after offset register RY shifts left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

32-bit instruction

Operation: Store low half-word in register to storage

MEM[RX + RY << IMM2] ← RZ[15:0]

Grammar:

- str32.h rz, (rx, ry << 0)
- str32.h rz, (rx, ry << 1)
- str32.h rz, (rx, ry << 2)
- str32.h rz, (rx, ry << 3)

Description: Store low half-word in register RZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value after offset register RY shifts left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

str32.h rz, (rx, ry << 0)

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 1 0 1	RY	RX	0 0 0 0 0 1	0 0 0 0 1	RZ	

str32.h rz, (rx, ry << 1)

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 1 0 1	RY	RX	0 0 0 0 0 1	0 0 0 1 0	RZ	

str32.h rz, (rx, ry << 2)

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 1 0 1	RY	RX	0 0 0 0 0 1	0 0 1 0 0	RZ	

str32.h rz, (rx, ry << 3)

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 1 0 1	RY	RX	0 0 0 0 0 1	0 1 0 0 0	RZ	

STR.W – Store word in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
str.w rz, (rx, ry << 0)	Store word in register to storage MEM[RX + RY << IMM2] ← RZ[31:0]	Only 32-bit instructions exist.
str.w rz, (rx, ry << 1)		str32.w rz, (rx, ry << 0)
str.w rz, (rx, ry << 2)		str32.w rz, (rx, ry << 1)
str.w rz, (rx, ry << 3)		str32.w rz, (rx, ry << 2)
		str32.w rz, (rx, ry << 3)

Description: Store word in register RZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value after offset register RY shifts left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

32-bit instruction

Operation: Store word in register to storage
MEM[RX + RY << IMM2] ← RZ[31:0]

Grammar: str32.w rz, (rx, ry << 0)
str32.w rz, (rx, ry << 1)
str32.w rz, (rx, ry << 2)
str32.w rz, (rx, ry << 3)

Description: Store word in register RZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value after offset register RY shifts left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB write invalid exception

Instruction

format:

str32.w rz, (rx, ry << 0)

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 1 0 1	RY	RX	0 0 0 0 1 0	0 0 0 0 1	RZ	
---	-----------	----	----	-------------	-----------	----	--

str32.w rz, (rx, ry << 1)

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 1 0 1	RY	RX	0 0 0 0 1 0	0 0 0 1 0	RZ	
---	-----------	----	----	-------------	-----------	----	--

str32.w rz, (rx, ry << 2)

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 1 0 1	RY	RX	0 0 0 0 1 0	0 0 1 0 0	RZ	
---	-----------	----	----	-------------	-----------	----	--

str32.w rz, (rx, ry << 3)

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 1 0 1	RY	RX	0 0 0 0 1 0	0 1 0 0 0	RZ	
---	-----------	----	----	-------------	-----------	----	--

STRAP – Enter safe state

**Unified
instruction**

Grammar	Operation	Compiling result
strap	Enter safe state SPC \leftarrow PC, SPSR \leftarrow PSR	Only 32-bit instructions exist. strap32

Attribute: Privileged instruction

Description: Save the current PC value in control register SPC and save the current PSR in control register SPSR; the program enters safe state and is executed from the safe entry address.

Influence on flag No influence

bit:

Exception: Privilege violation exception

32-bit instruction

Operation: Enter safe state

SPC \leftarrow PC, SPSR \leftarrow PSR

Grammar: strap32

Attribute: Privileged instruction

Description: Save the current PC value in control register SPC and save the current PSR in control register SPSR; the program enters safe state and is executed from the safe entry address.

Influence on flag No influence

bit:

Exception: Privilege violation exception

**Instruction
format:**

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	0	0	0	0	1	1	1	0

SUBC – Subtract with borrow unsigned

Unified instruction

Grammar	Operation	Compiling result
subc rz, rx	$RZ \leftarrow RZ - RX - (!C)$, $C \leftarrow \text{borrow}$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then subc16 rz, rx; else subc32 rz, rx;
subc rz, rx, ry	$RZ \leftarrow RX - RY - (!C)$, $C \leftarrow \text{borrow}$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x == z$) and ($y < 16$) and ($z < 16$), then subc16 rz, ry; else subc32 rz, rx, ry;

Description: For subc rz, rx, subtract the value of register RX and negative value of C bit from the value of RZ; for subc rz, rx, ry, subtract the value of register RY and negative value of C bit from the value of RX. Save the result in RZ and save borrow in C bit. For this subtract instruction, if borrow happens, C bit should be cleared; otherwise, C bit should be set.

Influence on flag $C \leftarrow \text{borrow}$

bit:

Exception: None

16-bit instruction

Operation: $RZ \leftarrow RZ - RX - (!C)$, $C \leftarrow \text{borrow}$

Grammar: subc16 rz, rx

Description: Subtract the value of register RX and negative value of C bit from the value of RZ, save the result in RZ, and save borrow in C bit. For this subtract instruction, if borrow happens, C bit should be cleared; otherwise, C bit should be set.

Influence on flag C \leftarrow borrow

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 0 0	RZ	RX	1 1

32-bit instruction

Operation: RZ \leftarrow RX - RY - (!C), C \leftarrow borrow

Grammar: subc32 rz, rx, ry

Description: Subtract the value of register RY and negative value of C bit from the value of RX, save the result in RZ, and save borrow in C bit. For this subtract instruction, if borrow happens, C bit should be cleared; otherwise, C bit should be set.

Influence on flag C \leftarrow borrow

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 0 0 0 0	0 1 0 0 0	RZ	

SUBI – Subtract immediate unsigned

Unified instruction

Grammar	Operation	Compiling result
subi rz, oimm12	$RZ \leftarrow RZ - \text{zero_extend(OIMM12)}$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if (oimm12<257) and (z<8), then subi16 rz, oimm8; else subi32 rz, rz, oimm12;
subi rz, rx, oimm12	$RZ \leftarrow RX - \text{zero_extend(OIMM12)}$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if (oimm12<8) and (z<8) and (x<8), then subi16 rz, rx, oimm3; elsif (x==z) and (z<8) and (oimm12<257), then subi16 rz, oimm8; else subi32 rz, rx, oimm12;

Description: Zero-extend the 12-bit immediate operand with offset 1 (OIMM12) to 32 bits, subtract this 32-bit number from RZ/RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x1-0x1000.

Exception: None

16-bit instruction----1

Operation: $RZ \leftarrow RZ - \text{zero_extend(OIMM8)}$

Grammar: subi16 rz, oimm8

Description: Zero-extend the 8-bit immediate operand with offset 1 (OIMM8) to 32 bits, subtract this 32-bit number from RZ value, and save the result in RZ.

Attention: The binary operand IMM8 is equal to OIMM8 – 1.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7; the range of immediate operand is 1-256.

Exception: None

Instruction

format:

15 14	11 10	8 7	0
-------	-------	-----	---

0	0 1 0 1	RZ	IMM8
---	---------	----	------

IMM8 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM8, the value OIMM8 subtracted from the register requires offset 1.

00000000 – -1

00000001 – -2

.....

11111111 – -256

16-bit

instruction----2

Operation: $RZ \leftarrow RX - \text{zero_extend}(OIMM3)$

Grammar: subi16 rz, rx, oimm3

Description: Zero-extend the 3-bit immediate operand with offset 1 (OIMM3) to 32 bits, subtract this 32-bit number from RX value, and save the result in RZ.

Attention: The binary operand IMM3 is equal to OIMM3 – 1.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7; the range of immediate operand is 1-8.

Exception: None

Instruction

format:

15 14	10	8 7	5 4	2 1 0
-------	----	-----	-----	-------

0	1 0 1 1	RX	RZ	IMM3	1 1
---	---------	----	----	------	-----

IMM3 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM3, the value OIMM3 subtracted from the register requires offset 1.

000 --1

001 --2

.....

111 --8

32-bit

instruction

Operation: $RZ \leftarrow RX - \text{zero_extend}(OIMM12)$

Grammar: subi32 rz, rx, oimm12

Description: Zero-extend the 12-bit immediate operand with offset 1 (OIMM12) to 32 bits, subtract this 32-bit number from RX value, and save the result in RZ.

Attention: The binary operand IMM12 is equal to OIMM12 – 1.

Influence on flag No influence

bit:

Restriction: The range of immediate operand is 0x1-0x1000.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	12 11	0
1	1 1 0 0 1	RZ	RX	0 0 0 1	IMM12

IMM12 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM12, the value OIMM12 subtracted from the register requires offset 1.

000000000000 --0x1

000000000001 --0x2

.....

111111111111 - -0x1000



SUBI(SP) – Subtract immediate unsigned (stack pointer)

Unified instruction

Grammar	Operation	Compiling result
subi sp, sp, imm	$SP \leftarrow SP - \text{zero_extend(IMM)}$	Only 16-bit instructions exist. subi sp, sp, imm

Description: Zero-extend the immediate operand (IMM) to 32 bits, make it shift left by 2 bits, subtract it from the value of stack pointer (SP), and save the result in SP.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0x1fc.

Exception: None

16-bit instruction

Operation: $SP \leftarrow SP - \text{zero_extend(IMM)}$

Grammar: subi sp, sp, imm

Description: Zero-extend the immediate operand (IMM) to 32 bits, make it shift left by 2 bits, subtract it from the value of stack pointer (SP), and save the result in stack pointer.

Attention: The immediate operand (IMM) is equal to the binary operand $\{\text{IMM2}, \text{IMM5}\} \ll 2$.

Influence on flag bit: No influence

Restriction: The source and destination registers are both stack instruction register (R14); the range of immediate operand is $(0x0-0x7f) \ll 2$.

Exception: None

Instruction format:

15 14 11 10 9 8 7 5 4 0

0	0	0	1	0	1	IMM2	0	0	1	IMM5	
---	---	---	---	---	---	------	---	---	---	------	--

IMM field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand {IMM2, IMM5}, the value IMM added into the register needs to shift left by 2 bits.

{00, 00000} – -0x0

{00, 00001} – -0x4

.....

{11, 11111} – -0x1fc

SUBU – Subtract unsigned

**Unified
instruction**

Grammar	Operation	Compiling result
subu rz, rx sub rz, rx	$RZ \leftarrow RZ - RX$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 16$) and ($x < 16$), then subu16 rz, rx; else subu32 rz, rx, rx;
subu rz, rx, ry	$RZ \leftarrow RX - RY$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($z < 8$) and ($x < 8$) and ($y < 8$), then subu16 rz, rx, ry; elsif ($x == z$) and ($z < 16$) and ($y < 16$), then subu16 rz, ry; else subu32 rz, rx, ry;

Description: For subu rz, rx, subtract RX value from RZ value and save the result in RZ.

For subu rz, rx, ry, subtract RY value from RX value and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

16-bit

instruction----1

Operation: $RZ \leftarrow RZ - RX$

Grammar: subu16 rz, rx

sub16 rz, rx

Description: Subtract RX value from RZ value and save the result in RZ.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14 10 9 6 5 2 1 0

0	1 1 0 0 0	RZ	RX	1 0
---	-----------	----	----	-----

16-bit

instruction----2

Operation: $RZ \leftarrow RX - RY$

Grammar: subu16 rz, rx, ry

sub16 rz, rx, ry

Description: Subtract RY value from RX value and save the result in RZ.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r7.

Exception: None

Instruction

format:

15 14 11 10 8 7 5 4 2 1 0

0	1 0 1 1	RX	RZ	RY	0 1
---	---------	----	----	----	-----

32-bit

instruction

Operation: $RZ \leftarrow RX - RY$

Grammar: subu32 rz, rx, ry

Description: Subtract RY value from RX value and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1 RY RX 0 0 0 0 0 0 0 0 1 0 0 RZ						

SYNC – Synchronize CPU

Unified instruction

Grammar	Operation	Compiling result
sync imm5	Synchronize CPU	Only 32-bit instructions exist. sync32 imm5

Description: When the processor meets sync instruction, the instruction will be suspended according to the indication range of immediate operand till all operations are completed. In another word, there is no instruction that is not completed.

The lowest bit of immediate operand (IMM5[0]) refers to the range of waiting for operation. If this bit is 0, the instruction will be suspended till all operations (including internal core, L2 Cache and bus) are completed. If this bit is 1, the instruction will be suspended till all operations in the core are completed.

Influence on flag bit: No influence

Exception: None

32-bit instruction

Operation: Synchronize CPU

Grammar: sync32 imm5

Description: When the processor meets sync instruction, the instruction will be suspended according to the indication range of immediate operand till all operations are completed. In another word, there is no instruction that is not completed.

The lowest bit of immediate operand (IMM5[0]) refers to the range of waiting for operation. If this bit is 0, the instruction will be suspended till all operations (including internal core, L2 Cache and bus) are completed. If this bit is 1, the instruction will be suspended till all operations in the core are completed.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0	IMM5	0 0 0 0 0	0 0 0 0 0 1	0 0 0 0 1	0 0 0 0 0	

TRAP – Operating system trap

**Unified
instruction**

Grammar	Operation	Description
trap 0, trap 1 trap 2, trap 3	Trigger trap exception	Only 32-bit instructions exist. trap32 0, trap32 1 trap32 2, trap32 3

Description: When the processor meets trap instruction, trap exception operation happens.

Influence on flag bit: No influence

Exception: Trap exception

**32-bit
instruction**

Operation: Trigger trap exception

Grammar: trap32 0,
trap32 1,
trap32 2,
trap32 3

Description: When the processor meets trap instruction, trap exception operation happens.

Influence on flag bit: No influence

Exception: Trap exception

**Instruction
format:**

trap32 0

31 30

26 25

21 20

16 15

10 9

5 4

0

1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 1 0 0 0	0 0 0 0 1	0 0 0 0 0
---	-----------	-----------	-----------	-------------	-----------	-----------

trap32 1

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 1 0 0 1	0 0 0 0 1	0 0 0 0 0
---	-----------	-----------	-----------	-------------	-----------	-----------

trap32 2

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 1 0 1 0	0 0 0 0 1	0 0 0 0 0
---	-----------	-----------	-----------	-------------	-----------	-----------

trap32 3

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 1 0 1 1	0 0 0 0 1	0 0 0 0 0
---	-----------	-----------	-----------	-------------	-----------	-----------

TST – Null-test

**Unified
instruction**

Grammar	Operation	Compiling result
tst rx, ry	If (RX & RY) != 0, then C ← 1; else C ← 0;	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if (x<16) and (y<16), then tst16 rx, ry; else tst32 rx, ry;

Description: Test the bitwise AND result of RX and RY values.
If the result is not equal to zero, set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: Set the condition bit C according to the bitwise AND result

Exception: None

**16-bit
instruction**

Operation: If (RX & RY) != 0, then

 C ← 1;

else

 C ← 0;

Grammar: tst16 rx, ry

Description: Test the bitwise AND result of RX and RY values.
If the result is not equal to zero, set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: Set the condition bit C according to the bitwise AND result

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 1 0	RY	RX	1 0

32-bit

instruction

Operation: If $(RX \& RY) \neq 0$, then

$C \leftarrow 1;$

else

$C \leftarrow 0;$

Grammar: tst32 rx, ry

Description: Test the bitwise AND result of RX and RY values.

If the result is not equal to zero, set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: Set the condition bit C according to the bitwise AND result

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 1 0 0 0	0 0 1 0 0	0 0 0 0 0	

TSTNBZ – Register test without byte equal to zero

Unified instruction

Grammar	Operation	Compiling result
tstnbz16 rx	If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then C ← 1; else C ← 0;	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if (x<16), then tstnbz16 rx; else tstnbz32 rx;

Description: Test whether there is byte equal to zero in RX. If there is no byte equal to zero in RX, set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: Set the condition bit C according to the bitwise AND result

Exception: None

16-bit instruction

Operation: If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then
C ← 1;
else
C ← 0;

Grammar: tstnbz16 rx

Description: Test whether there is byte equal to zero in RX. If there is no byte equal to zero in RX, set the condition bit C; otherwise, clear the condition bit C.

Influence on flag bit: Set the condition bit C according to the bitwise AND result

Restriction: The range of register is r0-r15.

Exception: None

Instruction
format:

15 14	10 9	6 5	2 1 0
0 1 1 0 1 0	0 0 0 0	RX	1 1

32-bit
instruction

Operation: If ((RX[31:24] != 0) &(RX[23:16] != 0) &(RX[15: 8] != 0) &(RX[7 : 0] != 0)), then

C ← 1;

else

C ← 0;

Grammar: tstnbz32 rx

Description: Test whether there is byte equal to zero in RX. If there is no byte equal to zero in RX, set the condition bit C; otherwise, clear the condition bit C.

Influence on flag Set the condition bit C according to the bitwise AND result

bit:

Exception: None

Instruction
format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0 1	0 0 0 0 0 0	RX	0 0 1 0 0 0	0 1 0 0 0	0 0 0 0 0	

VMULSH – 16-bit multiply signed in two branches

Unified instruction

Grammar	Operation	Compiling result
vmulsh rx, ry	<p>Multiply the 16-bit signed numbers in two branches respectively and put the result in accumulator</p> $HI \leftarrow RX[31:16] \times RY[31:16]$ $LO \leftarrow RX[15:0] \times RY[15:0]$	<p>Only 32-bit instructions exist.</p> <p>vmulsh32 rx, ry</p>

Description: Save the result of multiplying the 16 high bits of general-purpose register RX and 16 high bits of RY in the high-bit accumulator register HI. Save the result of multiplying the 16 low bits of general-purpose register RX and 16 low bits of RY in the low-bit accumulator register LO.

All contents in the registers are considered as signed numbers. The sign bits of source operands in register RX and RY are the 31st bit and the 15th bit of the register respectively; the sign bit of results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Influence on flag Overflow bit is cleared

bit:

Exception: None

32-bit instruction

Operation: Multiply the 16-bit signed numbers in two branches respectively and put the result in accumulator

$$HI \leftarrow RX[31:16] \times RY[31:16]$$

$$LO \leftarrow RX[15:0] \times RY[15:0]$$

Grammar: vmulsh32 rx, ry

Description: Save the result of multiplying the 16 high bits of general-purpose register RX and 16 high bits of RY in the high-bit accumulator register HI. Save the result of multiplying the 16 low bits of general-purpose register RX and 16 low bits of RY in the low-bit accumulator register LO.

All contents in the registers are considered as signed numbers. The sign

bits of source operands in register RX and RY are the 31st bit and the 15th bit of the register respectively; the sign bit of results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Influence on flag Overflow bit is cleared

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	1 0 1 1 0 0	0 0 0 0 1	0 0 0 0 0	

VMULSHA – 16-bit multiply-accumulate signed in two branches

**Unified
instruction**

Grammar	Operation	Compiling result
vmulsha rx, ry	<p>Multiply the 16-bit signed numbers in two branches respectively, add the results to the high-bit and low-bit values of accumulator respectively, and put the result in accumulator</p> $HI \leftarrow HI + RX[31:16] \times RY[31:16]$ $LO \leftarrow LO + RX[15:0] \times RY[15:0]$	<p>Only 32-bit instructions exist.</p> <p>vmulsha32 rx, ry</p>

Description: Add the 32-bit result of multiplying the 16 high bits of general-purpose register RX and 16 high bits of RY to the value in 32-high-bit register HI of 64-bit accumulator, and save the result in the high-bit accumulator register HI. Add the 32-bit result of multiplying the 16 low bits of general-purpose register RX and 16 low bits of RY to the value in 32-low-bit register LO of 64-bit accumulator, and save the result in the low-bit accumulator register LO.

All contents in register RX and RY as well as low-bit accumulator register LO and high-bit register LO are considered as signed numbers. The sign bits of source operands in register RX and RY are the 31st bit and the 15th bit of the register respectively; the sign bit of operands and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Each branch of this instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Overflow of any branch will set the overflow bit as 1

Exception: None

instruction

Operation: Multiply the 16-bit signed numbers in two branches respectively, add the results to the high-bit and low-bit values of accumulator respectively, and put the result in accumulator

$$HI \leftarrow HI + RX[31:16] \times RY[31:16]$$

$$LO \leftarrow LO + RX[15:0] \times RY[15:0]$$

Grammar: vmulsha32 rx, ry

Description: Add the 32-bit result of multiplying the 16 high bits of general-purpose register RX and 16 high bits of RY to the value in 32-high-bit register HI of 64-bit accumulator, and save the result in the high-bit accumulator register HI. Add the 32-bit result of multiplying the 16 low bits of general-purpose register RX and 16 low bits of RY to the value in 32-low-bit register LO of 64-bit accumulator, and save the result in the low-bit accumulator register LO.

All contents in register RX and RY as well as low-bit accumulator register LO and high-bit register LO are considered as signed numbers. The sign bits of source operands in register RX and RY are the 31st bit and the 15th bit of the register respectively; the sign bit of operands and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Each branch of this instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Overflow of any branch will set the overflow bit as 1

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RY	RX	1	0	1	1	0

VMULSHS – 16-bit multiply-subtract signed in two branches

Unified instruction

Grammar	Operation	Compiling result
vmulshs rx, ry	<p>Subtract the value of multiplying the 16-bit signed numbers in two branches from the high-bit and low-bit values of accumulator respectively, and put the result in accumulator</p> $HI \leftarrow HI - RX[31:16] \times RY[31:16]$ $LO \leftarrow LO - RX[15:0] \times RY[15:0]$	<p>Only 32-bit instructions exist.</p> <p>vmulshs32 rx, ry</p>

Description: Subtract the value of multiplying the 16 high bits of general-purpose register RX and 16 high bits of RY from the value in 32-high-bit register HI of 64-bit accumulator, and save the result in the high-bit accumulator register HI. Subtract the value of multiplying the 16 low bits of general-purpose register RX and 16 low bits of RY from the value in 32-low-bit register LO of 64-bit accumulator, and save the result in the low-bit accumulator register LO.

All contents in register RX and RY as well as low-bit accumulator register LO and high-bit register LO are considered as signed numbers. The sign bits of source operands in register RX and RY are the 31st bit and the 15th bit of the register respectively; the sign bit of operands and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Each branch of this instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Overflow of any branch will set the overflow bit as 1

Exception: None

32-bit instruction

Operation: Subtract the value of multiplying the 16-bit signed numbers in two

branches from the high-bit and low-bit values of accumulator respectively, and put the result in accumulator

$$HI \leftarrow HI - RX[31:16] \times RY[31:16]$$

$$LO \leftarrow LO - RX[15:0] \times RY[15:0]$$

Grammar: vmulshs32 rx, ry

Description: Subtract the value of multiplying the 16 high bits of general-purpose register RX and 16 high bits of RY from the value in 32-high-bit register HI of 64-bit accumulator, and save the result in the high-bit accumulator register HI. Subtract the value of multiplying the 16 low bits of general-purpose register RX and 16 low bits of RY from the value in 32-low-bit register LO of 64-bit accumulator, and save the result in the low-bit accumulator register LO.

All contents in register RX and RY as well as low-bit accumulator register LO and high-bit register LO are considered as signed numbers. The sign bits of source operands in register RX and RY are the 31st bit and the 15th bit of the register respectively; the sign bit of operands and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Each branch of this instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Overflow of any branch will set the overflow bit as 1

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RY	RX	1	0	1	1	0

VMULSW – 16x32 multiply signed in two branches

Unified instruction

Grammar	Operation	Compiling result
vmulsw rx, ry	<p>Multiply 16-bit signed numbers in two branches with a 32-bit signed number respectively, and put the result in accumulator</p> $HI \leftarrow (RX[31:16] \times RY[31:0])[47:16]$ $LO \leftarrow (RX[15:0] \times RY[31:0])[47:16]$	<p>Only 32-bit instructions exist.</p> <p>vmulsw32 rx, ry</p>

Description: Save the 32 high bits of the result after multiplying the 16 high bits of general-purpose register RX and contents of RY in high-bit accumulator register HI. Save the 32 high bits of the result after multiplying the 16 low bits of general-purpose register RX and contents of RY in low-bit accumulator register LO.

All contents in the registers are considered as signed numbers. The sign bits of source operands in register RX are the 31st bit and the 15th bit of the register respectively; the sign bit of operand in register RY and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Influence on flag bit: Overflow bit is cleared

Exception: None

32-bit instruction

Operation: Multiply 16-bit signed numbers in two branches with a 32-bit signed number respectively, and put the result in accumulator

$$HI \leftarrow (RX[31:16] \times RY[31:0])[47:16]$$

$$LO \leftarrow (RX[15:0] \times RY[31:0])[47:16]$$

Grammar:

Description: Save the 32 high bits of the result after multiplying the 16 high bits of general-purpose register RX and contents of RY in high-bit accumulator

register HI. Save the 32 high bits of the result after multiplying the 16 low bits of general-purpose register RX and contents of RY in low-bit accumulator register LO.

All contents in the registers are considered as signed numbers. The sign bits of source operands in register RX are the 31st bit and the 15th bit of the register respectively; the sign bit of operand in register RY and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Influence on flag Overflow bit is cleared

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	1 0 1 1 0 1	0 0 0 0 1	0 0 0 0 0	

VMULSWA – 16x32 multiply-accumulate signed in two branches

**Unified
instruction**

Grammar	Operation	Compiling result
vmulswa rx, ry	<p>Multiply 16-bit signed numbers in two branches with a 32-bit signed number respectively, add the results to the high-bit and low-bit values of accumulator respectively, and put the result in accumulator</p> $HI \leftarrow HI + (RX[31:16] \times RY[31:0])[47:16]$ $LO \leftarrow LO + (RX[15:0] \times RY[31:0])[47:16]$	<p>Only 32-bit instructions exist.</p> <p>vmulswa32 rx, ry</p>
Description:	<p>Add the 32 high bits of the result after multiplying the 16 high bits of general-purpose register RX and contents of RY to the value in 32-high-bit register HI of 64-bit accumulator, and save the result in high-bit accumulator register HI. Add the 32 high bits of the result after multiplying the 16 low bits of general-purpose register RX and contents of RY to the value in 32-low-bit register LO of 64-bit accumulator, and save the result in low-bit accumulator register LO.</p> <p>All contents in the registers are considered as signed numbers. The sign bits of source operands in register RX are the 31st bit and the 15th bit of the register respectively; the sign bit of operand in register RY and results in high-bit register HI and low-bit register LO is the 31st bit of the register.</p> <p>Each branch of this instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.</p>	
Influence on flag bit:	<p>Overflow of any branch will set the overflow bit as 1</p>	
Exception:	<p>None</p>	

32-bit

instruction

Operation: Multiply 16-bit signed numbers in two branches with a 32-bit signed number respectively, add the results to the high-bit and low-bit values of accumulator respectively, and put the result in accumulator

$$HI \leftarrow HI + (RX[31:16] \times RY[31:0])[47:16]$$

$$LO \leftarrow LO + (RX[15:0] \times RY[31:0])[47:16]$$

Grammar: vmulswa32 rx, ry

Description: Add the 32 high bits of the result after multiplying the 16 high bits of general-purpose register RX and contents of RY to the value in 32-high-bit register HI of 64-bit accumulator, and save the result in high-bit accumulator register HI. Add the 32 high bits of the result after multiplying the 16 low bits of general-purpose register RX and contents of RY to the value in 32-low-bit register LO of 64-bit accumulator, and save the result in low-bit accumulator register LO.

All contents in the registers are considered as signed numbers. The sign bits of source operands in register RX are the 31st bit and the 15th bit of the register respectively; the sign bit of operand in register RY and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Each branch of this instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag Overflow of any branch will set the overflow bit as 1
bit:

Exception: None

Instruction
format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RY	RX	1	0	1	1	0

VMULSWS – 16x32 multiply-subtract signed in two branches

Unified instruction

Grammar	Operation	Compiling result
vmuls ws rx, ry	<p>Subtract the 32 high bits of the result after multiplying the 16-bit signed numbers in two branches with a 32-bit signed number from the high-bit and low-bit values of accumulator respectively, and put the result in accumulator</p> $HI \leftarrow HI - (RX[31:16] \times RY[31:0])[47:16]$ $LO \leftarrow LO - (RX[15:0] \times RY[31:0])[47:16]$	<p>Only 32-bit instructions exist.</p> <p>vmuls ws32 rx, ry</p>

Description: Subtract the 32 high bits of the result after multiplying the 16 high bits of general-purpose register RX and contents of RY from the value in 32-high-bit register HI of 64-bit accumulator, and save the result in high-bit accumulator register HI. Subtract the 32 high bits of the result after multiplying the 16 low bits of general-purpose register RX and contents of RY from the value in 32-low-bit register LO of 64-bit accumulator, and save the result in low-bit accumulator register LO.

All contents in the registers are considered as signed numbers. The sign bits of source operands in register RX are the 31st bit and the 15th bit of the register respectively; the sign bit of operand in register RY and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Each branch of this instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Overflow of any branch will set the overflow bit as 1

Exception: None

32-bit instruction

Operation: Subtract the 32 high bits of the result after multiplying the 16-bit signed

numbers in two branches with a 32-bit signed number from the high-bit and low-bit values of accumulator respectively, and put the result in accumulator

$$HI \leftarrow HI - (RX[31:16] \times RY[31:0])[47:16]$$

$$LO \leftarrow LO - (RX[15:0] \times RY[31:0])[47:16]$$

Grammar:

vmulsws32 rx, ry

Description:

Subtract the 32 high bits of the result after multiplying the 16 high bits of general-purpose register RX and contents of RY from the value in 32-high-bit register HI of 64-bit accumulator, and save the result in high-bit accumulator register HI. Subtract the 32 high bits of the result after multiplying the 16 low bits of general-purpose register RX and contents of RY from the value in 32-low-bit register LO of 64-bit accumulator, and save the result in low-bit accumulator register LO.

All contents in the registers are considered as signed numbers. The sign bits of source operands in register RX are the 31st bit and the 15th bit of the register respectively; the sign bit of operand in register RY and results in high-bit register HI and low-bit register LO is the 31st bit of the register.

Each branch of this instruction supports 8 guard bits. See the descriptions about guard bit in the processor manual for more details.

Influence on flag bit: Overflow of any branch will set the overflow bit as 1

Exception: None

Instruction format:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RY	RX	1	0	1	1	0

WAIT – Enter low power consumption wait mode

**Unified
instruction**

Grammar	Operation	Compiling result
wait	Enter low power consumption wait mode	Only 32-bit instructions exist. wait32

Attribute: Privileged instruction

Description: This instruction will stop execution of the current instruction and waits for an interrupt. At this time, CPU clock is stopped. The peripheral equipment is still in operation. Besides, interrupt might be caused, which will make CPU exit from wait mode.

Influence on flag bit: No influence

Exception: Privilege violation instruction

**32-bit
instruction**

Operation: Enter low power consumption wait mode

Grammar: wait32

Attribute: Privileged instruction

Description: This instruction will stop execution of the current instruction and waits for an interrupt. At this time, CPU clock is stopped. The peripheral equipment is still in operation. Besides, interrupt might be caused, which will make CPU exit from wait mode.

Influence on flag bit: No influence

Exception: Privilege violation instruction

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
-------	-------	-------	-------	------	-----	---

1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 0 1 1	0 0 0 0 1	0 0 0 0 0
---	-----------	-----------	-----------	-------------	-----------	-----------



WE – Wait event

**Unified
instruction**

Grammar	Operation	Compiling result
we	Wait event	Only 32-bit instructions exist. we32

Description: WE instruction is executed and CPU enters a low power consumption mode of waiting event. It will be aroused when other processor cores send event.

Influence on flag bit: No influence

Exception: Privilege violation instruction

**32-bit
instruction**

Operation: Wait event

Grammar: we32

Description: WE instruction is executed and CPU enters a low power consumption mode of waiting event. It will be aroused when other processor cores send event.

Influence on flag bit: No influence

Exception: Privilege violation instruction

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 1 0 1	0 0 0 0 1	0 0 0 0 0	

XOR – Bitwise XOR

Unified instruction

Grammar	Operation	Compiling result
xor rz, rx	$RZ \leftarrow RZ \wedge RX$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then xor16 rz, rx; else xor32 rz, rz, rx;
xor rz, rx, ry	$RZ \leftarrow RX \wedge RY$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($y == z$) and ($z < 16$) and ($x < 16$), then xor16 rz, rx; else xor32 rz, rx, ry;

Description: Perform a bitwise XOR of RX and RZ/RY values and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow RZ \wedge RX$

Grammar: xor16 rz, rx

Description: Perform a bitwise XOR of RZ and RX values and save the result in RZ.

Influence on flag No influence

bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction

format:

15 14	10 9	6 5	2 1 0
0 1 1 0 1 1	RZ	RX	0 1

32-bit

instruction

Operation: $RZ \leftarrow RX \wedge RY$

Grammar: xor32 rz, rx, ry

Description: Perform a bitwise XOR of RX and RY values and save the result in RZ.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	RY	RX	0 0 1 0 0 1	0 0 0 1 0	RZ	

XORI – Bitwise XOR immediate

Unified instruction

Grammar	Operation	Compiling result
xori rz, rx, imm16	$RZ \leftarrow RX \wedge \text{zero_extend(IMM12)}$	Only 32-bit instructions exist. xori32 rz, rx, imm12

Description: Zero-extend the 12-bit immediate operand to 32 bits, perform a bitwise XOR with RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

32-bit

instruction

Operation: $RZ \leftarrow RX \wedge \text{zero_extend(IMM12)}$

Grammar: xori32 rz, rx, imm12

Description: Zero-extend the 12-bit immediate operand to 32 bits, perform a bitwise XOR with RX value, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of immediate operand is 0x0-0xFFFF.

Exception: None

Instruction

format:

31 30 26 25 21 20 16 15 12 11 0

1	1 1 0 0 1	RZ	RX	0 1 0 0	IMM12
---	-----------	----	----	---------	-------

XSR – Extended shift right

Unified instruction

Grammar	Operation	Compiling result
xsr rz, rx, oimm5	$\{RZ,C\} \leftarrow \{RX,C\} >>> OIMM5$	Only 32-bit instructions exist. xsr32 rz, rx, oimm5

Description: Perform a ring right shift on RX value with condition bit C ($\{RX,C\}$) (the original value shifts right and the bit shifting out from right side will shift to the left side), save the lowest bit ([0]) of the shifting result in C bit, and save the highest bit ([32:1]) in RZ; the range of right shift is decided by the value of 5-bit immediate operand with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, condition bit C is the highest bit of RX.

Influence on flag bit: $C \leftarrow RX[OIMM5 - 1]$

Restriction: The range of immediate operand is 1-32.

Exception: None

32-bit instruction

Operation: $\{RZ,C\} \leftarrow \{RX,C\} >>> OIMM5$

Grammar: xsr32 rz, rx, oimm5

Description: Perform a ring right shift on RX value with condition bit C ($\{RX,C\}$) (the original value shifts right and the bit shifting out from right side will shift to the left side), save the lowest bit ([0]) of the shifting result in C bit, and save the highest bit ([32:1]) in RZ; the range of right shift is decided by the value of 5-bit immediate operand with offset 1 (OIMM5). If the value of OIMM5 is equal to 32, condition bit C is the highest bit of RX.

Attention: The binary operand IMM5 is equal to OIMM5 – 1.

Influence on flag bit: $C \leftarrow RX[OIMM5 - 1]$

Restriction: The range of immediate operand is 1-32.

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	IMM5	RX	0 1 0 0 1 1	0 1 0 0 0	RZ	

IMM5 field – Assign the value of immediate operand without offset.

Attention: Compared with the binary operand IMM5, the shifting value OIMM5 requires offset 1.

00000 – shift by 1 bit

00001 – shift by 2 bits

.....

11111 – shift by 32 bits

XTRB0 – Extract byte 0 and extend unsigned

**Unified
instruction**

Grammar	Operation	Compiling result
xtrb0 rz, rx	RZ \leftarrow zero_extend(RX[31:24]); if (RX[31:24] == 0), then C \leftarrow 0; else C \leftarrow 1;	Only 32-bit instructions exist. xtrb0.32 rz, rx

Description: Extract byte 0 of RX (RX[31:24]) to the low bit of RZ (RZ[7:0]), and conduct zero-extension. If the result is equal to zero, clear C bit; otherwise, set C bit.

Influence on flag bit: If the result is equal to zero, clear C bit; otherwise, set C bit.

Exception: None

**32-bit
instruction**

Operation: RZ \leftarrow zero_extend(RX[31:24]);
if (RX[31:24] == 0), then
 C \leftarrow 0;
else
 C \leftarrow 1;

Grammar: xtrb0.32 rz, rx

Description: Extract byte 0 of RX (RX[31:24]) to the low bit of RZ (RZ[7:0]), and conduct zero-extension. If the result is equal to zero, clear C bit; otherwise, set C bit.

Influence on flag bit: If the result is equal to zero, clear C bit; otherwise, set C bit.

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 0 0	0 0 0 0 1	RZ



XTRB1 – Extract byte 1 and extend unsigned

**Unified
instruction**

Grammar	Operation	Compiling result
xtrb1 rz, rx	RZ \leftarrow zero_extend(RX[23:16]); if (RX[23:16] == 0), then C \leftarrow 0; else C \leftarrow 1;	Only 32-bit instructions exist. xtrb1.32 rz, rx

Description: Extract byte 1 of RX (RX[23:16]) to the low bit of RZ (RZ[7:0]), and conduct zero-extension. If the result is equal to zero, clear C bit; otherwise, set C bit.

Influence on flag bit: If the result is equal to zero, clear C bit; otherwise, set C bit.

Exception: None

**32-bit
instruction**

Operation: RZ \leftarrow zero_extend(RX[23:16]);
if (RX[23:16] == 0), then
 C \leftarrow 0;
else
 C \leftarrow 1;

Grammar: xtrb1.32 rz, rx

Description: Extract byte 1 of RX (RX[23:16]) to the low bit of RZ (RZ[7:0]), and conduct zero-extension. If the result is equal to zero, clear C bit; otherwise, set C bit.

Influence on flag bit: If the result is equal to zero, clear C bit; otherwise, set C bit.

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 0 0	0 0 0 1 0	RZ



XTRB2 – Extract byte 2 and extend unsigned

Unified instruction

Grammar	Operation	Compiling result
xtrb2 rz, rx	RZ \leftarrow zero_extend(RX[15:8]); if (RX[15:8] == 0), then C \leftarrow 0; else C \leftarrow 1;	Only 32-bit instructions exist. xtrb2.32 rz, rx

Description: Extract byte 2 of RX (RX[15:8]) to the low bit of RZ (RZ[7:0]), and conduct zero-extension. If the result is equal to zero, clear C bit; otherwise, set C bit.

Influence on flag bit: If the result is equal to zero, clear C bit; otherwise, set C bit.

Exception: None

32-bit instruction

Operation: RZ \leftarrow zero_extend(RX[15:8]);
if (RX[15:8] == 0), then
 C \leftarrow 0;
else
 C \leftarrow 1;

Grammar: xtrb2.32 rz, rx

Description: Extract byte 2 of RX (RX[15:8]) to the low bit of RZ (RZ[7:0]), and conduct zero-extension. If the result is equal to zero, clear C bit; otherwise, set C bit.

Influence on flag bit: If the result is equal to zero, clear C bit; otherwise, set C bit.

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 0 0	0 0 1 0 0	RZ



XTRB3 – Extract byte 3 and extend unsigned

Unified instruction

Grammar	Operation	Compiling result
xtrb3 rz, rx	RZ \leftarrow zero_extend(RX[7:0]); if (RX[7:0] == 0), then C \leftarrow 0; else C \leftarrow 1;	Only 32-bit instructions exist. xtrb3.32 rz, rx

Description: Extract byte 3 of RX (RX[7:0]) to the low bit of RZ (RZ[7:0]), and conduct zero-extension. If the result is equal to zero, clear C bit; otherwise, set C bit.

Influence on flag bit: If the result is equal to zero, clear C bit; otherwise, set C bit.

Exception: None

32-bit instruction

Operation: RZ \leftarrow zero_extend(RX[7:0]);
if (RX[7:0] == 0), then
 C \leftarrow 0;
else
 C \leftarrow 1;

Grammar: xtrb3.32 rz, rx

Description: Extract byte 3 of RX (RX[7:0]) to the low bit of RZ (RZ[7:0]), and conduct zero-extension. If the result is equal to zero, clear C bit; otherwise, set C bit.

Influence on flag bit: If the result is equal to zero, clear C bit; otherwise, set C bit.

Exception: None

**Instruction
format:**

31 30	26 25	21 20	16 15	10 9	5 4	0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 0 0	0 1 0 0 0	RZ



ZEXT – Extract bit and extend unsigned

Unified instruction

Grammar	Operation	Compiling result
zext rz, rx, msb, lsb	$RZ \leftarrow \text{zero_extend}(RX[\text{MSB:LSB}])$	Only 32-bit instructions exist. zext32 rz, rx, msb, lsb

Description: Extract a section of consecutive bits of RX (RX[MSB:LSB]) appointed by 2 5-bit immediate operands (MSB,LSB), sign-extend it to 32 bits, and save the result in RZ. If MSB is equal to 31 and LSB is equal to zero, RZ value is the same with RX value. If MSB is equal to LSB, RZ value is the result after one-bit zero-extension of RX[MSB] (i.e. RX[LSB]). If MSB is smaller than LSB, behavior of this instruction is unpredictable.

Influence on flag bit: No influence

Restriction: The range of MSB is 0-31, the range of LSB is 0-31, and MSB should be greater than or equal to LSB.

Exception: None

32-bit

instruction

Operation: $RZ \leftarrow \text{zero_extend}(RX[\text{MSB:LSB}])$

Grammar: zext32 rz, rx, msb, lsb

Description: Extract a section of consecutive bits of RX (RX[MSB:LSB]) appointed by 2 5-bit immediate operands (MSB,LSB), sign-extend it to 32 bits, and save the result in RZ. If MSB is equal to 31 and LSB is equal to zero, RZ value is the same with RX value. If MSB is equal to LSB, RZ value is the result after one-bit zero-extension of RX[MSB] (i.e. RX[LSB]). If MSB is smaller than LSB, behavior of this instruction is unpredictable.

Influence on flag bit: No influence

Restriction: The range of MSB is 0-31, the range of LSB is 0-31, and MSB should be greater than or equal to LSB.

Exception: None

Instruction

format:

31	30	26	25	21	20	16	15	10	9	5	4	0	
1	1	0	0	0	1	LSB	RX	0	1	0	1	MSB	RZ

MSB field – Assign the starting bit of LSB field – Assign the end bit of extraction.

00000 – 0

00000 – 0 bit

00001 – 1

00001 – 1 bit

.....

.....

11111 – 31

11111 – 31 bits

ZEXTB – Extract byte and extend unsigned#

Unified instruction

Grammar	Operation	Compiling result
zextb rz, rx	$RZ \leftarrow \text{zero_extend}(RX[7:0]);$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then zextb16 rz, rx; else zextb32 rz, rx

Description: Zero-extend low byte of RX (RX[7:0]) to 32 bits and save the result in RZ.

Influence on flag bit: No influence

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow \text{zero_extend}(RX[7:0]);$

Grammar: zextb16 rz, rx

Description: Zero-extend low byte of RX (RX[7:0]) to 32 bits and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:

15	14	10	9	6	5	2	1	0
0	1	1	1	0	1	RZ	RX	0 0

32-bit instruction

Operation: $RZ \leftarrow \text{zero_extend}(RX[7:0]);$

Grammar: zextb32 rz, rx

Description: Zero-extend low byte of RX (RX[7:0]) to 32 bits and save the result in RZ.

Attention: This instruction is the pseudo instruction of zext32 rz, rx,
0x7, 0x0.

Influence on flag No influence

bit:

Exception: None

Instruction

format:

31 30	26 25	21 20	16 15	10 9	5 4	0
1 1 0 0 0 1	0 0 0 0 0	RX	0 1 0 1 0 1	0 0 1 1 1	RZ	

ZEXTH – Extract half-word and extend unsigned#

Unified instruction

Grammar	Operation	Compiling result
zexth rz, rx	$RZ \leftarrow \text{zero_extend}(RX[15:0]);$	Compiled into corresponding 16-bit or 32-bit instructions according to the range of register. if ($x < 16$) and ($z < 16$), then zexth16 rz, rx; else zexth32 rz, rx

Description: Zero-extend low half-word of RX (RX[15:0]) to 32 bits and save the result in RZ.

Influence on flag bit: No influence

Exception: None

16-bit

instruction

Operation: $RZ \leftarrow \text{zero_extend}(RX[15:0]);$

Grammar: zexth16 rz, rx

Description: Zero-extend low half-word of RX (RX[15:0]) to 32 bits and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format



32-bit instruction

Operation:	RZ \leftarrow zero_extend(RX[15:0]);
Grammar:	zext32 rz, rx
Description:	Zero-extend low half-word of RX (RX[15:0]) to 32 bits and save the result in RZ.
	Attention: This instruction is the pseudo instruction of zext32 rz, rx, 0x15, 0x0.
Influence on flag bit:	No influence
Exception:	None
Instruction format:	

31 30	26 25	21 20	16 15	10 9	5 4	0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 0 1 0 1	0 1 1 1 1	RZ

6. Term list of floating point instructions

Specific descriptions of each floating point instruction realized by CK810 are provided in the following and each instruction is described in details according to the alphabetical order. The floating point instructions of CK810 are 32-bit instructions and independent vector general-purpose register (VR) is adopted.



FABSD – Double-precision floating point absolute value

Unified instruction

Grammar	Operation	Compiling result
fabsd vrz, vrx	$vrz = vrx $	Only 32-bit instructions exist. fabsd vrz, vrx

Data type Double-precision floating point

Description: Take the absolute value of double-precision floating point in vrx and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = |vrx|$

Grammar: fabsd vrz, vrx

Data type Double-precision floating point

Description: Take the absolute value of double-precision floating point in vrx and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	VRX	0	0

CSKY

FABSM – SIMD single-precision floating point absolute value

Unified instruction

Grammar	Operation	Compiling result
fabsm vrz, vrx	$vrz[31:0] = vrx[31:0] ; vrz[63:32] = vrx[63:32] $	Only 32-bit instructions exist. fabsm vrz, vrx

Data type Single-precision floating point

Description: As single instruction multiple data (SIMD), this instruction takes the absolute value of single-precision floating point in vrx[31:0] and saves the result in vrz[31:0]; it takes the absolute value of single-precision floating point in vrx[63:32] and saves the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz[31:0] = |vrx[31:0]|; vrz[63:32] = |vrx[63:32]|$

Grammar: fabsm vrz, vrx

Data type Single-precision floating point

Description: As single instruction multiple data (SIMD), this instruction takes the absolute value of single-precision floating point in vrx[31:0] and saves the result in vrz[31:0]; it takes the absolute value of single-precision floating point in vrx[63:32] and saves the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	VRX	0	0

CSKY

FABSS – Single-precision floating point absolute value

Unified instruction

Grammar	Operation	Compiling result
fabss vrz, vrx	$vrz = vrx $	Only 32-bit instructions exist. fabss vrz, vrx

Data type Single-precision floating point

Description: Take the absolute value of single-precision floating point in vrx and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = |vrx|$

Grammar: fabss vrz, vrx

Data type Single-precision floating point

Description: Take the absolute value of single-precision floating point in vrx and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	VRX	0	0

CSKY

FADDD – Double-precision floating point add

Unified instruction

Grammar	Operation	Compiling result
faddd vrz, vrx,vry	$vrz = vrx + vry$	Only 32-bit instructions exist. faddd vrz, vrx,vry

Data type Double-precision floating point

Description: Add double-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrx + vry$

Grammar: faddd vrz, vrx,vry

Data type Double-precision floating point

Description: Add double-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

CSKY

FADDM – SIMD single-precision floating point add

Unified instruction

Grammar	Operation	Compiling result
faddm vrz, vrx,vry	$vrz[31:0] = vrx[31:0] + vry[31:0];$ $vrz[63:32] = vrx[63:32] + vry[63:32]$	Only 32-bit instructions exist. faddm vrz, vrx,vry

Data type Single-precision floating point

Description: Add single-precision floating points in vrx[31:0] and vry[31:0], and save the result in vrz[31:0]; add single-precision floating points in vrx[63:32] and vry[63:32], and save the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz[31:0] = vrx[31:0] + vry[31:0]; vrz[63:32] = vrx[63:32] + vry[63:32]$

Grammar: faddm vrz, vrx,vry

Data type Single-precision floating point

Description: Add single-precision floating points in vrx[31:0] and vry[31:0], and save the result in vrz[31:0]; add single-precision floating points in vrx[63:32] and vry[63:32], and save the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

CSKY

FADDS – Single-precision floating point add

Unified instruction

Grammar	Operation	Compiling result
fadds vrz, vrx,vry	$vrz = vrx + vry$	Only 32-bit instructions exist. fadds vrz, vrx,vry

Data type Single-precision floating point

Description: Add single-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrx + vry$

Grammar: fadds vrz, vrx,vry

Data type Single-precision floating point

Description: Add single-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

CSKY

FCMPHSD – Double-precision floating point compare when greater than or equal

Unified instruction

Grammar	Operation	Compiling result
fcmphsd vr _x , vr _y	If vr _x >=vr _y set C; else clear C.	Only 32-bit instructions exist. fcmphsd vr _x , vr _y

Data type Double-precision floating point

Description: Compare vr_x and vr_y. If vr_x>=vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x>=vr_y
set C;
else
clear C.

Grammar: fcmphsd vr_x, vr_y

Data type Double-precision floating point

Description: Compare vr_x and vr_y. If vr_x>=vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

FCMPHSS – Single-precision floating point compare when greater than or equal

Unified instruction

Grammar	Operation	Compiling result
fcmphss vr _x , vr _y	If vr _x >=vr _y set C; else clear C.	Only 32-bit instructions exist. fcmphss vr _x , vr _y

Data type Single-precision floating point

Description: Compare vr_x and vr_y. If vr_x>=vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x>=vr_y
 set C;
else
 clear C.

Grammar: fcmphss vr_x, vr_y

Data type Single-precision floating point

Description: Compare vr_x and vr_y. If vr_x>=vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0 0 0 0 0 0 1 1 0 0 0 0 0 0 0

FCMPLTD – Double-precision floating point compare when

smaller

Unified instruction

Grammar	Operation	Compiling result
fcmpltd vr _x , vr _y	If vr _x <vr _y set C; else clear C.	Only 32-bit instructions exist. fcmpltd vr _x , vr _y

Data type Double-precision floating point

Description: Compare vr_x and vr_y. If vr_x<vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x<vr_y

Operation: set C;
else
clear C.

Grammar: fcmpltd vr_x, vr_y

Data type Double-precision floating point

Description: Compare vr_x and vr_y. If vr_x<vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0 0 0 0 1 0 0 1 1 0 1 0 0 0 0

CSKY

FCMPLTS – Single-precision floating point compare when

smaller

Unified instruction

Grammar	Operation	Compiling result
fcmplts vr _x , vr _y	If vr _x <vr _y set C; else clear C.	Only 32-bit instructions exist. fcmplts vr _x , vr _y

Data type Single-precision floating point

Description: Compare vr_x and vr_y. If vr_x<vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: If vr_x<vr_y
 set C;
else
 clear C.

Grammar: fcmplts vr_x, vr_y

Data type Single-precision floating point

Description: Compare vr_x and vr_y. If vr_x<vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

**Instruction
format:**

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0 0 0 0 0 0 1 1 0 1 0 0 0 0

CSKY

FCMPNED – Double-precision floating point compare when not equal

Unified instruction

Grammar	Operation	Compiling result
fcmpned vr _x , vr _y	If vr _x !=vr _y set C; else clear C.	Only 32-bit instructions exist. fcmpned vr _x , vr _y

Data type Double-precision floating point

Description: Compare vr_x and vr_y. If vr_x is not equal to vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x!=vr_y
 set C;
else
 clear C.

Grammar: fcmpned vr_x, vr_y

Data type Double-precision floating point

Description: Compare vr_x and vr_y. If vr_x is not equal to vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0 0 0 0 1 0 0 1 1 1 0 0 0 0 0

FCMPNES – Single-precision floating point compare when not equal

Unified instruction

Grammar	Operation	Compiling result
fcmpnes vr _x , vr _y	If vr _x !=vr _y set C; else clear C.	Only 32-bit instructions exist. fcmpnes vr _x , vr _y

Data type Single-precision floating point

Description: Compare vr_x and vr_y. If vr_x is not equal to vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: If vr_x!=vr_y
 set C;
else
 clear C.

Grammar: fcmpnes vr_x, vr_y

Data type Single-precision floating point

Description: Compare vr_x and vr_y. If vr_x is not equal to vr_y, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0

FCMPUOD – Judge whether the double operand of double-precision floating point is NaN

Unified instruction

Grammar	Operation	Compiling result
fcmpuod vr _x , vr _y	If vr _x ==NaN vr _y ==NaN set C; else clear C.	Only 32-bit instructions exist. fcmpuod vr _x , vr _y

Data type Double-precision floating point

Description: If vr_x is NaN or vr_y is vry, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x==NaN || vr_y==NaN
 set C;
 else
 clear C.

Grammar:

fcmpuod vr_x, vr_y

Data type

Double-precision floating point

Description:

If vr_x is NaN or vr_y is vry, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0 0 0 0 1 0 0 1 1 1 1 0 0 0 0

CSKY

FCMPUOS – Judge whether the double operand of single-precision floating point is NaN

Unified instruction

Grammar	Operation	Compiling result
fcmpuos vr _x , vr _y	If vr _x ==NaN vr _y ==NaN set C; else clear C.	Only 32-bit instructions exist. fcmpuos vr _x , vr _y

Data type Single-precision floating point

Description: If vr_x is NaN or vr_y is vry, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x==NaN||vr_y==NaN
 set C;
 else
 clear C.

Grammar: fcmpuos vr_x, vr_y

Data type Single-precision floating point

Description: If vr_x is NaN or vr_y is vry, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0 0 0 0 0 0 1 1 1 1 0 0 0 0 0

CSKY

FCMPZHSD – Double-precision floating point compare when greater than or equal to zero

Unified instruction

Grammar	Operation	Compiling result
fcmpzhsd vr _x	If vr _x >=0 set C; else clear C.	Only 32-bit instructions exist. fcmpzhsd vr _x

Data type Double-precision floating point

Description: If vr_x is greater than or equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x>=0
set C;
else
clear C.

Grammar: fcmpzhsd vr_x

Data type Double-precision floating point

Description: If vr_x is greater than or equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

**Instruction
format:**

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	0	0

CSKY

FCMPZHSS – Single-precision floating point compare when greater than or equal to zero

Unified instruction

Grammar	Operation	Compiling result
fcmpzhss vr _x	If vr _x >=0 set C; else clear C.	Only 32-bit instructions exist. fcmpzhss vr _x

Data type Single-precision floating point

Description: If vr_x is greater than or equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x>=0

Operation: set C;
else

clear C.

Grammar: fcmpzhss vr_x

Data type Single-precision floating point

Description: If vr_x is greater than or equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20	16 15	5 4 3	0
----	----------	-------	-------	-------	---

1	1	1	1	0	1	0	0	0	0	0	VRX	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

CSKY 中文

FCMPZLSD – Double-precision floating point compare when smaller than or equal to zero

Unified instruction

Grammar	Operation	Compiling result
fcmpzlsd vr _x	If vr _x <0 set C; else clear C.	Only 32-bit instructions exist. fcmpzlsd vr _x

Data type Double-precision floating point

Description: If vr_x is smaller than or equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x<0

Operation: set C;
else

clear C.

Grammar: fcmpzlsd vr_x

Data type Double-precision floating point

Description: If vr_x is smaller than or equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0											
1	1	1	1	0	1	0	0	0	0	VRX	0	0	0	0	1	0	0	1	0	0	0	0

CSKY

FCMPZLSS – Single-precision floating point compare when smaller than or equal to zero

Unified instruction

Grammar	Operation	Compiling result
fcmpzrss vr _x	If vr _x <0 set C; else clear C.	Only 32-bit instructions exist. fcmpzrss vr _x

Data type Single-precision floating point

Description: If vr_x is smaller than or equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x<0

Operation: set C;
else
clear C.

Grammar: fcmpzrss vr_x

Data type Single-precision floating point

Description: If vr_x is smaller than or equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	0

CSKY

FCMPZNED – Double-precision floating point compare when not equal to zero

Unified instruction

Grammar	Operation	Compiling result
fcmpzned vr _x	If vr _x !=0 set C; else clear C.	Only 32-bit instructions exist. fcmpzned vr _x

Data type Double-precision floating point

Description: If vr_x is not equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x!=0

Operation: set C;
else
 clear C.

Grammar: fcmpzned vr_x

Data type Double-precision floating point

Description: If vr_x is not equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

**Instruction
format:**

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0															
1	1	1	1	0	1	0	0	0	0	VRX	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0

CSKY

FCMPZNES – Single-precision floating point compare when not equal to zero

Unified instruction

Grammar	Operation	Compiling result
fcmpznes vr _x	If vr _x !=0 set C; else clear C.	Only 32-bit instructions exist. fcmpznes vr _x

Data type Single-precision floating point

Description: If vr_x is not equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x!=0

Operation: set C;
else
 clear C.

Grammar: fcmpznes vr_x

Data type Single-precision floating point

Description: If vr_x is not equal to zero, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

**Instruction
format:**

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0												
1	1	1	1	0	1	0	0	0	0	VRX	0	0	0	0	0	0	1	0	1	0	0	0	0

CSKY

FCMPZUOD – Judge whether the single operand of double-precision floating point is NaN

Unified instruction

Grammar	Operation	Compiling result
fcmpzuod vr _x	If vr _x ==NaN set C; else clear C.	Only 32-bit instructions exist. fcmpzuod vr _x

Data type Double-precision floating point

Description: If vr_x is NaN, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x==NaN

Operation: set C;
else
 clear C.

Grammar: fcmpzuod vr_x

Data type Double-precision floating point

Description: If vr_x is NaN, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	0	0

CSKY

FCMPZUOS – Judge whether the single operand of single-precision floating point is NaN

Unified instruction

Grammar	Operation	Compiling result
fcmpzuos vr _x	If vr _x ==NaN set C; else clear C.	Only 32-bit instructions exist. fcmpzuos vr _x

Data type Single-precision floating point

Description: If vr_x is NaN, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

If vr_x==NaN

Operation: set C;
else
 clear C.

Grammar: fcmpzuos vr_x

Data type Single-precision floating point

Description: If vr_x is NaN, set the condition bit C; otherwise, clear the condition bit.

Influence on Yes

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	0	0

CSKY

FDIVD – Double-precision floating point multiply

Unified instruction

Grammar	Operation	Compiling result
fdivd vrz, vrx,vry	$vrz = vrx/vry$	Only 32-bit instructions exist. fdivd vrz, vrx,vry

Data type Double-precision floating point

Description: Divide double-precision floating points in vrx and vry by each other, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrx/vry$

Grammar: fdivd vrz, vrx,vry

Data type Double-precision floating point

Description: Divide double-precision floating points in vrx and vry by each other, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FDIVS – Single-precision floating point multiply

Unified instruction

Grammar	Operation	Compiling result
fdivs vrz, vrx,vry	$vrz = vrx/vry$	Only 32-bit instructions exist. fdivs vrz, vrx,vry

Data type Single-precision floating point

Description: Divide single-precision floating points in vrx and vry by each other, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrx/vry$

Grammar: fdivs vrz, vrx,vry

Data type Single-precision floating point

Description: Divide single-precision floating points in vrx and vry by each other, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FDTOS – Transform double-precision floating point into single-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fdtos vrz, vrx	vrz= (float)vrx	Only 32-bit instructions exist. fdtos vrz, vrx

Data type Double-precision floating point

Description: Transform double-precision floating point in vrx into single-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= (float)vrx

Grammar: fdtos vrz, vrx

Data type Double-precision floating point

Description: Transform double-precision floating point in vrx into single-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0									
1	1	1	1	0	1	0	0	0	0	VRX	0	0	0	1	1	0	1	1	0	0	VRZ

CSKY

FDTOSI – Transform double-precision floating point into signed integer

Unified instruction

Grammar	Operation	Compiling result
fdtosi.rm vrz, vrx where rm is rn/rz/rpi/rni	vrz= (signed long)vrx	Only 32-bit instructions exist. fdtosi.rm vrz, vrx

Data type Double-precision floating point

Transform double-precision floating point in vrx into signed integer, and save the result in vrz. RM refers to the rounding mode.

RM represents:

2'b00: Round to nearest; the corresponding assembly instruction is fdtosi.rn

Description: 2'b01: Round to 0; the corresponding assembly instruction is fdtosi.rz

2'b10: Round to positive infinity; the corresponding assembly instruction is fdtosi.rpi

2'b11: Round to negative infinity; the corresponding assembly instruction is fdtosi.rni

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= (signed long)vrx

Grammar: fdtosi.rm vrz, vrx

Data type Double-precision floating point

Transform double-precision floating point in vrx into signed integer, and save the result in vrz. RM refers to the rounding mode.

Description: RM represents:

2'b00: Round to nearest; the corresponding assembly instruction is fdtosi.rn

2'b01: Round to 0; the corresponding assembly instruction is fdtosi.rz

2'b10: Round to positive infinity; the corresponding assembly instruction is fdtosi.rpi

2'b11: Round to negative infinity; the corresponding assembly instruction is fdtosi.rni

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	VRX	0	0

FDTOUT – Transform double-precision floating point into unsigned integer

Unified instruction

Grammar	Operation	Compiling result
fdtoui.rm vrz, vrx where rm is rn/rz/rpi/rni	vrz= (long)vrx	Only 32-bit instructions exist. fdtoui.rm vrz, vrx

Data type Double-precision floating point

Transform double-precision floating point in vrz into unsigned integer, and save the result in vrz. RM refers to the rounding mode.

RM represents:

2'b00: Round to nearest; the corresponding assembly instruction is fdtosi.rn

Description: 2'b01: Round to 0; the corresponding assembly instruction is fdtosi.rz

2'b10: Round to positive infinity; the corresponding assembly instruction is fdtosi.rpi

2'b11: Round to negative infinity; the corresponding assembly instruction is fdtosi.rni

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= (unsigned long)vrx

Grammar: fdtoui.rm vrz, vrx

Data type Double-precision floating point

Transform double-precision floating point in vrz into unsigned integer, and save the result in vrz. RM refers to the rounding mode.

Description: RM represents:

2'b00: Round to nearest; the corresponding assembly instruction is fdtosi.rn

2'b01: Round to 0; the corresponding assembly instruction is fdtosi.rz

2'b10: Round to positive infinity; the corresponding assembly instruction is fdtosi.rpi

2'b11: Round to negative infinity; the corresponding assembly instruction is fdtosi.rni

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	VRX	0	0

FLDD – Load double-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fldd vrz, (rx, disp)	$VRZ[63:0] \leftarrow MEM[RX + \text{zero_extend}(\text{offset} \ll 2)];$ $VRZ[127:64] \leftarrow 64'b0;$	Only 32-bit instructions exist. fldd vrz, (rx, disp)

Description: Load double-precision floating point from storage to vector register VRZ. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by 2 bits to 32 bits. The address space of FLDD instruction is +1KB.

Attention: The offset DISP is gained after the binary operand {IMM4H,IMM4L} shifts left by two bits.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load double-precision floating point from storage to vector register VRZ

$$VRZ[63:0] \leftarrow MEM[RX + \text{zero_extend}(\text{offset} \ll 2)];$$

$$VRZ[127:64] \leftarrow 64'b0;$$

Grammar: fldd vrz, (rx, disp)

Description: Load double-precision floating point from storage to vector register VRZ. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by 2 bits to 32 bits. The address space of FLDD instruction is +1KB.

Attention: The offset DISP is gained after the binary operand {IMM4H,IMM4L} shifts left by two bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25	24	21 20	16 15	8 7	4 3	0
1 1 1 1 0 1 0 IMM4H RX 0 0 1 0 0 0 0 1 IMM4L VRZ							

FLDM – Load vector floating point

Unified instruction

Grammar	Operation	Compiling result
fldm vrz, (rx, disp)	VRZ[63:0] ← MEM[RX + zero_extend(offset << 3)]; VRZ[127:64] ← 64'b0;	Only 32-bit instructions exist. fldm vrz, (rx, disp)

Description: Load two single-precision floating points from storage to vector register VRZ. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by 3 bits to 32 bits. The address space of FLDM instruction is +1KB.

Attention: The offset DISP is gained after the binary operand {IMM4H,IMM4L} shifts left by three bits.

**Influence on flag
bit:** No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load vector floating point from storage to vector register
 $VRZ[63:0] \leftarrow MEM[RX + zero_extend(offset << 3)];$
 $VRZ[127:64] \leftarrow 64'b0;$

Grammar: fldm vrz, (rx, disp)

Description: Load two single-precision floating points from storage to vector register VRZ. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by 3 bits to 32 bits. The address space of FLDM instruction is +1KB.

Attention: The offset DISP is gained after the binary operand {IMM4H,IMM4L} shifts left by three bits.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction format:

31 30	26 25 24	21 20	16 15	8 7	4 3	0
1 1 1 1 0 1 0 IMM4H RX 0 0 1 0 0 0 1 0 IMM4L VRZ						

FLDMD – Load consecutive double-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fldmd vry-vrz, (rx)	<p>Load multiple consecutive double-precision floating points from storage to a group of consecutive vector register files</p> <pre>dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[63:0] ← MEM[addr]; VRdst[127:64] ← 64'b0; dst ← dst + 1; addr ← addr + 8; }</pre>	<p>Only 32-bit instructions exist.</p> <p>fldmd vry-vrz, (rx);</p>

Description: Load multiple consecutive double-precision floating points from storage to a group of consecutive vector register files starting from vector register VRY successively. In another word, load the first double word of the address appointed by storage to vector register VRY; load the second double word to register VR(Y+1), and the like; load the last double word to register VRZ. The effective address of storage is decided by the contents of base register RX.

Influence on flag No influence

bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load multiple consecutive double-precision floating points from storage to a group of consecutive vector register files

```
dst ← Y; addr ← RX;
for (n = 0; n <= (Z-Y); n++){
```

```

VRdst[63:0] ← MEM[addr];
VRdst[127:64] ← 64'b0;
dst ← dst + 1;
addr ← addr + 8;
}

```

Grammar: fldmd vry-vrz, (rx)

Description: Load multiple consecutive double-precision floating points from storage to a group of consecutive vector register files starting from vector register VRY successively. In another word, load the first **word** of the address appointed by storage to vector register VRY; load the second **word** to register VR(Y+1), and the like; load the last **word** to register VRZ. The effective address of storage is decided by the contents of base register RX.

Influence on No influence

flag bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31	30	26	25	24	21	20	16	15	8	7	4	3	0								
1	1	1	1	0	1	0	IMM4	RX	0	0	1	1	0	0	0	1	0	0	0	0	VRY

IMM4 field – Assign the number of destination registers, IMMM4 = Z – Y.

0000 – 1 destination register

0001 – 2 destination registers

.....

1111 – 16 destination registers

FLDMM – Load consecutive vector floating point

**Unified
instruction**

Grammar	Operation	Compiling result
fldmd vry-vrz, (rx)	<p>Load multiple consecutive vector floating points from storage to a group of consecutive vector register files</p> <pre>dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++) { VRdst[63:0] ← MEM[addr]; VRdst[127:64] ← 64'b0; dst ← dst + 1; addr ← addr + 8; }</pre>	<p>Only 32-bit instructions exist.</p> <p>fldmm vry-vrz, (rx);</p>

Description: Load multiple consecutive vector floating points (each vector floating point includes two single-precision floating points) from storage to a group of consecutive vector register files starting from vector register VRY successively. In another word, load the first double word of the address appointed by storage to vector register VRY; load the second double word to register VR(Y+1), and the like; load the last double word to register VRZ. The effective address of storage is decided by the contents of base register RX.

Influence on flag No influence

bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

**32-bit
instruction**

Operation:

Load multiple consecutive vector floating points from storage to a group of consecutive vector register files

```
dst ← Y; addr ← RX;
```

```

for (n = 0; n <= (Z-Y); n++){
    VRdst[63:0] ← MEM[addr];
    VRdst[127:64] ← 64'b0;
    dst ← dst + 1;
    addr ← addr + 8;
}

```

Grammar: fldmd vry-vrz, (rx)

Description: Load multiple consecutive vector floating points (each vector floating point includes two single-precision floating points) from storage to a group of consecutive vector register files starting from vector register VRY successively. In another word, load the first **word** of the address appointed by storage to vector register VRY; load the second **word** to register VR(Y+1), and the like; load the last **word** to register VRZ. The effective address of storage is decided by the contents of base register RX.

Influence on No influence

flag bit:

Restriction: VRZ SHOULD BE GREATER THAN OR EQUAL TO VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31	30	26	25	24	21	20	16	15	8	7	4	3	0								
1	1	1	1	0	1	0	IMM4	RX	0	0	1	1	0	0	1	0	0	0	0	0	VRY

IMM4 field – Assign the number of destination registers, $IMM4 = Z - Y$.

0000 – 1 destination register

0001 – 2 destination registers

.....

1111 – 16 destination registers

FLDMS – Load consecutive single-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fldms vry-vrz, (rx)	<p>Load multiple consecutive single-precision floating points from storage to a group of consecutive vector register files</p> <pre>dst ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ VRdst[31:0] ← MEM[addr]; VRdst[127:32] ← 96'b0; dst ← dst + 1; addr ← addr + 4; }</pre>	<p>Only 32-bit instructions exist.</p> <p>fldms vry-vrz, (rx);</p>

Description: Load multiple consecutive single-precision floating points from storage to a group of consecutive vector register files starting from vector register VRY successively. In another word, load the first word of the address appointed by storage to vector register VRY; load the second word to register VR(Y+1), and the like; load the last word to register VRZ. The effective address of storage is decided by the contents of base register RX.

Influence on flag bit: No influence

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load multiple consecutive single-precision floating points from storage to a group of consecutive vector register files

```
dst ← Y; addr ← RX;
for (n = 0; n <= (Z-Y); n++){
```

```

VRdst ← MEM[addr];
dst ← dst + 1;
addr ← addr + 4;
}

```

Grammar: fldms vry-vrz, (rx)

Description: Load multiple consecutive single-precision floating points from storage to a group of consecutive vector register files starting from vector register VRY successively. In another word, load the first word of the address appointed by storage to vector register VRY; load the second word to register VR(Y+1), and the like; load the last word to register VRZ. The effective address of storage is decided by the contents of base register RX.

Influence on No influence

flag bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31	30	26	25	24	21	20	16	15	8	7	4	3	0							
1	1	1	1	0	1	0	IMM4	RX	0	0	1	1	0	0	0	0	0	0	0	VRY

IMM4 field – Assign the number of destination registers, $IMM4 = Z - Y$.

0000 – 1 destination register

0001 – 2 destination registers

.....

1111 – 16 destination registers

FLDRD – Load double-precision floating point in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
fldrdrd vrz, (rx, ry << 0)	Load double-precision floating point from storage to register VRZ[63:0] ← MEM[RX + RY << IMM2];	Only 32-bit instructions exist.
fldrdrd vrz, (rx, ry << 1)		fldrdrd vrz, (rx, ry << 1)
fldrdrd vrz, (rx, ry << 2)		fldrdrd vrz, (rx, ry << 2)
fldrdrd vrz, (rx, ry << 3)		fldrdrd vrz, (rx, ry << 3)

Description: Load double-precision floating point from storage to register VRZ. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit

instruction

Operation: Load double-precision floating point from storage to register VRZ[63:0] ← MEM[RX + RY << IMM2];
VRZ[127:64] = 64'b0;

Grammar: fldrdrd vrz, (rx, ry << 0)
fldrdrd vrz, (rx, ry << 1)
fldrdrd vrz, (rx, ry << 2)
fldrdrd vrz, (rx, ry << 3)

Description: Load double-precision floating point from storage to register VRZ. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2.

The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

fldrdr vrz, (rx, ry << 0)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	0	0

fldrdr vrz, (rx, ry << 1)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	0	0

fldrdr vrz, (rx, ry << 2)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	0	0

fldrdr vrz, (rx, ry << 3)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	0	0

FLDRM – Load vector floating point in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
fldr m vrz, (rx, ry << 0)	Load vector floating point from storage to register VRZ[63:0] ← MEM[RX + RY << IMM2];	Only 32-bit instructions exist.
fldr m vrz, (rx, ry << 1)	VRZ[127:64] = 64'b0;	fldr vrz, (rx, ry << 1)
fldr m vrz, (rx, ry << 2)		fldr vrz, (rx, ry << 2)
fldr m vrz, (rx, ry << 3)		fldr vrz, (rx, ry << 3)

Description: Load two single-precision floating points from storage to register VRZ. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load double-precision floating point from storage to register

VRZ[63:0] ← MEM[RX + RY << IMM2];
 VRZ[127:64] = 64'b0;

Grammar:
 fldr **m** vrz, (rx, ry << 0)
 fldr **m** vrz, (rx, ry << 1)
 fldr **m** vrz, (rx, ry << 2)
 fldr **m** vrz, (rx, ry << 3)

Description: Load two single-precision floating points from storage to register VRZ. Adopt the addressing mode of register and register offset. The effective

address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

fldr m vrz, (rx, ry << 0)

31 30	26 25	21 20	16 15	8 7	6	5	4	3	0
1 1 1 1 0 1	RY	RX	0 0 1 0 1 0 1 0	0 0 0 0 0					VRZ

fldr m vrz, (rx, ry << 1)

31 30	26 25	21 20	16 15	8 7	6	5	4	3	0
1 1 1 1 0 1	RY	RX	0 0 1 0 1 0 1 0	0 0 1 0					VRZ

fldr m vrz, (rx, ry << 2)

31 30	26 25	21 20	16 15	8 7	6	5	4	3	0
1 1 1 1 0 1	RY	RX	0 0 1 0 1 0 1 0	0 1 0 0					VRZ

fldr m vrz, (rx, ry << 3)

31 30	26 25	21 20	16 15	8 7	6	5	4	3	0
1 1 1 1 0 1	RY	RX	0 0 1 0 1 0 1 0	0 1 1 0					VRZ

FLDRS – Load single-precision floating point in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
fldrsvrz, (rx, ry << 0)	Load single-precision floating point from storage to register VRZ[31:0] ← MEM[RX + RY << IMM2];	Only 32-bit instructions exist.
fldrsvrz, (rx, ry << 1)		fldrsvrz, (rx, ry << 1)
fldrsvrz, (rx, ry << 2)		fldrsvrz, (rx, ry << 2)
fldrsvrz, (rx, ry << 3)		fldrsvrz, (rx, ry << 3)

Description: Load single-precision floating point from storage to register VRZ. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load single-precision floating point from storage to register VRZ[31:0] ← MEM[RX + RY << IMM2];
VRZ[127:32] = 96'b0;

Grammar:
fldrsvrz, (rx, ry << 0)
fldrsvrz, (rx, ry << 1)
fldrsvrz, (rx, ry << 2)
fldrsvrz, (rx, ry << 3)

Description: Load double-precision floating point from storage to register VRZ. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2.

The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

fldrsvrz, (rx, ry << 0)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	0	0

fldrsvrz, (rx, ry << 1)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	0	1

fldrsvrz, (rx, ry << 2)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	0	0

fldrsvrz, (rx, ry << 3)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

FLDS – Load single-precision floating point

Unified instruction

Grammar	Operation	Compiling result
flds vrz, (rx, disp)	$VRZ[31:0] \leftarrow MEM[RX + zero_extend(offset \ll 2)];$ $VRZ[127:32] \leftarrow 96'b0;$	Only 32-bit instructions exist. flds vrz, (rx, disp)

Description: Load single-precision floating point from storage to register VRZ.
 Adopt the addressing mode of register and immediate operand offset.
 The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by 2 bits to 32 bits. The address space of FLDS instruction is +1KB.
 Attention: The offset DISP is gained after the binary operand {IMM4H,IMM4L} shifts left by two bits.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load single-precision floating point from storage to register
 $VRZ[31:0] \leftarrow MEM[RX + zero_extend(offset \ll 2)];$
 $VRZ[127:32] \leftarrow 96'b0;$

Grammar: flds vrz, (rx, disp)

Description: Load single-precision floating point from storage to register VRZ.
 Adopt the addressing mode of register and immediate operand offset.
 The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by 2 bits to 32 bits. The address space of FLDS instruction is +1KB.
 Attention: The offset DISP is gained after the binary operand

{IMM4H,IMM4L} shifts left by two bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25 24	21 20	16 15	8 7	4 3	0
1 1 1 1 0 1 0 IMM4H RX 0 0 1 0 0 0 0 0 IMM4L VRZ						

FLRWD – Double-precision floating point storage read-in

Unified instruction

Grammar	Operation	Compiling result
flrwd vrz, ±m.n	$\text{VRZ}[63:0] \leftarrow \text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffff];$ $\text{VRZ}[127:64] \leftarrow 64'b0;$	Only 32-bit instructions exist. flrwd vrz, ±m.n

Description: Load the **64-bit single-precision floating point** ($\pm m.n$) from storage to floating point register VRZ. The effective address of storage is gained by adding PC to the value of unsigned extending the relative offset shifting left by 2 bits to 32 bits. The address space of FLRWD instruction is 4GB.

Attention: Due to insufficient precision or floating point immediate operand that exceeds the expression range, the compiler might give errors.

Influence on flag bit: No influence

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load **single-precision floating point** from storage to floating point register

$$\text{VRZ}[63:0] \leftarrow \text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffff];$$

$$\text{VRZ}[127:32] \leftarrow 64'b0;$$

Grammar: flrwd vrz, imm32

Description: Load the 32-bit **single-precision floating point** ($\pm m.n$) from storage to floating point register VRZ. The effective address of storage is gained by adding PC to the value of unsigned extending the relative offset shifting left by 2 bits to 32 bits. The address space of FLRWD instruction is 4GB.

Attention: Due to insufficient precision or floating point immediate

operand that exceeds the expression range, the compiler might give errors.

Influence on flag bit:

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	13	12	11	8	7	4	3	0
1 1 1 1 0 1 0 IMM4H RX 0 0 1 1 1 0 0 1 IMM4L VRZ																

FLRWS – Single-precision floating point storage read-in

Unified instruction

Grammar	Operation	Compiling result
flrws vrz, ±m.n	$\text{VRZ}[31:0] \leftarrow \text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffff];$ $\text{VRZ}[127:32] \leftarrow 96'b0;$	Only 32-bit instructions exist. flrws vrz, ±m.n

Description: Load the 32-bit single-precision floating point ($\pm m.n$) from storage to floating point register VRZ. The effective address of storage is gained by adding PC to the value of unsigned extending the relative offset shifting left by 2 bits to 32 bits. The address space of FLRWS instruction is 4GB.

Attention: Due to insufficient precision or floating point immediate operand that exceeds the expression range, the compiler might give errors.

Influence on flag bit: No influence

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Load single-precision floating point from storage to floating point register

$$\text{VRZ}[31:0] \leftarrow \text{MEM}[(\text{PC} + \text{zero_extend}(\text{offset} \ll 2)) \& 0xffffffff];$$

$$\text{VRZ}[127:32] \leftarrow 96'b0;$$

Grammar: flrws vrz, imm32

Description: Load the 32-bit single-precision floating point ($\pm m.n$) from storage to floating point register VRZ. The effective address of storage is gained by adding PC to the value of unsigned extending the relative offset shifting left by 2 bits to 32 bits. The address space of FLRWS instruction is 4GB.

Attention: Due to insufficient precision or floating point immediate

operand that exceeds the expression range, the compiler might give errors.

Influence on flag bit:

Exception: Access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	13	12	11	8	7	4	3	0
1 1 1 1 0 1 0 IMM4H RX 0 0 1 1 1 0 0 0 IMM4L VRZ																

FMACD – Double-precision floating point multiply-accumulate

Unified instruction

Grammar	Operation	Compiling result
fmacd vrz, vrx, vry	$vrz += vrz * vry$	Only 32-bit instructions exist. fmacd vrz, vrx, vry

Data type Double-precision floating point

Description: Add the product of multiplying double-precision floating point in vrz and double-precision floating point in vry to the value in vrz, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz += vrz * vry$

Grammar: fmacd vrz, vrx, vry

Data type Double-precision floating point

Description: Add the product of multiplying double-precision floating point in vrz and double-precision floating point in vry to the value in vrz, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

CSKY

FMACM – SIMD single-precision floating point

multiply-accumulate

Unified instruction

Grammar	Operation	Compiling result
fmacm vrz, vrx, vry	$vrz[31:0] += vrx[31:0]*vry[31:0]$, $vrz[63:32] += vrx[63:32]*vry[63:32]$	Only 32-bit instructions exist. fmacm vrz, vrx, vry

Data type Single-precision floating point

Add the product of multiplying single-precision floating point in vrx[31:0] and single-precision floating point in vry[31:0] to the value in vrz[31:0], and save the result in vrz[31:0].

Description: Add the product of multiplying single-precision floating point in vrx[63:32] and single-precision floating point in vry[63:32] to the value in vrz[63:32], and save the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz[31:0] += vrx[31:0]*vry[31:0]$, $vrz[63:32] += vrx[63:32]*vry[63:32]$

Grammar: fmacm vrz, vrx, vry

Data type Single-precision floating point

Add the product of multiplying single-precision floating point in vrx[31:0] and single-precision floating point in vry[31:0] to the value in vrz[31:0], and save the result in vrz[31:0].

Description: Add the product of multiplying single-precision floating point in vrx[63:32] and single-precision floating point in vry[63:32] to the value in vrz[63:32], and save the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20 19	16 15	5 4 3	0
1 1 1 1 0 1 0 VRY 0 VRX 0 0 0 1 0 0 1 0 1 0 0 0 VRZ					

FMACS – Single-precision floating point multiply-accumulate

Unified instruction

Grammar	Operation	Compiling result
fmacs vrz, vrx, vry	$vrz += vrz * vry$	Only 32-bit instructions exist. fmacs vrz, vrx, vry

Data type Single-precision floating point

Description: Add the product of multiplying single-precision floating point in vrz and single-precision floating point in vry to the value in vrz, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz += vrz * vry$

Grammar: fmacs vrz, vrx, vry

Data type Single-precision floating point

Description: Add the product of multiplying single-precision floating point in vrz and single-precision floating point in vry to the value in vrz, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0											
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	0	0	0	1	0	1	0	0	0	0	VRZ

CSKY

FMFVRH – Read transfer high word from floating point register

register

Unified instruction

Grammar	Operation	Compiling result
fmfvrh rz, vrx	rz= vrx[63:32]	Only 32-bit instructions exist. fmfvrh rz, vrx

Data type Single-precision floating point

Description: Transfer the high word of floating point register to rz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: rz= vrx[63:32]

Grammar: fmfvrh rz, vrx

Data type Single-precision floating point

Description: Transfer the high word of floating point register to rz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	0	0	VRZ

FMFVRL – Read transfer low word from floating point register

Unified instruction

Grammar	Operation	Compiling result
fmfvrl rz, vrz	$rz = vrz[31:0]$	Only 32-bit instructions exist. fmfvrl rz, vrz

Data type Single-precision floating point

Description: Transfer the low word of floating point register to rz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $rz = vrz[31:0]$

Grammar: fmfvrl rz, vrz

Data type Single-precision floating point

Description: Transfer the low word of floating point register to rz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	0	VRZ

FMOVD – Double-precision floating point move

Unified instruction

Grammar	Operation	Compiling result
fmovd vrz, vrx	vrz= vrx	Only 32-bit instructions exist. fmovd vrz, vrx

Data type Double-precision floating point

Description: Transfer double-precision floating point in vrx to vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= vrx

Grammar: fmovd vrz, vrx

Data type Double-precision floating point

Description: Transfer double-precision floating point in vrx to vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	0	0

VRX VRZ

FMOVM – SIMD single-precision floating point move

Unified instruction

Grammar	Operation	Compiling result
fmovm vrz, vrx	vrz= vrx	Only 32-bit instructions exist. fmovm vrz, vrx

Data type Single-precision floating point

Description: Transfer single-precision floating point in vrx to vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= vrx

Grammar: fmovm vrz, vrx

Data type Single-precision floating point

Description: Transfer single-precision floating point in vrx to vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	0	0

VRX VRZ

FMOVS – Single-precision floating point move

Unified instruction

Grammar	Operation	Compiling result
fmovs vrz, vrx	vrz= vrx	Only 32-bit instructions exist. fmovs vrz, vrx

Data type Single-precision floating point

Description: Transfer single-precision floating point in vrx to vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= vrx

Grammar: fmovs vrz, vrx

Data type Single-precision floating point

Description: Transfer single-precision floating point in vrx to vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	0	1	0

VRX VRZ

FMSCD – Double-precision floating point multiply-subtract

Unified instruction

Grammar	Operation	Compiling result
fmscd vrz, vrx, vry	$vrz = -vrz + vrx * vry$	Only 32-bit instructions exist. fmscd vrz, vrx, vry

Data type Double-precision floating point

Description: Subtract the value in vrz from the product of multiplying double-precision floating point in vrx and double-precision floating point in vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = -vrz + vrx * vry$

Grammar: fmscd vrz, vrx, vry

Data type Double-precision floating point

Description: Subtract the value in vrz from the product of multiplying double-precision floating point in vrx and double-precision floating point in vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20 19	16 15	5 4 3	0
----	----------	----------	-------	-------	---

1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	0	1	0	1	0	1	0	VRZ
---	---	---	---	---	---	---	-----	---	-----	---	---	---	---	---	---	---	---	---	---	-----



FMSCM – SIMD single-precision floating point multiply-subtract

Unified instruction

Grammar	Operation	Compiling result
fmsecm vrz, vrx, vry	$vrz[31:0] = -vrz[31:0] + vrx[31:0]*vry[31:0]$, $vrz[63:32] = -vrz[63:32] + vrx[63:32]*vry[63:32]$	Only 32-bit instructions exist. fmsecm vrz, vrx, vry

Data type Single-precision floating point

Subtract the value in $vrz[31:0]$ from the product of multiplying single-precision floating point in $vrx[31:0]$ and single-precision floating point in $vry[31:0]$, and save the result in $vrz[31:0]$.

Description: Subtract the value in $vrz[63:32]$ from the product of multiplying single-precision floating point in $vrx[63:32]$ and single-precision floating point in $vry[63:32]$, and save the result in $vrz[63:32]$.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit instruction

Operation: $vrz[31:0] = -vrz[31:0] + vrx[31:0]*vry[31:0]$,
 $vrz[63:32] = -vrz[63:32] + vrx[63:32]*vry[63:32]$

Grammar: fmsecm vrz, vrx, vry

Data type Single-precision floating point

Subtract the value in $vrz[31:0]$ from the product of multiplying single-precision floating point in $vrx[31:0]$ and single-precision floating point in $vry[31:0]$, and save the result in $vrz[31:0]$.

Description: Subtract the value in $vrz[63:32]$ from the product of multiplying single-precision floating point in $vrx[63:32]$ and single-precision floating point in $vry[63:32]$, and save the result in $vrz[63:32]$.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FMSCS – Single-precision floating point multiply-subtract

Unified instruction

Grammar	Operation	Compiling result
fmscs vrz, vrx, vry	$vrz = -vrz + vrx * vry$	Only 32-bit instructions exist. fmscs vrz, vrx, vry

Data type Single-precision floating point

Description: Subtract the value in vrz from the product of multiplying single-precision floating point in vrx and single-precision floating point in vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = -vrz + vrx * vry$

Grammar: fmscs vrz, vrx, vry

Data type Single-precision floating point

Description: Subtract the value in vrz from the product of multiplying single-precision floating point in vrx and single-precision floating point in vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0										
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	0	0	0	1	0	1	0	1	0	VRZ

CSKY

FMTVRH – Write transfer to high word of floating point register

register

Unified instruction

Grammar	Operation	Compiling result
fmtvrh vrz, rx	vrz[63:32] = rx	Only 32-bit instructions exist. fmtvrh vrz, rx

Data type Single-precision floating point

Description: Transfer rx to the high word of floating point register.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz[63:32] = rx

Grammar: fmtvrh vrz, rx

Data type Single-precision floating point

Description: Transfer rx to the high word of floating point register.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	1	0

FMTVRL – Write transfer to low word of floating point register

register

Unified instruction

Grammar	Operation	Compiling result
fmtvrl vrz, rx	vrz[31:0] = rx	Only 32-bit instructions exist. fmtvrh vrz, rx

Data type Single-precision floating point

Description: Transfer rx to the low word of floating point register.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz[31:0] = rx

Grammar: Fmtvrl vrz, rx

Data type Single-precision floating point

Description: Transfer rx to the low word of floating point register.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	VRX	0	0

FMULD – Double-precision floating point multiply

Unified instruction

Grammar	Operation	Compiling result
fmuld vrz, vrx,vry	$vrz = vrx * vry$	Only 32-bit instructions exist. fmuld vrz, vrx,vry

Data type Double-precision floating point

Description: Multiply single-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrx * vry$

Grammar: fmuld vrz, vrx,vry

Data type Double-precision floating point

Description: Multiply double-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FMULM – SIMD single-precision floating point multiply

Unified instruction

Grammar	Operation	Compiling result
fmulm vrz, vrx,vry	$vrz[31:0] = vrx[31:0]*vry[31:0];$ $vrz[63:32] = vrx[63:32]*vry[63:32]$	Only 32-bit instructions exist. fmulm vrz, vrx,vry

Data type Single-precision floating point

Description: Multiply single-precision floating points in $vrx[31:0]$ and $vry[31:0]$, and save the result in $vrz[31:0]$; multiply single-precision floating points in $vrx[63:32]$ and $vry[63:32]$, and save the result in $vrz[63:32]$.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz[31:0] = vrx[31:0]*vry[31:0]; vrz[63:32] = vrx[63:32]*vry[63:32]$

Grammar: fmulm vrz, vrx,vry

Data type Single-precision floating point

Description: Multiply single-precision floating points in $vrx[31:0]$ and $vry[31:0]$, and save the result in $vrz[31:0]$; multiply single-precision floating points in $vrx[63:32]$ and $vry[63:32]$, and save the result in $vrz[63:32]$.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

CSKY

FMULS – Single-precision floating point multiply

Unified instruction

Grammar	Operation	Compiling result
fmuls vrz, vrx,vry	$vrz = vrx * vry$	Only 32-bit instructions exist. fmuls vrz, vrx,vry

Data type Single-precision floating point

Description: Multiply single-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrx * vry$

Grammar: fmuls vrz, vrx,vry

Data type Single-precision floating point

Description: Multiply single-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FNEGD – Double-precision floating point negate

Unified instruction

Grammar	Operation	Compiling result
fnegd vrz, vrx	$vrz = -vrx$	Only 32-bit instructions exist. fnegd vrz, vrx

Data type Double-precision floating point

Description: Negate double-precision floating point in vrx and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = -vrx$

Grammar: fnegd vrz, vrx

Data type Double-precision floating point

Description: Negate double-precision floating point in vrx and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	1	1

FNEGM – SIMD single-precision floating point negate

Unified instruction

Grammar	Operation	Compiling result
fnegm vrz, vrx	vrz[31:0]= -vrx[31:0]; vrz[63:32]= -vrx[63:32]	Only 32-bit instructions exist. fnegm vrz, vrx

Data type Single-precision floating point

Description: Negate single-precision floating point in vrz[31:0] and save the result in vrz[31:0]; negate single-precision floating point in vrz[63:32] and save the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz[31:0]= -vrz[31:0]; vrz[63:32]= -vrz[63:32]

Grammar: fnegm vrz, vrx

Data type Single-precision floating point

Description: Negate single-precision floating point in vrz[31:0] and save the result in vrz[31:0]; negate single-precision floating point in vrz[63:32] and save the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0										
1	1	1	1	0	1	0	0	0	0	VRX	0	0	0	1	0	0	0	1	1	1	0	VRZ

CSKY

FNEGS – Single-precision floating point negate

Unified instruction

Grammar	Operation	Compiling result
fnegs vrz, vrx	$vrz = -vrx$	Only 32-bit instructions exist. fnegs vrz, vrx

Data type Single-precision floating point

Description: Negate single-precision floating point in vrx and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = -vrx$

Grammar: fnegs vrz, vrx

Data type Single-precision floating point

Description: Negate single-precision floating point in vrx and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	1	1

VRX

VRZ

FNMACD – Double-precision floating point multiply-negate-accumulate

Unified instruction

Grammar	Operation	Compiling result
fnmacd vrz, vrx, vry	$vrz = -vrz * vry$	Only 32-bit instructions exist. fnmacd vrz, vrx, vry

Data type Double-precision floating point

Description: Negate the product of multiplying double-precision floating point in vrz and double-precision floating point in vry, add the result to the value in vrz, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = -vrz * vry$

Grammar: fnmacd vrz, vrx, vry

Data type Double-precision floating point

Description: Negate the product of multiplying double-precision floating point in vrz and double-precision floating point in vry, add the result to the value in vrz, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0									
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	1	0	1	0	1	1	0	0	VRZ

FNMACM – SIMD single-precision floating point multiply-negate-accumulate

Unified instruction

Grammar	Operation	Compiling result
fnmacm vrz, vrx, vry	$vrz[31:0] = vrz[31:0] - vrz[31:0] * vry[31:0]$, $vrz[63:32] = vrz[63:32] - vrz[63:32] * vry[63:32]$	Only 32-bit instructions exist. fnmacm vrz, vrx, vry

Data type Single-precision floating point

Negate the product of multiplying single-precision floating point in vrz[31:0] and single-precision floating point in vry[31:0], add the result to the value in vrz[31:0], and save the final result in vrz[31:0].

Description: Negate the product of multiplying single-precision floating point in vrz[63:32] and single-precision floating point in vry[63:32], add the result to the value in vrz[63:32], and save the final result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz[31:0] = vrz[31:0] - vrz[31:0] * vry[31:0]$, $vrz[63:32] = vrz[63:32] - vrz[63:32] * vry[63:32]$

Grammar: fnmacm vrz, vrx, vry

Data type Single-precision floating point

Negate the product of multiplying single-precision floating point in vrz[31:0] and single-precision floating point in vry[31:0], add the result to the value in vrz[31:0], and save the final result in vrz[31:0].

Description: Negate the product of multiplying single-precision floating point in vrz[63:32] and single-precision floating point in vry[63:32], add the result to the value in vrz[63:32], and save the final result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

**Instruction
format:**

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FNMACS – Single-precision floating point

multiply-negate-accumulate

Unified instruction

Grammar	Operation	Compiling result
fnmacs vrz, vrx, vry	$vrz = vrz - vrz * vrz$	Only 32-bit instructions exist. fnmacs vrz, vrx, vry

Data type Single-precision floating point

Description: Negate the product of multiplying single-precision floating point in vrz and single-precision floating point in vry, add the result to the value in vrz, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrz - vrz * vrz$

Grammar: fnmacs vrz, vrx, vry

Data type Single-precision floating point

Description: Negate the product of multiplying single-precision floating point in vrz and single-precision floating point in vry, add the result to the value in vrz, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0									
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	0	0	1	0	1	1	0	0	VRZ

FNMSCD – Double-precision floating point multiply-negate-subtract

Unified instruction

Grammar	Operation	Compiling result
fnmscd vrz, vrx, vry	$vrz = - vrz - vrx * vry$	Only 32-bit instructions exist. fnmscd vrz, vrx, vry

Data type Double-precision floating point

Description: Negate the product of multiplying double-precision floating point in vrx and double-precision floating point in vry, subtract the value in vrz from the result, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = - vrz - vrx * vry$

Grammar: fnmscd vrz, vrx, vry

Data type Double-precision floating point

Description: Negate the product of multiplying double-precision floating point in vrx and double-precision floating point in vry, subtract the value in vrz from the result, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0									
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	1	0	1	0	1	1	1	0	VRZ

FNMSCM – SIMD single-precision floating point

multiply-negate-subtract

Unified instruction

Grammar	Operation	Compiling result
fnmscm vrz, vrx, vry	$vrz[31:0] = -vrz[31:0] - vrz[31:0] * vrz[31:0]$ $vrz[63:32] = -vrz[63:32] - vrz[63:32] * vrz[63:32]$	Only 32-bit instructions exist. fnmscm vrz, vrx, vry

Data type Single-precision floating point

Negate the product of multiplying single-precision floating point in vrz[31:0] and single-precision floating point in vry[31:0], subtract the value in vrz[31:0] from the result, and save the final result in vrz[31:0].

Description:

Negate the product of multiplying single-precision floating point in vrz[63:32] and single-precision floating point in vry[63:32], subtract the value in vrz[63:32] from the result, and save the final result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz[31:0] = -vrz[31:0] - vrz[31:0] * vrz[31:0]$,
 $vrz[63:32] = -vrz[63:32] - vrz[63:32] * vrz[63:32]$

Grammar: fnmscm vrz, vrx, vry

Data type Single-precision floating point

Negate the product of multiplying single-precision floating point in vrz[31:0] and single-precision floating point in vry[31:0], subtract the value in vrz[31:0] from the result, and save the final result in vrz[31:0].

Description:

Negate the product of multiplying single-precision floating point in vrz[63:32] and single-precision floating point in vry[63:32], subtract the value in vrz[63:32] from the result, and save the final result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

**Instruction
format:**

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0										
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	1	0	0	1	0	1	1	1	0	VRZ

FNMSCS – Single-precision floating point

multiply-negate-subtract

Unified instruction

Grammar	Operation	Compiling result
fnmcs v _{rz} , v _{rx} , v _{ry}	v _{rz} =- v _{rz} -v _{rx} *v _{ry}	Only 32-bit instructions exist. fnmcs v _{rz} , v _{rx} , v _{ry}

Data type Single-precision floating point

Description: Negate the product of multiplying single-precision floating point in v_{rx} and single-precision floating point in v_{ry}, subtract the value in v_{rz} from the result, and save the final result in v_{rz}.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: v_{rz}=- v_{rz}-v_{rx}*v_{ry}

Grammar: fnmcs v_{rz}, v_{rx}, v_{ry}

Data type Single-precision floating point

Description: Negate the product of multiplying single-precision floating point in v_{rx} and single-precision floating point in v_{ry}, subtract the value in v_{rz} from the result, and save the final result in v_{rz}.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0									
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	0	0	1	0	1	1	1	0	VRZ

FNMULD – Double-precision floating point multiply-negate

Unified instruction

Grammar	Operation	Compiling result
fnmuld vrz, vrx,vry	$vrz = -(vrx * vry)$	Only 32-bit instructions exist. fnmuld vrz, vrx,vry

Data type Double-precision floating point

Description: Multiply double-precision floating points in vrx and vry, negate the result, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = -(vrx * vry)$

Grammar: fnmuld vrz, vrx,vry

Data type Double-precision floating point

Description: Multiply double-precision floating points in vrx and vry, negate the result, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FNMULM – SIMD single-precision floating point multiply-negate

Unified instruction

Grammar	Operation	Compiling result
fnmulm vrz, vrx,vry	$vrz[31:0] = -(vrx[31:0]*vry[31:0]);$ $vrz[63:32] = -(vrx[63:32]*vry[63:32])$	Only 32-bit instructions exist. fnmulm vrz, vrx,vry

Data type Single-precision floating point

Description: Multiply single-precision floating points in vrx[31:0] and vry[31:0], negate the result, and save the final result in vrz[31:0]; multiply single-precision floating points in vrx[63:32] and vry[63:32], negate the result, and save the final result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz[31:0] = -(vrx[31:0]*vry[31:0]);$ $vrz[63:32] = -(vrx[63:32]*vry[63:32])$

Grammar: fnmulm vrz, vrx,vry

Data type Single-precision floating point

Description: Multiply single-precision floating points in vrx[31:0] and vry[31:0], negate the result, and save the final result in vrz[31:0]; multiply single-precision floating points in vrx[63:32] and vry[63:32], negate the result, and save the final result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0										
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	1	0	0	1	0	0	0	1	0	VRZ

FNMULS – Single-precision floating point multiply-negate

Unified instruction

Grammar	Operation	Compiling result
fnmuls vrz, vrx,vry	$vrz = -(vrx * vry)$	Only 32-bit instructions exist. fnmuls vrz, vrx,vry

Data type Single-precision floating point

Description: Multiply single-precision floating points in vrx and vry, negate the result, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = -(vrx * vry)$

Grammar: fnmuls vrz, vrx,vry

Data type Single-precision floating point

Description: Multiply single-precision floating points in vrx and vry, negate the result, and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FRECIPD – Double-precision floating point reciprocal

Unified instruction

Grammar	Operation	Compiling result
frecipd vrz, vrx	$vrz = 1/vrx$	Only 32-bit instructions exist. frecipd vrz, vrx

Data type Double-precision floating point

Description: Take the reciprocal of double-precision floating point in vrx and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = 1/vrx$

Grammar: frecipd vrz, vrx

Data type Double-precision floating point

Description: Take the reciprocal of double-precision floating point in vrx and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	0	VRZ

FRECIPS – Single-precision floating point reciprocal

Unified instruction

Grammar	Operation	Compiling result
frecips vrz, vrx	$vrz = 1/vrx$	Only 32-bit instructions exist. frecips vrz, vrx

Data type Single-precision floating point

Description: Take the reciprocal of single-precision floating point in vrx and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = 1/vrx$

Grammar: frecips vrz, vrx

Data type Single-precision floating point

Description: Take the reciprocal of single-precision floating point in vrx and save the final result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	1	0

VRX VRZ

FSITOD – Transform signed integer into double-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fsitod vrz, vrx	$vrz = (\text{double})vrx$	Only 32-bit instructions exist. fsitod vrz, vrx

Data type 32-bit signed integer

Description: Transform signed integer in vrx into double-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = (\text{double})vrx$

Grammar: fsitod vrz, vrx

Data type 32-bit signed integer

Description: Transform signed integer in vrx into double-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20 19	16 15	5 4 3	0
----	----------	----------	-------	-------	---

1	1	1	1	0	1	0	0	0	0	0	VRX	0	0	0	1	1	0	1	0	1	0	0	0	0	VRZ
---	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

中大傳媒

CSKY

FSITOS – Transform signed integer into single-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fsitos vrz, vrx	vrz= (float)vrx	Only 32-bit instructions exist. fsitos vrz, vrx

Data type 32-bit signed integer

Description: Transform signed integer in vrx into single-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= (float)vrx

Grammar: fsitos vrz, vrx

Data type 32-bit signed integer

Description: Transform signed integer in vrx into single-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20 19	16 15	5 4 3	0
----	----------	----------	-------	-------	---

1	1	1	1	0	1	0	0	0	0	0	VRX	0	0	0	1	1	0	1	0	0	0	0	0	0	0	VRZ
---	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----



FSQRTD – Double-precision floating point square root

Unified instruction

Grammar	Operation	Compiling result
fsqrtd vrz, vrx	$vrz = \sqrt{vrx}$	Only 32-bit instructions exist. fsqrtd vrz, vrx

Data type Double-precision floating point

Description: Take the square root of double-precision floating point in vrx, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = \sqrt{vrx}$

Grammar: fsqrtd vrz, vrx

Data type Double-precision floating point

Description: Take the square root of double-precision floating point in vrx, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	1	0

VRX

VRZ

FSQRTS – Single-precision floating point square root

Unified instruction

Grammar	Operation	Compiling result
fsqrts vrz, vrx	$vrz = \sqrt{vrx}$	Only 32-bit instructions exist. fsqrts vrz, vrx

Data type Single-precision floating point

Description: Take the square root of single-precision floating point in vrx, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = \sqrt{vrx}$

Grammar: fsqrts vrz, vrx

Data type Single-precision floating point

Description: Take the square root of single-precision floating point in vrx, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	1	1	0

VRX

VRZ

FSTD – Store double-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fstd vrz, (rx, disp)	MEM[RX + zero_extend(offset << 2)] ← VRZ[63:0];	Only 32-bit instructions exist. fstd vrz, (rx, disp)

Description: Store double-precision floating point in register VRZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by two bits to 32 bits. The address space of FSTD instruction is +1KB.
 Attention: The offset DISP is gained after the binary operand {IMM4H,IMM4L} shifts left by two bits.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Store double-precision floating point in register VRZ to storage
 $MEM[RX + zero_extend(offset \ll 2)] \leftarrow VRZ[63:0];$

Grammar: fstd vrz, (rx, disp)

Description: Store double-precision floating point in register VRZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by two bits to 32 bits. The address space of FSTD instruction is +1KB.
 Attention: The offset DISP is gained after the binary operand

{IMM4H,IMM4L} shifts left by two bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25 24	21 20	16 15	8 7	4 3	0
1 1 1 1 0 1 0 IMM4H RX 0 0 1 0 0 1 0 1 IMM4L VRZ						

FSTM – Store vector floating point

Unified instruction

Grammar	Operation	Compiling result
fstm vrz, (rx, disp)	MEM[RX + zero_extend(offset << 3)] ← VRZ[63:0];	Only 32-bit instructions exist. fstm vrz, (rx, disp)

Description: Store two single-precision floating points in register VRZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by three bits to 32 bits. The address space of FSTM instruction is +1KB.
 Attention: The offset DISP is gained after the binary operand {IMM4H,IMM4L} shifts left by three bits.

**Influence on flag
bit:** No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Store two single-precision floating points in register VRZ to storage.

MEM[RX + zero_extend(offset << 3)] ← VRZ[63:0];

Grammar: fstm vrz, (rx, disp)

Description: Store two single-precision floating points in register VRZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by three bits to 32 bits. The address space of FSTM instruction is +1KB.

Attention: The offset DISP is gained after the binary operand

{IMM4H,IMM4L} shifts left by three bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25 24	21 20	16 15	8 7	4 3	0
1 1 1 1 0 1 0 IMM4H RX 0 0 1 0 0 1 1 0 IMM4L VRZ						

FSTMD – Store consecutive double-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fstmd vry-vrz, (rx)	<p>Store double-precision floating points in a group of consecutive register files to a group of consecutive storage addresses successively</p> <pre>src ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ MEM[addr] ← VRsrc[63:0]; src ← src + 1; addr ← addr + 8; }</pre>	<p>Only 32-bit instructions exist.</p> <p>fstmd vry-vrz, (rx);</p>

Description: Store double-precision floating points in a group of consecutive register files starting from VRY to a group of consecutive storage addresses successively. In other word, store contents in register VRY to the address of the first double word in the address appointed by storage; store the contents in register VR(Y+1) to the address of the second double word in the address appointed by storage, and the like; store the contents in register VRZ to the address of the last double word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Influence on flag bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation:

Store double-precision floating points in a group of consecutive register files to a group of consecutive storage addresses successively

```
src ← Y; addr ← RX;
```

```

for (n = 0; n <= (Z-Y); n++){
    MEM[addr] ← VRsrc[63:0];
    src ← src + 1;
    addr ← addr + 8;
}

```

Grammar: fstmd vry-vrz, (rx)

Description: Store double-precision floating points in a group of consecutive register files starting from VRY to a group of consecutive storage addresses successively. In other word, store contents in register VRY to the address of the first double word in the address appointed by storage; store the contents in register VR(Y+1) to the address of the second double word in the address appointed by storage, and the like; store the contents in register VRZ to the address of the last double word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Influence on No influence

flag bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31	30	26	25	24	21	20	16	15	8	7	4	3	0								
1	1	1	1	0	1	0	IMM4	RX	0	0	1	1	0	1	0	1	0	0	0	0	VRY

IMM4 field – Assign the number of destination registers, $IMM4 = Z - Y$.

0000 – 1 destination register

0001 – 2 destination registers

.....

1111 – 16 destination registers

FSTMM – Store consecutive vector floating point

**Unified
instruction**

Grammar	Operation	Compiling result
fstmm vry-vrz, (rx)	<p>Store vector floating points in a group of consecutive register files to a group of consecutive storage addresses successively</p> <pre>src ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ MEM[addr] ← VRsrc[63:0]; src ← src + 1; addr ← addr + 8; }</pre>	<p>Only 32-bit instructions exist.</p> <p>fstmm vry-vrz, (rx);</p>

Description: Store vector floating point (including two single-precision floating points) in a group of consecutive register files starting from VRY to a group of consecutive storage addresses successively. In other word, store contents in register VRY to the address of the first double word in the address appointed by storage; store the contents in register VR(Y+1) to the address of the second double word in the address appointed by storage, and the like; store the contents in register VRZ to the address of the last double word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Influence on flag No influence

bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

**32-bit
instruction**

Operation: Store vector floating points in a group of consecutive register files to a group of consecutive storage addresses successively

```
src ← Y; addr ← RX;
```

```

for (n = 0; n <=(Z-Y); n++){
    MEM[addr] ← VRsrc[63:0];
    src ← src + 1;
    addr ← addr + 8;
}

```

Grammar: fstmm vry-vrz, (rx)

Description: Store vector floating point (including two single-precision floating points) in a group of consecutive register files starting from VRY to a group of consecutive storage addresses successively. In other word, store contents in register VRY to the address of the first double word in the address appointed by storage; store the contents in register VR(Y+1) to the address of the second double word in the address appointed by storage, and the like; store the contents in register VRZ to the address of the last word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Influence on No influence

flag bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31	30	26	25	24	21	20	16	15	8	7	4	3	0
1	1	1	1	0	1	0	IMM4	RX	0	0	1	1	0

IMM4 field – Assign the number of destination registers, IMM4 = Z – Y.

0000 – 1 destination register

0001 – 2 destination registers

.....

1111 – 16 destination registers

FSTMS – Store consecutive single-precision floating point

**Unified
instruction**

Grammar	Operation	Compiling result
fstms vry-vrz, (rx)	<p>Store single-precision floating points in a group of consecutive register files to a group of consecutive storage addresses successively</p> <pre>src ← Y; addr ← RX; for (n = 0; n <= (Z-Y); n++){ MEM[addr] ← VRsrc[31:0]; src ← src + 1; addr ← addr + 4; }</pre>	<p>Only 32-bit instructions exist.</p> <p>fstms vry-vrz, (rx);</p>

Description: Store single-precision floating points in a group of consecutive register files starting from VRY to a group of consecutive storage addresses successively. In other word, store contents in register VRY to the address of the first word in the address appointed by storage; store the contents in register VR(Y+1) to the address of the second word in the address appointed by storage, and the like; store the contents in register VRZ to the address of the last word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Influence on flag No influence

bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

**32-bit
instruction**

Operation:

Store single-precision floating points in a group of consecutive register files to a group of consecutive storage addresses successively

```
src ← Y; addr ← RX;
```

```

for (n = 0; n <= (Z-Y); n++){
    MEM[addr] ← VRsrc[31:0];
    src ← src + 1;
    addr ← addr + 4;
}

```

Grammar: fstms vry-vrz, (rx)

Description: Store single-precision floating points in a group of consecutive register files starting from VRY to a group of consecutive storage addresses successively. In other word, store contents in register VRY to the address of the first word in the address appointed by storage; store the contents in register VR(Y+1) to the address of the second word in the address appointed by storage, and the like; store the contents in register VRZ to the address of the last word in the address appointed by storage. The effective address of storage is decided by the contents of base register RX.

Influence on No influence

flag bit:

Restriction: VRZ should be greater than or equal to VRY.

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31	30	26	25	24	21	20	16	15	8	7	4	3	0				
1	1	1	1	0	1	0	IMM4	RX	0	0	1	1	0	0	0	0	VRY

IMM4 field – Assign the number of destination registers, $IMM4 = Z - Y$.

0000 – 1 destination register

0001 – 2 destination registers

.....

1111 – 16 destination registers

FSTOD – Transform single-precision floating point into double-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fstod vrz, vrx	vrz= (double)vrx	Only 32-bit instructions exist. fstod vrz, vrx

Data type Single-precision floating point

Description: Transform single-precision floating point in vrx into double-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= (double)vrx

Grammar: fstod vrz, vrx

Data type Single-precision floating point

Description: Transform single-precision floating point in vrx into double-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20 19	16 15	5 4 3	0
----	----------	----------	-------	-------	---

1	1	1	1	0	1	0	0	0	0	0	VRX	0	0	0	1	1	0	1	0	1	1	1	0	VRZ
---	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	-----

CSKY 中大易

FSTOSI – Transform single-precision floating point into signed integer

Unified instruction

Grammar	Operation	Compiling result
fstosi.rm vrz, vrx where rm is rn/rz/rpi/rni	vrz= (signed long)vrx	Only 32-bit instructions exist. fstosi.rm vrz, vrx

Data type Single-precision floating point

Transform single-precision floating point in vrx into signed integer, and save the result in vrz. RM refers to the rounding mode.

RM represents:

2'b00: Round to nearest; the corresponding assembly instruction is fdtosi.rn

Description: 2'b01: Round to 0; the corresponding assembly instruction is fdtosi.rz

2'b10: Round to positive infinity; the corresponding assembly instruction is fdtosi.rpi

2'b11: Round to negative infinity; the corresponding assembly instruction is fdtosi.rni

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= (signed long)vrx

Grammar: fstosi.rm vrz, vrx

Data type Single-precision floating point

Transform single-precision floating point in vrx into signed integer, and save the result in vrz. RM refers to the rounding mode.

Description: RM represents:

2'b00: Round to nearest; the corresponding assembly instruction is fdtosi.rn

2'b01: Round to 0; the corresponding assembly instruction is fdtosi.rz

2'b10: Round to positive infinity; the corresponding assembly instruction is fdtosi.rpi

2'b11: Round to negative infinity; the corresponding assembly instruction is fdtosi.rni

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	VRX	0	0

FSTOUI – Transform single-precision floating point into unsigned integer

Unified instruction

Grammar	Operation	Compiling result
fstoui.rm vrz, vrx where rm is rn/rz/rpi/rni	vrz= (unsigned long)vrx	Only 32-bit instructions exist. fstoui.rm vrz, vrx

Data type Single-precision floating point

Transform single-precision floating point in vrx into unsigned integer, and save the result in vrz. RM refers to the rounding mode.

RM represents:

2'b00: Round to nearest; the corresponding assembly instruction is fdtosi.rn

Description: 2'b01: Round to 0; the corresponding assembly instruction is fdtosi.rz

2'b10: Round to positive infinity; the corresponding assembly instruction is fdtosi.rpi

2'b11: Round to negative infinity; the corresponding assembly instruction is fdtosi.rni

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= (unsigned long)vrx

Grammar: fstoui.rm vrz, vrx

Data type Single-precision floating point

Transform single-precision floating point in vrx into unsigned integer, and save the result in vrz. RM refers to the rounding mode.

Description: RM represents:

2'b00: Round to nearest; the corresponding assembly instruction is fdtosi.rn

2'b01: Round to 0; the corresponding assembly instruction is fdtosi.rz

2'b10: Round to positive infinity; the corresponding assembly instruction is fdtosi.rpi

2'b11: Round to negative infinity; the corresponding assembly instruction is fdtosi.rni

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	0	0	0	VRX	0	0

FSTRD – Store double-precision floating point in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
fstrd vrz, (rx, ry << 0)	Store double-precision floating point to storage	fstrd vrz, (rx, ry << 0)
fstrd vrz, (rx, ry << 1)		fstrd vrz, (rx, ry << 1)
fstrd vrz, (rx, ry << 2)	MEM[RX + RY << IMM2] ← VRZ[63:0]	fstrd vrz, (rx, ry << 2)
fstrd vrz, (rx, ry << 3)		fstrd vrz, (rx, ry << 3)

Description: Store double-precision floating point in register VRZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit

instruction

Operation: Store double-precision floating point to storage

MEM[RX + RY << IMM2] ← VRZ[63:0]

Grammar:

- fstrd vrz, (rx, ry << 0)
- fstrd vrz, (rx, ry << 1)
- fstrd vrz, (rx, ry << 2)
- fstrd vrz, (rx, ry << 3)

Description: Store double-precision floating point in register VRZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

fstrd vrz, (rx, ry << 0)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

fstrd vrz, (rx, ry << 1)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

fstrd vrz, (rx, ry << 2)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

fstrd vrz, (rx, ry << 3)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

FSTRM – Store vector floating point in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
fstrm vrz, (rx, ry << 0)	Store vector floating point to storage MEM[RX + RY << IMM2] ← VRZ[63:0]	Only 32-bit instructions exist.
fstrm vrz, (rx, ry << 1)		fstrm vrz, (rx, ry << 0)
fstrm vrz, (rx, ry << 2)		fstrm vrz, (rx, ry << 1)
fstrm vrz, (rx, ry << 3)		fstrm vrz, (rx, ry << 2)
		fstrm vrz, (rx, ry << 3)

Description: Store two single-precision floating points in register VRZ to storage.

Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Store vector floating point to storage

MEM[RX + RY << IMM2] ← VRZ[63:0]

Grammar:

- fstrm vrz, (rx, ry << 0)
- fstrm vrz, (rx, ry << 1)
- fstrm vrz, (rx, ry << 2)
- fstrm vrz, (rx, ry << 3)

Description: Store two single-precision floating points in register VRZ to storage.
Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value

gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

fstrm vrz, (rx, ry << 0)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

fstrm vrz, (rx, ry << 1)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

fstrm vrz, (rx, ry << 2)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

fstrm vrz, (rx, ry << 3)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0

FSTRS – Store single-precision floating point in register offset addressing

Unified instruction

Grammar	Operation	Compiling result
fstrs vrz, (rx, ry << 0)	Store single-precision floating point to storage	fstrs vrz, (rx, ry << 0)
fstrs vrz, (rx, ry << 1)		fstrs vrz, (rx, ry << 1)
fstrs vrz, (rx, ry << 2)	MEM[RX + RY << IMM2] ← VRZ[31:0]	fstrs vrz, (rx, ry << 2)
fstrs vrz, (rx, ry << 3)		fstrs vrz, (rx, ry << 3)

Description: Store single-precision floating point in register VRZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag bit: No influence

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit

instruction

Operation: Store single-precision floating point to storage

MEM[RX + RY << IMM2] ← VRZ[31:0]

Grammar:

- fstrs vrz, (rx, ry << 0)
- fstrs vrz, (rx, ry << 1)
- fstrs vrz, (rx, ry << 2)
- fstrs vrz, (rx, ry << 3)

Description: Store single-precision floating point in register VRZ to storage. Adopt the addressing mode of register and register offset. The effective address of storage is gained by adding the base register RX to the value gained by making offset register RY shift left by 2-bit immediate operand IMM2. The default value of IMM2 is 0.

Influence on flag No influence

bit:

Exception: Unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

fstrs vrz, (rx, ry << 0)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0		
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0	0	VRZ

fstrs vrz, (rx, ry << 1)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0		
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0	0	VRZ

fstrs vrz, (rx, ry << 2)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0		
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0	0	VRZ

fstrs vrz, (rx, ry << 3)

31	30	26	25	21	20	16	15	8	7	6	5	4	3	0		
1	1	1	1	0	1	RY	RX	0	0	1	0	1	1	0	0	VRZ

FSTS – Store single-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fst <i>vrz</i> , (<i>rx</i> , <i>disp</i>)	MEM[RX + zero_extend(offset << 2)] ← VRZ[31:0];	Only 32-bit instructions exist. fst <i>vrz</i> , (<i>rx</i> , <i>disp</i>)

Description: Store single-precision floating point in register VRZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by two bits to 32 bits. The address space of FSTS instruction is +1KB.

Attention: The offset DISP is gained after the binary operand {IMM4H,IMM4L} shifts left by two bits.

**Influence on flag
bit:** No influence

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

32-bit instruction

Operation: Store single-precision floating point in register VRZ to storage
MEM[RX + zero_extend(offset << 2)] ← VRZ[31:0];

Grammar: fst *vrz*, (*rx*, *disp*)

Description: Store single-precision floating point in register VRZ to storage. Adopt the addressing mode of register and unsigned immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 8-bit relative offset shifting left by two bits to 32 bits. The address space of FSTS instruction is +1KB.

Attention: The offset DISP is gained after the binary operand

{IMM4H,IMM4L} shifts left by two bits.

Influence on flag No influence

bit:

Exception: Unaligned access exception, unaligned access exception, access error exception, TLB unrecoverable exception, TLB mismatch exception, and TLB read invalid exception

Instruction

format:

31 30	26 25 24	21 20	16 15	8 7	4 3	0
1 1 1 1 0 1 0 IMM4H RX 0 0 1 0 0 1 0 0 IMM4L VRZ						

FSUBD – Double-precision floating point subtract

Unified instruction

Grammar	Operation	Compiling result
fsubd vrz, vrx,vry	$vrz = vrx - vry$	Only 32-bit instructions exist. fsubd vrz, vrx,vry

Data type Double-precision floating point

Description: Subtract double-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrx - vry$

Grammar: fsubd vrz, vrx,vry

Data type Double-precision floating point

Description: Subtract double-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FSUBM – SIMD single-precision floating point subtract

Unified instruction

Grammar	Operation	Compiling result
fsubm vrz, vrx,vry	$vrz[31:0] = vrx[31:0] - vry[31:0];$ $vrz[63:32] = vrx[63:32] - vry[63:32]$	Only 32-bit instructions exist. fsubm vrz, vrx,vry

Data type Single-precision floating point

Description: Subtract single-precision floating points in vrx[31:0] and vry[31:0], and save the result in vrz[31:0]; subtract single-precision floating points in vrx[63:32] and vry[63:32], and save the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz[31:0] = vrx[31:0] - vry[31:0]; vrz[63:32] = vrx[63:32] - vry[63:32]$

Grammar: fsubm vrz, vrx,vry

Data type Single-precision floating point

Description: Subtract single-precision floating points in vrx[31:0] and vry[31:0], and save the result in vrz[31:0]; subtract single-precision floating points in vrx[63:32] and vry[63:32], and save the result in vrz[63:32].

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20 19	16 15	5 4 3	0
----	----------	----------	-------	-------	---

1	1	1	1	0	1	0	VRY	0	VRX	0	0	0	1	0	0	0	0	0	0	0	0	1	0	VRZ
---	---	---	---	---	---	---	-----	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

CSKY 中大易

FSUBS – Single-precision floating point subtract

Unified instruction

Grammar	Operation	Compiling result
fsubs vrz, vrx,vry	$vrz = vrx - vry$	Only 32-bit instructions exist. fsubs vrz, vrx,vry

Data type Single-precision floating point

Description: Subtract single-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = vrx - vry$

Grammar: fsubs vrz, vrx,vry

Data type Single-precision floating point

Description: Subtract single-precision floating points in vrx and vry, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26	25	24	21	20	19	16	15	5	4	3	0
1	1	1	1	0	1	0	VRY	0	VRX	0	0	0

FUITOD – Transform unsigned integer into double-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fuitod vrz, vrx	$vrz = (\text{double})(\text{unsigned } 32)\text{vrx}[31:0]$	Only 32-bit instructions exist. fuitod vrz, vrx

Data type 32-bit unsigned integer

Description: Transform unsigned integer in vrx into double-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: $vrz = (\text{double})\text{vrx}$

Grammar: fuitod vrz, vrx

Data type 32-bit unsigned integer

Description: Transform unsigned integer in vrx into double-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20 19	16 15	5 4 3	0
----	----------	----------	-------	-------	---

1	1	1	1	0	1	0	0	0	0	0	VRX	0	0	0	1	1	0	1	0	1	0	1	0	VRZ
---	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	-----

CSKY 中大易

FUITOS – Transform unsigned integer into single-precision floating point

Unified instruction

Grammar	Operation	Compiling result
fuitos vrz, vrx	vrz= (float)vrx	Only 32-bit instructions exist. fuitos vrz, vrx

Data type 32-bit unsigned integer

Description: Transform unsigned integer in vrx into single-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

32-bit

instruction

Operation: vrz= (float)vrx

Grammar: fuitos vrz, vrx

Data type 32-bit unsigned integer

Description: Transform unsigned integer in vrx into single-precision floating point, and save the result in vrz.

Influence on None

flag bit:

Restriction: The range of register is vr0-vr15.

Exception: None

Instruction

format:

Instruction code

31	26 25 24	21 20 19	16 15	5 4 3	0
----	----------	----------	-------	-------	---

1	1	1	1	0	1	0	0	0	0	0	VRX	0	0	0	1	1	0	1	0	0	0	1	0	VRZ
---	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	-----

CSKY 中大傳媒

CSKY 中天傳媒