CSKY Architecture

USER GUIDE

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.

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

Revision history:

Revision	Date	Description	Author
1.0	12/01/2017	First release	Hangzhou C-Sky Microsystems Corporation

Table of contents:

LIST OF CHARTS:

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.

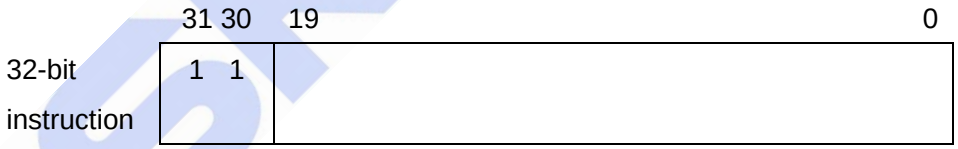
Main characteristics of CSKY instruction set architecture are as follows:

- 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.

1.1. 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.



	15 14	13 0
16-bit instruction	0 0	
	0 1	
	1 0	

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

1.2. Programming model

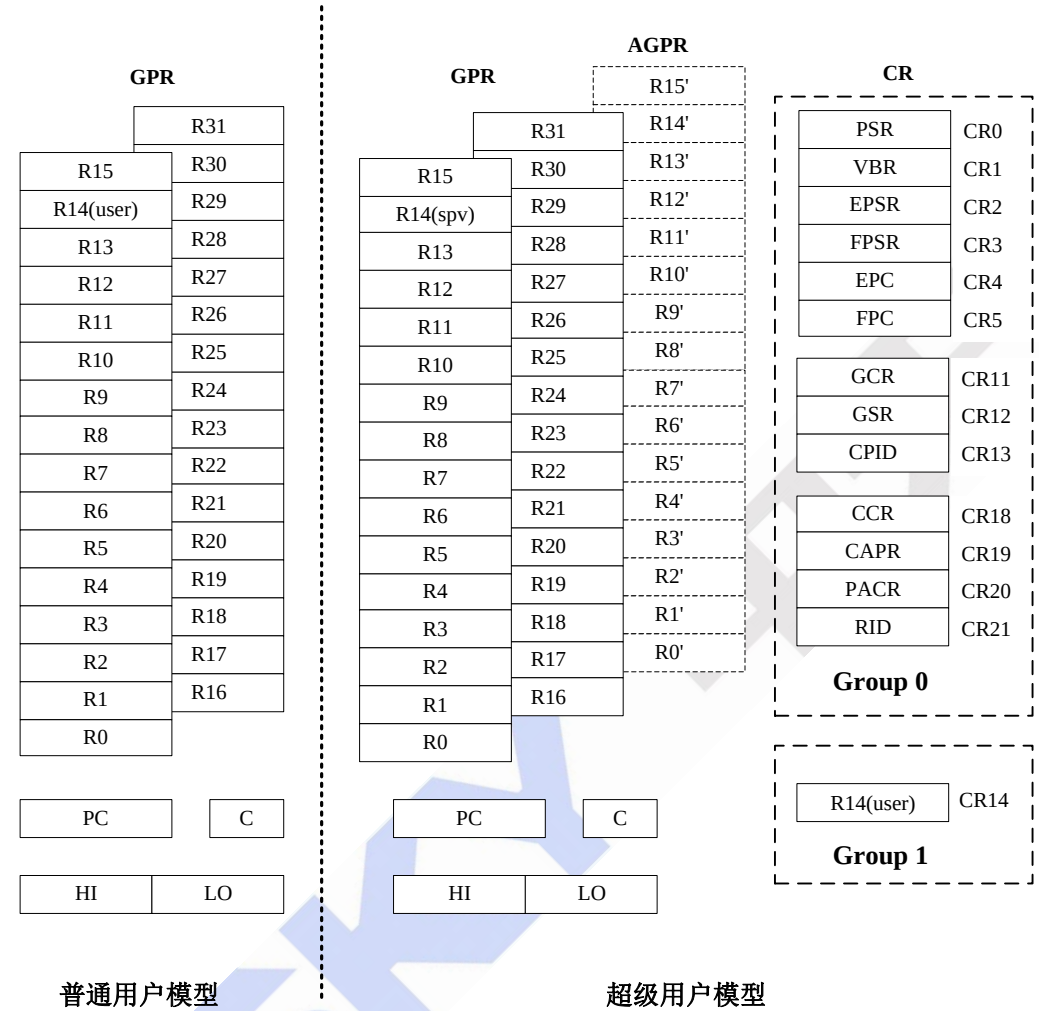


Chart 1-2 program model

CSKY defines two operation modes: user mode and supervisor mode. When the S bit in PSR is set, processor operates under supervisor mode. Also, it changes to supervisor mode after reset.

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; 3) access to the control register of tightly coupled IP. User mode is only allowed to access general purpose register while supervisor mode is allowed to access all general purpose register and control register. In this way, user mode program can be prevented from privilege information, and operating system provides management and service for user mode program by coordinating with the user mode program. While in supervisor mode, the program can access all control register of tightly coupled IP to schedule CPU resources.

In the user mode, the program is permitted to access user stack pointer (user SP), but not supervisor stack pointer (spv SP).

In the user mode, the program is permitted to access link register (LR) shared with supervisor mode.

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.

Most instructions can be used under user mode, except for some privilege ones such as stop, doze, wait, mfcr, mtcr, rte which may greatly influence the system. Besides, trap #n instruction can be used to change from user mode to supervisor mode.

In the supervisor mode, all general purpose registers and control registers can be accessed. In addition, both user SP and spv SP are accessible. To access user SP under supervisor mode, mtcr rx cr<14, 1> and mfcr rz cr<14, 1> instructions should be used.

1.3. Data format

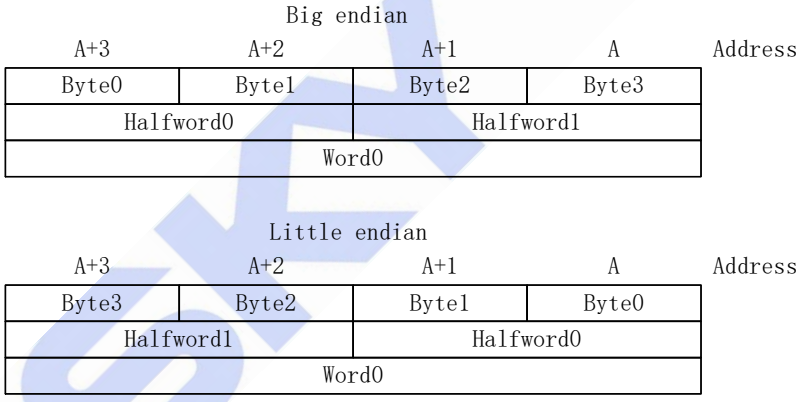


Chart 1-3 data structure in memory

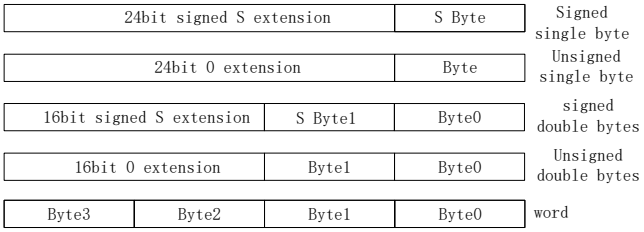


Chart 1-4 data structure in register

CSKY supports two's complement integer. The length of operand in every instruction can be explicitly encoded in the program (load/store instruction), or implicitly indicated in instruction operation (index operation, byte extraction). Usually, instructions generate 32-bit results with 32-bit operands.

The memory of CSKY can be configured to big endian or little endian. The highest byte of word 0 is located in the address 0 under big endian mode while it is located in the address 3 under little endian mode (default mode). The 31st bit is the MSB in the register.

2. Naming conventions

2.1. Symbol

The standard symbols and operators used in this document list below:

symbol	function
+	Add
-	Subtract
*	Multiply
/	Divide
>	Greater than
<	Less than
=	Equal
≥	Greater or equal than
≤	Less or equal than
1/4	Not equal
.	AND
+	OR
⊕	XOR
NOT	NOT
:	concatenate
⇒	Transfer
⇔	Exchange
±	deviation
0b0011	Binary number
0x0F	Hexadecimal number

2.2. Term

- Logic 1 represents the true value of Boolean logic.
- Logic 0 represents the false value of Boolean logic.
- Set means change one or more bits to logic 1 value.

- Clear means change one or more bits to logic 0 value.
- Reserved bit is reserved for function extension, and its value should be 0 without further explanation.
- Signal is an electric current or electromagnetic field used to convey data by transforming states.
- Pin is an external electric physical connection, and a single pin can be connected with multiple signals.
- Enable means make some discrete signal in an effective state:

  Active-low signal changes from high to low;

  Active-high signal changes from low to high.
- Disable means make some enabled signal change state:

  Active-low signal changes from low to high;

  Active-high signal changes from high to low.
- LSB represents the lowest significant bit, and MSB represents the highest significant bit.

  The memory and register apply big endian mode when “pad_sysio_bigend_b=0”, which means the highest byte locates at the lowest address. Namely, a word start with the highest byte (bit 31-24).
- Little endian is adopted when “pad_sysio_bigend_b=1”.
- Signal, bit field, control bit use a common rule.
- Identifier followed by the numbers from high to low which indicating range, represents a set of signal. For example, addr[4:0] represents a set of address bus, and addr[4] is the MSB while addr[0] is the LSB.
- Single identifier represents single signal. For example, pad_cpu_rst_b means a single signal. Sometimes it is meaningful to add number after identifier. For instance, addr15 means the 16th bit of a set of bus.

3. Register description

This chapter mainly introduces the architecture of general purpose registers and control registers in CSKY under user mode or supervisor mode.

3.1. User mode program model

Chart 3-1 lists some registers under user model:

- 32 32-bit general purpose registers(R31~R0)
- 32-bit program counter(PC)
- Condition / carry bit(C bit)

Name	Function
R0	Undetermined, the first parameter called by function
R1	Undetermined, the second parameter called by function
R2	Undetermined, the third parameter called by function
R3	Undetermined, the fourth parameter called by function
R4	Undetermined
R5	Undetermined
R6	Undetermined
R7	Undetermined
R8	Undetermined
R9	Undetermined
R10	Undetermined
R11	Undetermined
R12	Undetermined
R13	Undetermined
R14(user)	Stack pointer(user model)
R15	Link register
R16	Undetermined
R17	Undetermined

R18	Undetermined
R19	Undetermined
R20	Undetermined
R21	Undetermined
R22	Undetermined
R23	Undetermined
R24	Undetermined
R25	Undetermined
R26	Undetermined
R27	Undetermined
R28	Undetermined
R29	Undetermined
R30	Undetermined
R31	Undetermined

PC	Program counter

C	condition/carry bit

Chart 3-1 user model register

3.1.1.General purpose register

General purpose registers contains operands, results and address information of the instruction. For both hardware and software, these registers are conventionally used for link call of subprogram, parameter pass and stack pointer, etc.

Among those registers, R14 is the user stack pointer under user mode, and its index is same as other general purpose registers.

3.1.2.Program counter

Program counter contains the current address of executing instructions. The value of program counter cannot be modified by instruction directly, instead the processor will automatically accumulate program counters or place a new value into the program counter according to the program operation situations during normal operation or exception handling of program. 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.

3.1.3.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 XTRB[0-3] will influence the value of condition/carry bit.

3.2. Supervisor mode program model

System programmer utilizes supervisor mode to set system operation function, I/O control and other restricted operation.

Supervisor model consists of general purpose registers and other registers below, listed as Chart 3-2:

- 1 supervisor mode stack pointer register(R14)
- Processor Status Register(PSR);
- Vector Base Register(VBR);
- Exception Program Counter (EPC);
- Exception Processor Status Register (EPSR);
- 32-bit Global Control Register (GCR)(configurable width)*;
- 32-bit Global Status Register (GSR)(configurable width)*;
- CPU Identification Register (CPUIDR);
- Cache Configure register (CCR);
- Cacheability and Access Permission Register (CAPR) *;
- Protection region Control Register (PACR) *;
- Protection Region Select Register (PRSR) *;
- CPU Hint Register (CHR).

*note: The selectable register is valid only for specific configurations.

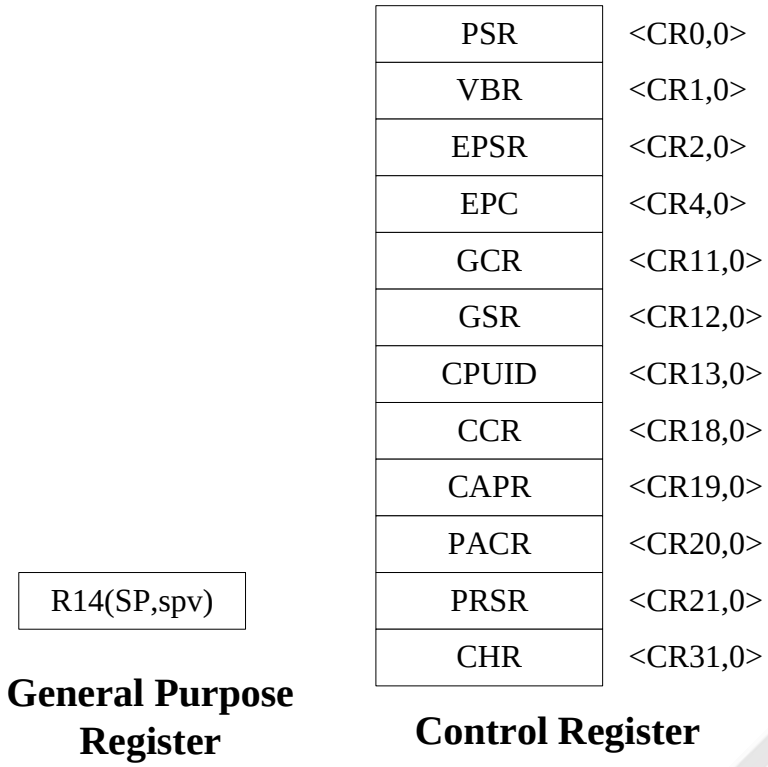


Chart 3-2 additional resources in supervisor model

3.2.1.Supervisor mode stack pointer R14(spv SP)

In supervisor mode, indexing R14 will automatically select R14(spv SP), which is used as the stack pointer for supervisor mode.

In the meantime, R14(user sp) is not accessible directly. To access user SP under supervisor mode, mfcr/mtcr cr<14, 1> instructions should be used.

3.2.2.Processor status register(PSR,CR<0,0>)

Processor status register (PSR) stores the current status and control information of processor, including C bit, interrupt enable bit and other control bits. In the supervisor mode, software is able to access processor status register (PSR). And the S bit in the PSR indicates whether the processor stays at supervisor mode or user mode. In addition, other control bits in PSR indicate whether the EPSR and EPC can store the current content or whether the interrupt is effective.

	31	30						24	23							16
	S	0							VEC［7:0］							
Reset	1	0							0							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	00		0	0	0		MM	EE	IC	IE	0	0	0	0	0	C

Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Chart 3-3 processor status register

S-Supervisor mode set bit:

When S equals 0, the processor works under user mode;

When S equals 1, the processor works under supervisor mode;

This bit is set by hardware during reset or entering exception handler.

VEC[7:0]-exception VECtor:

When exception happens, these bits can be used to calculate the entrance address of exception handler, and it will be cleared to zero during reset.

MM-Misalign exception Mask bit:

When MM equals 0, misalign exception will happen if the address of load/store instruction is misaligned.

When MM equals 1, misalign exception will not happen even if the address of load/store instruction is misaligned. If the processor support misalign access, this misalign address will be used to access memory. If the processor doesn't support misalign access, the misaligned address will be transformed to the aligned address (set lower bits to 0) in order to access the memory. However, in any situation, if the address of multi-cycle memory access instructions (such as STM, LDM, PUSH, POP, NIE, NIR, IPUSH, IPOP, etc.) is misaligned, the misalign exception will happen.

Misalign access operations list below:

- The address 1, 2, 3 of word read access will behave as two word-read accesses on the bus, and the address is 0 and 4 respectively.
- The address 1 of half-word read access will behave as one word-read accesses on the bus, and the address is 0.
- The address 3 of half-word read access will behave as two word-read accesses on the bus, and the address is 0 and 4 respectively.
- The address 1 of word write access will behave as an address 1 byte write, an address 2 half-word write and an address 4 byte write one the bus.
- The address 2 of word write access will behave as an address 2 half-word write and an address 4 half-word write on the bus.
- The address 3 of word write access will behave as an address 3 byte write, an address 4 half-word write and an address 6 byte write on the bus.
- The address 1 of half-word write access will behave as two byte-write accesses on the bus, and the address is 1 and 2 respectively.
- The address 3 of half-word write access will behave as two byte-write accesses on the bus, and the address is 3 and 4 respectively.

This bit will be cleared to zero during reset.

EE-Exception Enable bit:

When EE equals 0, exception is not effective, and any exception except interrupt will be recognized as unrecoverable exception.

When EE equals 1, exception is effective, all exceptions will be normally responded by using EPSR and EPC.

This bit will be cleared to zero during reset or when the exception is responded by processor.

IC-Interrupt Control bit:

When IC equals 0, interrupt can only be responded between instructions;

When IC equals 1, interrupt can be responded before completion of multi-cycle instructions.

This bit will be cleared to zero during reset, not affected by other exceptions.

IE-Interrupt Enable bit:

When IE equals 0, interrupt is not valid, neither are EPC and EPSR;

When IE equals 1, interrupt is valid (EE bit should be 1, otherwise the interrupt is still invalid);

This bit will be cleared to zero during reset or when the exception is responded by processor.

C-Condition / Carry bit

This bit is used for the condition judgment bit of some instructions.

This bit will be cleared to zero during reset.

3.2.2.1. Update PSR

PSR can be updated in several different ways, which will result in different influence.it can be modified by exception response, exception handling and execution of instructions like,rte,mtcr. The modification can be implemented in four ways.

- Exception response and exception handling updating PSR:

Updating PSR is part of calculating the entrance address of exception handler, and it includes S, VEC, IE, EE bits in PSR. The priority of modifying S, VEC, IE, EE bits is higher than calculating the entrance address of exception handler. And changing VEC bit has higher priority than the execution of first instruction in exception handler.

- RTE instruction updating PSR:

Updating PSR is part of the execution of rte instruction, and it may include all bits in PSR. The priority of modifying S, IE, EE bits is higher than fetching the return PC. And changing VEC, MM, IC and C bits has higher priority than the execution of first instruction after return.

- MTCR instruction updating PSR:

If the target register is CR<0, 0>, updating PSR will be part of the execution of mtcr instruction, and it may changes all bits in PSR. The new PSR value will be used by the following instructions, exceptions and interrupt response.

3.2.3. Vector Base Register (VBR, CR<1, 0>)

VBR is used for storing the base address of exception vector. It contains 22 high valid bits and 10 reserved bits (the value is 0). The value of VBR is 0X00000000 after reset.

	31	10	9	0
	VECTOR BASE		RESERVED	
Reset	0		0	

Chart 3-4 vector base register

3.2.4. Exception Register(CR<2,0>～CR<5,0>)

EPSR and EPC is used for storing current contents when exception occurs. Please refer to chapter 6 for detailed information.

3.2.5. Global Control Register (GCR, CR<11, 0>)

Global control register is used to control external devices and events, and it can control specific ones by parallel output interface on the chip. Generally speaking, GCR can be used to manager power, control device, handle events and perform other basic functions by simple setting. User may define which bit in GCR corresponding to which function, and all bits in GCR is readable and writable. Also, the width of GCR is hardware configurable.

3.2.6. Global Status Register (GSR, CR<12, 0>)

Global status register is used to mark external devices and events, and it can convey external status to the inside of CPU by the input interface on the chip in order to achieve monitoring. Generally speaking, GSR can be used to monitor the status of external devices and events, and GSR is readable. Also, the width of GSR is hardware configurable.

3.2.7. CPU Identification Register (CPUIDRR, CR<13, 0>)

This register is used for storing the internal number of the product developed by C-Sky Microsystems Corporation, and it is read-only register. The value after reset is determined by product itself.

3.2.8. Cache Configure Register (CCR, CR<18, 0>)

Cache configure register is used to configure memory protection region, Endian mode and core-to-bus clock ratio.

31 14	13	12 11	10 8	7	6 2	1 0
0	BE_V2	0	SCK	BE	0	MP
Reset 0			-	-	0	0

Chart 3-5 Cache Configure register

BE_V2-V2 endian mode:

When BE_V2 equals 0, not V2 endian mode;

When BE_V2 equals 1, V2 endian mode;

This bit along with BE decides under which endian mode the processor works. This bit is valid only when BE equals 1.

BE_V2 should not be changed after being configured during power on reset, and it has corresponding pin on the CPU.

SCK-core-to-bus clock ratio:

This bit is used to indicate the clock ratio between system bus and CPU core. Its calculation formula is RATIO＝SCK＋1,and CPU has corresponding pin. SCK should not be changed after being configured during power on reset

It has no specific function now, except for software searching.

BE-Endian mode:

When BE equals 0, little endian;

When BE equals 1, big endian;

BE should not be changed after being configured during power on reset, and it has corresponding pin on the CPU.

MP-Memory protection setting bit:

MP is used to set whether MPU is valid, which lists below:

MP	Function
00	MPU invalid
01	MPU valid

Chart 3-6 Memory protection setting

3.2.9. Cacheability and Access Permission Register (CAPR, CR<19, 0>)

Each bit of CAPR lists below:

31	30	29	28	27	26	25	24	23 22	21 20	19 18	17 16
S7	S6	S5	S4	S3	S2	S1	S0	AP7	AP6	AP5	AP4

Reset	0	0	0	0	0	0	0	0	0	0	0	0				
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AP3		AP2		AP1		AP0		NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0
Reset	0		0		0		0		0	0	0	0	0	0	0	0

Chart 3-7 Cacheability and Access Permission Register

NX0~NX7-inexecutable attribute setting bit:

When X equals 0, the area is executable;

When X equals 1, the area is inexecutable.

Note: when processor fetches instruction from inexecutable area, access error exception will occur.

S0~S7-secure attribute setting bit:

When S equals 0, the area is not secure;

When S equals 1, the area is secure.

Note: This bit determines the secure attribute of memory access, and this attribute will be conveyed to bus.

AP0~AP7-access permission setting bit:

AP	Supervisor permission	User permission
00	Not accessible	Not accessible
01	Readable and writable	Not accessible
10	Readable and writable	Read-only
11	Readable and writable	Readable and writable

Chart 3-8 access permission setting

3.2.10. Protection region Control Register (PACR,CR<20,0>)

Each bit of PACR lists below:

	31									10	9				6	5				1	0
	Base Address										0					Size					E
Reset	-										0					-					0

Chart 3-9 Protection region Control Register

Base Address-The higher address bits of protection region:

The register indicates the higher address bits of protection region, and the written base address should be aligned with the size of page. For example, if the page size is 8M, CR<20, 0>[22:10] should be 0. The specific requirement of each page lists below: Chart

3-10 Protection region size setting and base address requirement .

Size-Protection region size:

The size of protection region ranges from 1KB to 4GB, and it can be calculated through formula: protection region ＝2^(Size+1). Hence, the value of size should range from 01001 to 11111, otherwise some unpredictable result may occur.

Size	Protection region size	Base address requirement
00000—01000	Reserved	—
01001	1KB	No requirement
01010	2KB	CR<20,0>.bit[10]=0
01011	4KB	CR<20,0>.bit[11:10]=0
01100	8KB	CR<20,0>.bit[12:10]=0
01101	16KB	CR<20,0>.bit[13:10] =0
01110	32KB	CR<20,0>.bit[14:10] =0
01111	64KB	CR<20,0>.bit[15:10] =0
10000	128KB	CR<20,0>.bit[16:10] =0
10001	256KB	CR<20,0>.bit[17:10] =0
10010	512KB	CR<20,0>.bit[18:10] =0
10011	1MB	CR<20,0>.bit[19:10] =0
10100	2MB	CR<20,0>.bit[20:10] =0
10101	4MB	CR<20,0>.bit[21:10] =0
10110	8MB	CR<20,0>.bit[22:10] =0
10111	16MB	CR<20,0>.bit[23:10] =0
11000	32MB	CR<20,0>.bit[24:10] =0
11001	64MB	CR<20,0>.bit[25:10] =0
11010	128MB	CR<20,0>.bit[26:10] =0
11011	256MB	CR<20,0>.bit[27:10] =0
11100	512MB	CR<20,0>.bit[28:10] =0
11101	1GB	CR<20,0>.bit[29:10] =0
11110	2GB	CR<20,0>.bit[30:10] =0
11111	4GB	CR<20,0>.bit[31:10] =0

Chart 3-10 Protection region size setting and base address requirement

E-Protection region enable setting:

When E equals 0, protection region is disabled;

When E equals 1, protection region is enabled.

3.2.11. Protection Region Select Register(PRSR,CR<21,0>)

PRSR is used to select the current protection region, and each bit lists below:

	31 ... 3	2 ... 0
	0	RID
Reset	0	-

Chart 3-11 protection region select register

RID-Protection region index:

RID indicates the selected protection region. For example, 000 represents the 0th protection region.

3.2.12. MPU operation

3.2.12.1. MPU enable control

The 0th bit of CR<20, 0> is the MPU enable bit. Before MPU becomes valid, At least one region should be specified and the corresponding NX, S, AP bits should be set. In addition, The MPU enable instruction must be located at the place where the address is valid. In other words, the address should not be in MPU denied region. Otherwise, unpredictable results may occur.

3.2.12.2. Memory access handling

When MPU is enabled, MPU will check whether the current access address locates at the protection region:

If the address is not located in any region, the memory access will be stopped;

If the address is located in one or more of those regions, the access will be controlled by the highest indexed region which has been enabled. (The highest is 7 while the lowest is 0).

3.2.12.3. Memory access start address setting

CR<20, 0> defines the start address and size of 4/8 protection regions. The size of protection region must be a power of 2, ranging from 1KB to 4GB. And the start address must be aligned with the size of region. For example, the start address of an 8KB protection region could be 32'h12346000. However, for a 16KB protection region, this start address is not valid, instead 32'h12344000 could be a valid one.

3.2.13. CPU Hint Register(CHR,CR<31,0>)

CR<31, 0> is used to implement all kinds of CPU hint operation in processor, which includes software reset and interrupt response acceleration.

	31 16	15 5	4	3 0
	SRST_VAL	0	IAE	0
Reset	0	0	0	0

Chart 3-12 CPU hint register

Software reset decision value SRST_VAL:

If SRST_VAL is written into a certain value, the software reset will be implemented. The default value is 16'hABCD.

The software reset operation will reset all general purpose registers, control registers. And a software reset identified signal lasting one system clock period will be sent outside by the processor.

If the processor executes software reset instruction under normal running mode, then it will enter the reset exception handling program (i.e. Exception handler with vector number 0) to execute corresponding operation; However, if the processor executes the instruction under debug mode, then the processor will remains at debug mode. Instead, it will automatically enter the exception handling program (i.e. Exception handler with vector number 0) to execute corresponding operation when exiting debug mode.

Any reading operation for SRST_VAL will unconditionally returns 0, and writing value other than the specific one to the register will not produce any result.

The software reset operation above must be executed on the condition that the exception enable bit (EE bit in the PSR) has been set. Otherwise unrecoverable exception will occur.

Interrupt response acceleration enable bit IAE:

When IAE equals 1, the interrupt response acceleration mechanism is enabled. The processor will push the current contents into stack speculatively which accelerates the interrupt response.

When IAE equals 0, the interrupt response acceleration mechanism is disabled.

The above interrupt response acceleration mechanism controlled by IAE bit is only valid when the interrupt nesting instructions NIE,NIR,IPUSH,IPOP are hardware configured.

3.2.14. User mode stack pointer register 14(R14(user),CR<14,1>)

In the supervisor mode, R14(user) is mapped to the control register CR<14,1>. In other words, supervisor can access CR<14, 1> to control R14(user).

4. 32-bit Instructions

This chapter mainly focus on the 32-bit instruction set of CSKY, including its functionality, coding and addressing mode.

4.1. Functional Classification of 32-bit Instructions

According to functions, 32 bit instructions of CSKY can be classified as following categories:

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

4.1.1. Data Operation Instructions

Data operation instructions can be further classified as：

Add/Sub Instructions：

Chart 4-1 list of 32-bit Add/Sub Instructions

ADDU32	Unsigned-add
ADDC32	Unsigned-add with carry
ADDI32	Unsigned-add immediate
SUBU32	Unsigned-sub
SUBC32	Unsigned-sub with carry
SUBI32	Unsigned-sub immediate
RSUB32	Reversed sub
IXH32	Indexed half-word
IXW32	Indexed word
INCF32	Add immediate with carry bit 0
INCT32	Add immediate with carry bit 1
DECF32	Sub immediate with carry bit 0
DECT32	Sub immediate with carry bit 1

Logic Instructions：

Chart 4-2 list of 32-bit Logic Instructions

AND32	Bitwise logic and
ANDI32	Bitwise logic and immediate
ANDN32	Bitwise logic nand
ANDNI32	Bitwise logic nand immediate
OR32	Bitwise logic or

ORI32	Bitwise logic or immediate
XOR32	Bitwise logic xor
XORI32	Bitwise logic xor immediate
NOR32	Bitwise logic nor
NOT32	Bitwise logic not

Shift instructions：

Chart 4-3 list of 32-bit Shift Instructions

LSL32	Logic shift left
LSLI32	Logic shift left immediate
LSLC32	Logic shift left immediate to carry bit
LSR32	Logic shift right
LSRI32	Logic shift right immediate
LSRC32	Logic shift right immediate to carry bit
ASR32	Arithmetic shift right
ASRI32	Arithmetic shift right immediate
ASRC32	Arithmetic shift right immediate to carry bit
ROTL32	Circular shift left
ROTLI32	Circular shift left immediate
XSR32	Extended shift right

Comparison instructions：

Chart 4-2 list of 32-bit comparison instructions

CMPNEI32	Not equal immediate
CMPHSI32	Unsigned lager or equal immediate
CMPLTI32	Signed smaller immediate

Data Transfer Instructions：

Chart 4-3 list of 32-bit data transfer instructions

MOV32	Data Transfer
MOVF32	Data Transfer with carry bit 0
MOVT32	Data Transfer with carry bit 1
MOVI32	Data Transfer immediate
MOVIH32	High word data transfer immediate
MVC32	Carry bit data transfer
LRW32	Data transfer from memory

Bit instructions：

Chart 4-4 list of 32-bit Bit instructions

BCLRI32	Clear immediate
BSETI32	Set immediate
BTSTI32	Test immediate

Extract and Insert Instructions：

Chart 4-5 list of 32-bit Extract and Insert Instructions

XTRB0.32	Extract byte 0 and unsigned extend
XTRB1.32	Extract byte 1 and unsigned extend
XTRB2.32	Extract byte 2 and unsigned extend
XTRB3.32	Extract byte 3 and unsigned extend

Mul and Div Instructions:

Chart 4-6 list of 32-bit Mul and Div instruction

MULT32	Multiply

Other Arithmetic Instructions：

Chart 4-7 list of other 32-bit arithmetic Instructions

FF0. 32	Fast find 0
FF1. 32	Fast find 1
BMASKI32	Bit mask generation immediate
BGENI32	Bit generation immediate

4.1.2. Branch Jump Instructions

Branch jump instructions can be further classified as：

Branch instructions：

Chart 4-8 list of 32-bit branch instructions

BT32	Branch instruction with carry bit 1
BF32	Branch instruction with carry bit 0

Jump instructions：

Chart 4-9 list of 32-bit jump instruction

BR32	Unconditional jump
BSR32	Jump to Subprogram
RTS32	Link register jump

4.1.3. Memory Access Instruction

Memory access instructions can be further classified as：

Immediate operand offset access instructions

Chart 4-10 List of 32-bit Immediate operand offset access instruction

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
ST32.B	Store byte
ST32.H	Store half-word
ST32.W	Store word

Multi-register access instructions:

Chart 4-13 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

4.1.4. Privileged Instruction

Privileged instruction can be further divided into:

Control register operation instructions:

Chart 4-14 List of 32-bit control register operation instructions

MFCR32	Read from control register
MTCR32	Write to control register

Low power consumption instructions:

Chart 4-11 List of 32-bit low power consumption instructions

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

Abnormal return instruction:

Chart 4-12 List of 32-bit abnormal return instructions

RTE32	Return from exception/interrupt

4.1.5. Special Function Instructions

Special function includes

Chart 4-13 List of 32-bit Special function

SYNC32	Synchronize CPU
TRAP32	Unconditional operating system trap

4.2. 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)

4.2.1. 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.

4.2.2. Immediate operand type

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

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	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 or 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.

4.2.3. 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 which adopts one-hot coding mode.

4.3. Operand Addressing Mode of 32-bit Instructions

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

4.3.1. Addressing Mode of Jump-type Instructions

The 32-bit instructions of jump type in CSKY v2 only have one addressing mode.

4.3.1.1. 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

4.3.2. Addressing Mode of Immediate Operand-type Instructions

The 32-bit instructions of immediate operand type in CSKY V2 have four addressing modes

4.3.2.1. 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 30	29 26	25 21	20 16	15 0
1 1	OP	RZ	RX	IMM16
2	4	5	5	16

4.3.2.2. 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 directly participate in data operation as 16-bit immediate operand. Instructions of this format include cmphsi32, cmplti32.

31 30	29 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 be used in data operation as 16-bit immediate operand or be left to user for customization. Instructions of this format include movi32, movih32 and lrw32.

31 30	29 26	25 21	20 16	15 0

1 1	OP	SOP	RZ	IMM16
2	4	5	5	16

4.3.2.3. 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 which is considered as offset and can be used to generate destination address. Instructions of this format include br32, bf32, bt32.

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

4.3.2.4. Addressing Mode of Two Registers 12-bit Immediate Operand

In the instructions that adopt the addressing mode of two registers 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.bs, ld32.hs, st32.b, st32.h, addi32, subi32, andi32, andni32 and xori32.

31 30 29	26 25	21 20	16 15	12 11	0
1 1	OP	RZ	RX	SOP	IMM12
2	4	5	5	4	12

4.3.3. Addressing Mode of Register-type Instructions

The 32-bit instructions of register type in CSKY V2 have five addressing modes.

4.3.3.1. Addressing Mode of Ternary Registers

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、and32、andn32、or32、xor32、nor32、lsl32、lsr32、asr32、rotl32、mult32.

31 30 29	26 25	21 20	16 15	10 9	5 4	0
1 1	OP	RY	RX	SOP	Pcode	RZ
2	4	5	5	6	5	5

4.3.3.2. 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 lsli32、lsri32、asri32、rotli32、lslc32、lsrc32、asrc32、xsr32、bclri32、bseti32.

31 30	29 26	25 21	20 16	15 10	9 5	4 0
1 1	OP	IMM5	RX	SOP	Pcode	RZ
2	4	5	5	6	5	5

In the second format, IMM5 field is the 5-bit immediate operand and used 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、dect32.

31 30	29 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

4.3.3.3. Addressing Mode of Two Register

In the 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 xtrb0.32、xtrb1.32、xtrb2.32、xtrb3.32、ff0.32、ff1.32.

31 30	29 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

4.3.3.4. 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 30	29 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

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 30	29 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

4.3.3.5. Addressing Mode of Single Register

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.

31 30	29 26	25 21	20 16	15 10	9 5	4 0
1 1	OP	0	0	SOP	Pcode	RZ
2	4	5	5	6	5	5

5. 16-bit Instruction Set

In this chapter, 16-bit instruction set of CSKY will be introduced, covering its functional classification, encoding and addressing mode.

5.1. Functional Classification of 16-bit Instructions

According to functions of instruction realization the 16-bit instruction set can be classified into following categories:

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

5.1.1. Data Operation Instruction

Data operation instruction can be further divided into:

Add/Sub instruction:

ADDU16	Add unsigned
ADDC16	Add unsigned with carry
ADDI16	Add unsigned immediate
SUBU16	Sub unsigned
SUBC16	Sub unsigned with carry
SUBI16	Sub unsigned immediate

Chart 5-1 List of 16-bit Add/Sub instructions

Logical operation instruction:

AND16	Bitwise logical and
ANDN16	Bitwise logical and-not
OR16	Bitwise logical or
XOR16	Bitwise logical xor
NOR16	Bitwise logical or-not
NOT16	Bitwise logical not

Chart 5-2 List of 16-bit Logical operation instructions

Shift instruction:

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

Chart 5-3 List of 16-bit Shift instructions

Compare instruction:

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

Chart 5-4 List of 16-bit compare instructions

Data transfer instruction:

MOV16	Move
MOVI16	Move immediate
MVCV16	Carry bit data transfer
LRW16	Read from memory

Chart 5-5 List of 16-bit data transfer instructions

Bit operation instruction:

BCLRI16	Bit clear immediate
BSETI16	Bit set immediate
BTSTI16	Bit test immediate

Chart 5-6 List of 16-bit bit operation instructions

Extract and insert instruction:

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	Extract half-word and extend signed
REVH16	Half-word byte-reverse

Chart 5-7 List of 16-bit extract and insert instructions

Multiply-divide instruction:

MULT16	Multiply

Chart 5-8 List of 16-bit multiply-divide instructions

5.1.2. Branch Jump Instruction

Branch jump instruction can be further divided into:

Branch instruction:

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

Chart 5-9 List of 16-bit branch instructions

Jump instruction:

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

Chart 5-9 List of 16-bit jump instructions

5.1.3. Memory Access Instruction

Memory access instruction can be further divided into:

Immediate operand offset access instruction:

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

Chart 5-10 List of 16-bit immediate operand offset access instructions

Multi-register access instructions：

POP16	Pop
IPOP16	Interrupt pop
PUSH16	Push
IPUSH16	Interrupt push
NIE16	Interrupt nesting enable
NIR16	Interrupt nesting return

Chart 5-11 List of 16-bit Multi-register access instruction

note：NIE16 and NIR16 need to be executed in supervisor mode

5.1.4. Privilege Instructions

16-bit Privilege instructions ：

NIE16	Interrupt nesting enable
NIR16	Interrupt nesting return

Chart 5-12 List of 16-bit Privilege instructions

note：NIE16 和 NIR16 are Multi-register access instructions at the same time.

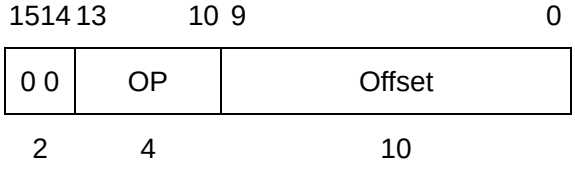
5.2. Coding Mode of 16-bit Instructions

The 16-bit instruction set of CSKY V2 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)

5.2.1. Jump Type

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

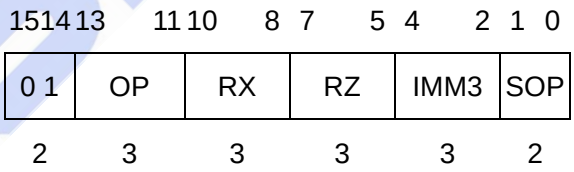


OP field is the main operation code and instructions of this coding type can be identified through 4-bit main operation code; Offset field is the offset of jump instruction.

5.2.2. 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.

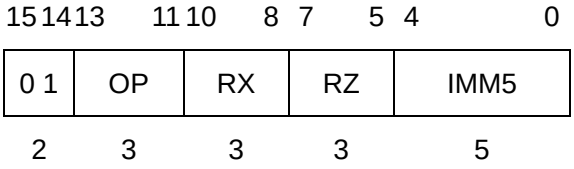
The coding mode of 3-bit immediate operand is shown in the following chart:



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:



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

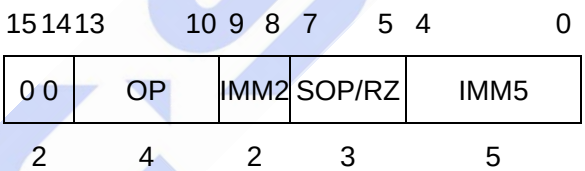
The second coding mode of 5-bit immediate operand is shown in the following chart:

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

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.

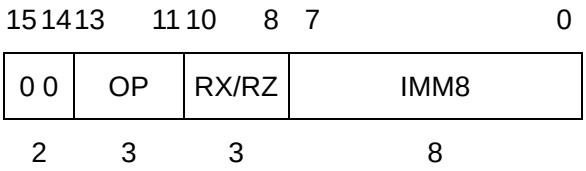
The coding mode of 7-bit immediate operand is shown in the following chart:



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

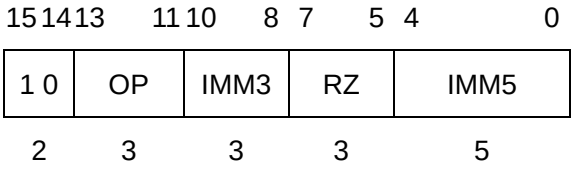
and five low bits of 7-bit immediate operand; SOP/RZ field is the sub-operation code field or destination register field. The instruction type can be gained after decoding the main operation code OP, and the specific instruction can be obtained only after further decoding the sub-operation code SOP.

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



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:

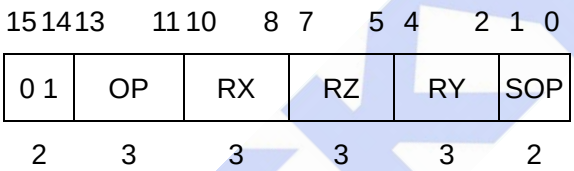


OP field is the main operation code and the instruction or instruction type can be identified through 3-bit main operation code; IMM3 field and IMM5 field are three high bits and five low bits of 8-bit immediate operand; RZ field is the destination register field.

5.2.3. Register Type

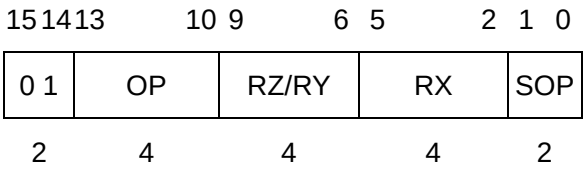
Register type (R type) covers two coding modes including 3-bit operand and 2-bit operand.

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



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:



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.

5.3. Operand Addressing Mode of 16-bit Instructions

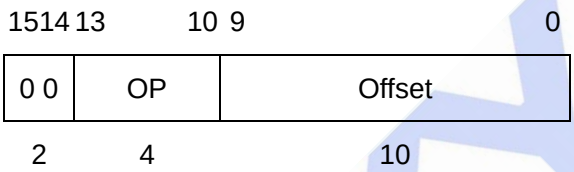
The 16-bit instruction set of CSKY follows three instruction coding modes and each coding mode has its own operand addressing mode. In the following, various operand addressing modes will be introduced.

5.3.1. Addressing Mode of Jump-type Instructions

The 16-bit instructions of jump type in CSKY V2 only have one addressing mode.

5.3.1.1. 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.



5.3.2. Addressing Mode of Immediate-type Instructions

The 16-bit instructions of immediate operand type in CSKY V2 have six addressing modes

5.3.2.1. 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 the source register field; RZ field is the destination register field; IMM3 field directly participates 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	

5.3.2.2. 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 lsli16, lsri16 and asri16

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	

5.3.2.3. Addressing Mode of Single Register 5-bit Immediate Operand

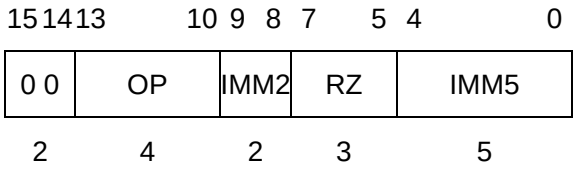
In the instructions that adopt the addressing mode of single register 5-bit immediate operand, RX field is the source register field or destination register; SOP field is the sub-operation code field. Instructions of this format include cmphsi16, cmplti16, cmpnei16, bclri16, bseti16 and btsti16.

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

5.3.2.4. 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.



5.3.2.5. Addressing Mode of 7-bit Immediate Operand

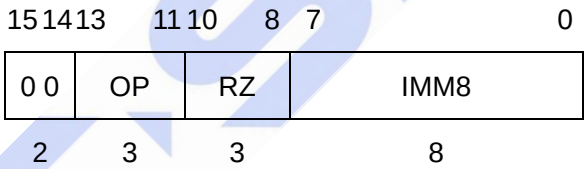
In the instructions that adopt the addressing mode of 7-bit immediate operand, IMM2 field and IMM5 field can be combined into 7-bit immediate operand to directly participate in data operation; SOP field is the sub-operation code field. Instructions of this format include push16, pop16, bpush16.h, bpush16.w,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

5.3.2.6. 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.

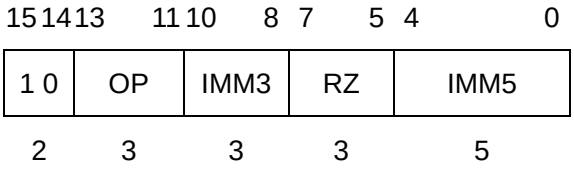
In the first format, RZ field is the destination register field; IMM8 field can also directly participate in data operation as 8-bit immediate operand; SOP field is the sub-operation code field. Instructions of this format include addi16(SP), subi16(SP) and movi16.

In the second format, RZ field is the source register field or 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 and subi16.



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).

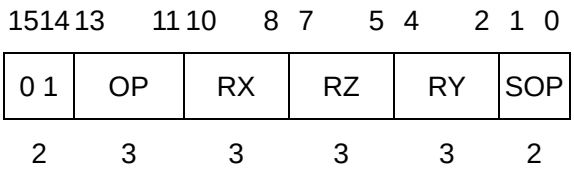


5.3.3. Addressing Mode of Register-type Instructions

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

5.3.3.1. Addressing Mode of Ternary Register

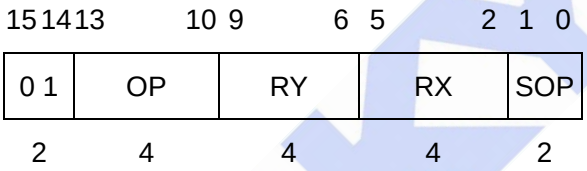
In the instructions that adopt the addressing mode of ternary register, the two register fields RX and RY are the first source register field and second source register field respectively; RZ field is the destination register field; SOP field is the sub-operation code field. Instructions of this format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



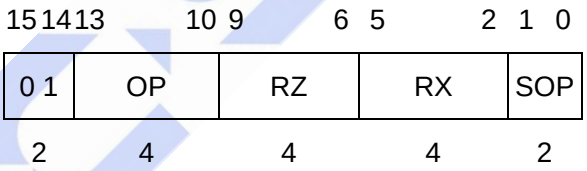
5.3.3.2. Addressing Mode of Two Register

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,tst16.



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.



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 and mult16.

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

5.3.3.3. 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	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	2

6. Exception Handling

Exception handling (including instruction exception and external interruption) is an important technology of processors. When exceptions happen this technology can be used to help processors deal with these events. Exceptions include hardware error, instruction error and user request etc. In this chapter, the category of exceptions, exception priority, exception vector table, return from exception and bus error recovery will be introduced.

6.1. Overview of Exception Handling

Exception handling is the process which processors change from normal flow of programs to exception handler to respond the occurrence of internal or external exceptions. Events trigger external exceptions including: interrupt request, read/write access error or hardware reset; events trigger internal exceptions include illegal instructions, misaligned error, privilege violation exceptions and instruction trace. Trap and bkpt cause internal exceptions even when being executed normally. Moreover, illegal instructions, load/store instruction misaligned, executing privilege instructions in user mode would result in exception too. Exception handler will jump to the entrance of exception handle service with the help of exception vector table.

The key to exception handling is that when exceptions happen, the current state of execution can be preserved and later resumed at the original location. Exceptions can be recognized at every stage of the pipeline and following instructions should not be allowed to change the state of CPU. Exceptions are handled at the boundary of instructions, namely CPU responds to interrupt after instruction retires, saves the address of next instruction before jumping to exception handler. In order not to affect the performance, same instructions should not be executed again after returning from the exception handler. CPU would save the address of instruction based upon whether it is finished when exception occurs. For example, if the exception is an external interrupt, the interrupted instruction would retire as usual, the address of the next instruction(PC+2/PC+4 decided based on 16/32 bit) would be restored in exception PC register(EPC) as the entrance when return from exception handler; if the exception is caused by access error, the instruction itself is not finished, it will retire but the state of CPU should not be changed(REG values remain the same), and its PC would be stored in exception PC register(EPC), CPU will execute the instruction again when return from exception handler.

Exception handling should be processed with following these steps:

Step 1, store PSR and PC into shadow registers (EPC and EPSR)

Step 2, set the S bit of PSR(disregard of current mode) to enter the supervisor mode;

Step 3, update the vector number in the VEC field with the current exception number, mark out the type of the exception and the condition of shared exception handler;

Step 4, clear the exception enable bit (EE) in PSR, forbid any new exceptions. If exceptions(except for interrupt) occur when EE equals to 0, processor would regard them as unrecoverable exceptions. When any unrecoverable exceptions happen, both EPSR and EPC would be updated;

Step 5, clear the interrupt enable(IE) in PSR, forbid responding to interrupt;

Note Step 2-4, happens simultaneously;

Step 6, processor generates the entrance address of exception handler based on the exception vector number and fetch the address of first instruction of exception handler. Multiply the exception vector number by 4 and sum up with exception vector base stored in vector base register(VBR), otherwise equals to 0 if VBR doesn't exist, to obtain the exception handler entrance address. Using this address to acquire the address of 1st instruction in exception handler from memory. For interrupt, the exception vector is determined by external interrupt controller;

Last step, processor executes the exception handler from its first instruction, starts handling exception.

All exception vectors are stored in supervisor mode address space and indexed by instructions, among which only reset vector base is fixed. Once the processor finishes initializing, if configured with VBR, the exception base address will be reloaded.

CSKY supports a 256 bytes vector table which includes 64 exception vectors(check Chart7-1). The first 30 vectors are used to identify internal vectors. 31st vector is reserved for software as a pointer points to system descriptor. The rest 32 vectors are reserved for external devices. Exception handler is controlled to respond to the interrupt requests by 8 bits interrupt vector and interrupt controller. When processor responds to the interrupt request it will lock the interrupt vector.

Chart 6-1 Exception vector table

Vector number	Vector offset(Hex)	Exception
0	000	Reset exception
1	004	Unaligned Memory Access exception
2	008	Access error exception
3	00C	Reserved
4	010	Illegal instruction exception

Vector number	Vector offset(Hex)	Exception
5	014	Instruction Privilege violation exception
6	018	Reserved
7	01C	Break-point exception
8	020	Unrecoverable exception
9-15	024-03C	Reserved
16－19	040－04C	Trap exception（TRAP #0-3）
20-30	050-078	Reserved
31	07C	Points to system descriptors
32－255	080－FC	Reserved for interrupt controller

6.2. Exception Types

In this section internal and external exceptions of CSKY CPU will be introduced. Exceptions needs to handle can be partitioned into follow categories,

- Reset exception;
- Misaligned memory access exception;
- Access error exception;
- Illegal instruction exception;
- Instruction privilege violation exception;
- Break-point;
- Unrecoverable exception;
- Vector interrupt;

6.2.1.Reset Exception(offset 0x0)

Reset exception is the exception with highest priority, it is used to initialize the system or recover it from major malfunctions. Reset exception would stop all activities in the processor, all of which are unrecoverable. Reset is also used to test the initialization of scan-chain, check the value of latch in clock control logic and initialize after power-on.

Reset exception sets PSR(S) high to ensure the processor works in supervisor mode. It will also clear PSR(IE) and PSR(EE) to stop the processor to respond to exceptions or interruptions. At the meantime, the VBR gets cleared and its default value is 0x00000000. CPU reads from the exception vector table with offset 0x0, to get the entrance of exception handler and loads it to PC. The exception handling would start from that

address.

6.2.2.Misaligned Memory Access Exception (offset 0x4)

When the processor tries to access to misaligned address, exception occurs. By setting PSR(MM), this exception can be masked, processor will not check alignment. When MM is set, if the processor supports unaligned access, it will access the memory with unaligned address otherwise it sets the lowest bit 0 to access the memory. EPC points to the instruction with unaligned address. In CSKY CPU unaligned access only happens when access data.

Under any circumstances, if split memory access instructions (including LDM, STM, PUSH, POP, NIE, NIR, IPUSH, IPOP etc.) have unaligned address, the processor needs to respond the unaligned memory access exception.

6.2.3.Access Error Exception (offset 0x8)

If external bus returns error signal (for example pad_biu_hresp[1:0]=1) , access error exception happens. When access to area protected by MPU, access error occurs too. EPC points to the instruction which sends out the request to the bus.

Any errors on bus would result in access error, the exception handler would be triggered.

6.2.4.Illegal Instruction Exception (offset 0x10)

If instructions are found out illegal or cannot be implemented while decoding, CPU would not execute these instructions, instead the exception handler would be triggered. EPC points to that illegal instruction.

6.2.5.Instruction Privilege Violation Exception (offset 0x14)

For safety concern, some instructions are privileged, they can only be executed in supervisor mode. Any attempt to execute them in user mode would result in privilege violation exception.

If a privilege violation happens, exception handler takes effect before execute that instruction. EPC points to that instruction.

6.2.6.Break-point Exception (offset 0x1C)

CSKY CPU responds to break-point exception when FDB bit of CSR is 0 and bkpt instruction is executed. EPC points to the bkpt instruction.

6.2.7.Unrecoverable Exception (offset 0x20)

When PSR (EE) is 0, all exceptions except for the reset exception will result in unrecoverable exception. Because at this time, EPC and EPSR might being overwritten by other exceptions.

Because software should eliminate the possibility of any exceptions when EE bit of PSR is 0, any unrecoverable exception always indicates system error. In exception handler, the type of exception that causes unrecoverable exception is undetermined.

6.2.8.Interrupt Exception

When external devices need to request service or send data, they can use interrupt signal and corresponding interrupt vector to trigger an interrupt exception.

Common interrupt is identified on the boundary of instructions. If IC bit of PSR is set, LDM, STM, PUSH, POP, IPUSH, IPOP and some other multi-cycle instructions can be interrupted before they finish to shorten the wait time. Multi-cycle instruction NIE cannot be interrupted and NIR can only respond to interrupt at the end of its execution, regardless of IC bit in PSR.

6.2.8.1. Exception Vector (INT)

If IE bit in PSR is cleared, the interrupt signal is masked, processor would not respond to interrupt. Common interrupt uses EPSR and EPC as its shadow registers, it can also be masked by EE bit in PSR. When interrupt is enabled, processor uses specified signal to provide interrupt exception number which can be any one from 32-255(0-31 are not allowed).

6.2.8.2. Process of handling interrupt

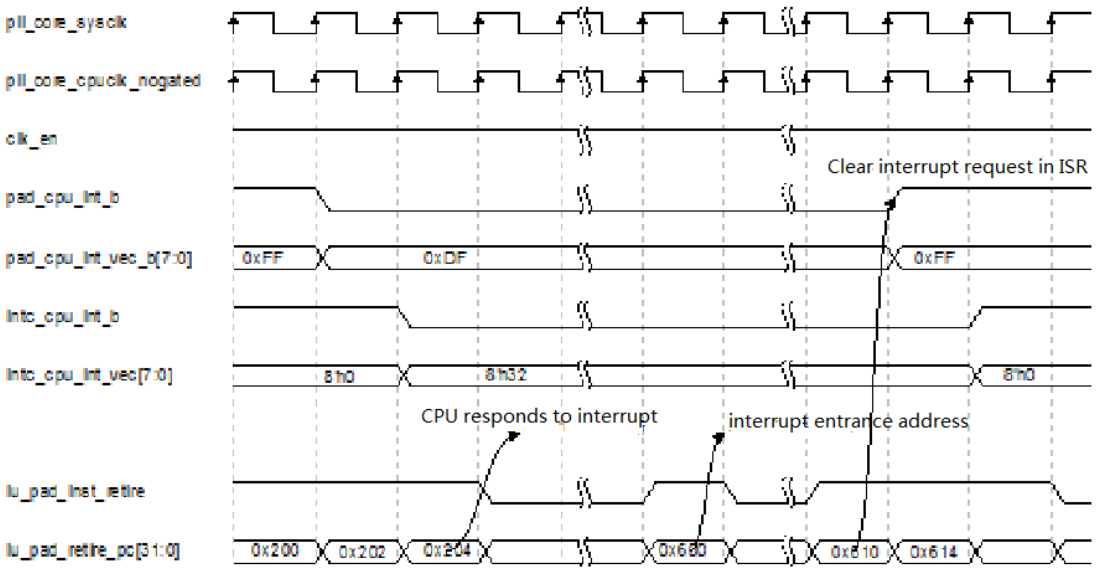


Chart 6-2 process of handling interrupt

In the above Chart, when interrupt vector number is ready, the interrupt signal is pulled down. After this signal is sampled by the posedge of both system clk and CPU clk, CPU would receive the interrupt and the interrupt vector based on the interrupt code to enter the interrupt handler. In interrupt handler, external interrupt source should be cleared by software, namely pull up the interrupt enable signal. This signal is also required to be sampled by two clks.

For more detailed introduction of interrupt mechanism and signal interface, please check related guide.

6.2.9.Trap Exception (offset 0x40-0x4C)

Some instructions can be used to generate trap exceptions. Trap #n can enforce the generation of exception, which can be used while debugging. In exception handler, EPC points to trap instruction.

6.3. Exception priority

As shown in Chart 7-3, based on the property and the handling order, CSKY prioritizes exceptions into 5 classes. In Chart 7-3, 1 represent highest priority while 5 is the lowest. It is worth noting that, in class 4,5, several exceptions share the same priority, because they are mutually exclusive.

In CSKY CPU, multiple exceptions can occur simultaneously. Reset exception has the highest priority for its special property.

All other exceptions are handled in an order based on the priority listed in Chart 7-3.

If EE bit of PSR is cleared, any exception would result in an unrecoverable exception.

When several exceptions happen at the same time, the one with highest priority would be dealt with first. When processor returns from exception and the particular instruction is re-executed, other exceptions would reappear.

Chart 6-3 Exception priority

Priority	Exception types	Exception properties
1	Reset exception	Processor stops all programs and reset the system
2	Unaligned Memory Access exception	After the retirement of the instruction, processor saves the current state and starts handling exception.
3	Interrupt	If IC=0, interrupt would be responded after retirement, otherwise before instruction finishes.
4	Unrecoverable exception; Access error exception;	After the retirement of the instruction, processor saves the current state and starts handling exception.
5	Illegal exception; Instruction Privilege violation exception; Trap exception; Break-point exception;	After the retirement of the instruction, processor saves the current state and starts handling exception.

6.3.1.Debug Request while Handling Exceptions

If processor receives a debug request while handling exceptions, debug request would be responded first. Exception handling would be delayed until processor exits debug mode and the instruction causes the exception is re-executed.

6.4. Return from Exception

Processor can return from exception by executing rte instruction. Rte instruction uses the information stored in shadow register EPSR and EPC to return from exception.

7. Work Mode Switching

CSKY CPU has three work modes, normal mode, low power mode and debug mode. Low power mode can be further partitioned into three modes: STOP mode, DOZE mode and WAIT mode. This chapter will explain all these modes and how they switch.

7.1. CSKY CPU Work Mode and Switching

As shown in chart 10-1, there are three work modes in CSKY CPU, namely normal mode, low power mode and debug mode. Which mode the CPU is in can be determined by checking had_pad_jdb_pm[1:0].

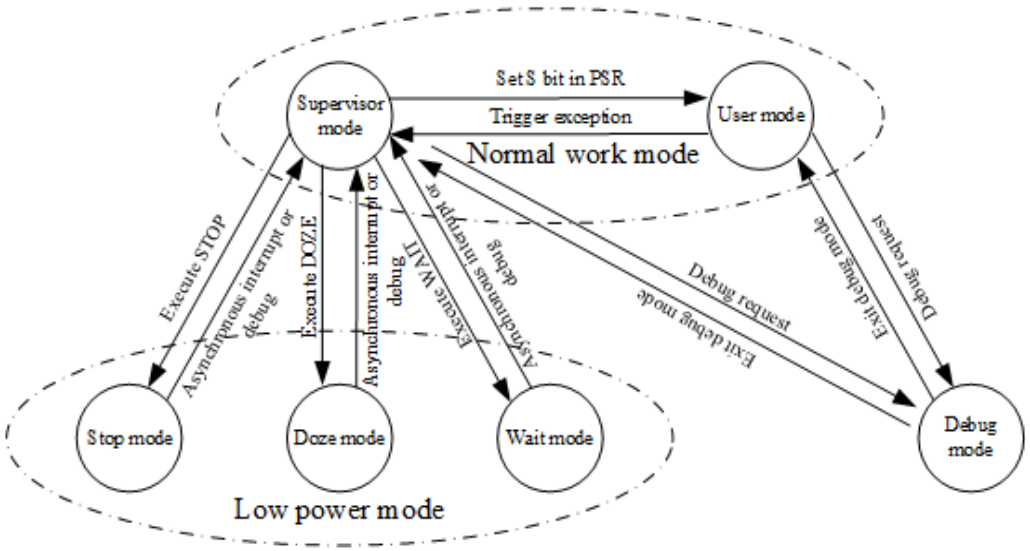


Chart 7-1 CPU Work Modes

7.1.1. Normal Mode

The normal mode can be partitioned into two types: supervisor mode and user mode, which mode the CPU is in can be determined by checking the S bit in PSR. When S bit is 1, CPU is in supervisor mode; when S is 0, CPU is in user mode. If the CPU is in supervisor mode, it can switch to user mode by clear S bit; if CPU is in user mode, it needs to trigger exceptions to enter supervisor mode.

7.1.2.Low Power Mode

After executing low power instructions (STOP, DOZE, WAIT), CPU enters low power mode. In low power mod, the CPU clock is stopped, only asynchronous interrupt request (pad_sysio_intraw_b) or debug request can help exit low power mode. Which low power

mode the CPU is in can be determined by checking had_pad_jdb_pm[1:0].

7.1.2.1. DOZE Mode

After executing DOZE instruction, the CPU enters DOZE low power mode. In this mode the CPU clock is stopped, whether external clocks are stopped depends on practical need.

7.1.2.2. STOP Mode

After executing STOP instruction, the CPU enters STOP low power mode. In this mode the CPU clock is stopped, so do almost all external clocks.

7.1.2.3. WAIT Mode

After executing WAIT instruction, the CPU enters WAIT low power mode. In this mode the CPU clock is stopped, but almost all external clocks remain working and can generate new interrupt.

7.1.3.Debug Mode

7.1.3.1. Entering Debug Mode

After receiving debug request, CPU enters debug mode, the request may come from:

- When ADR bit in HCR is set, processor enters debug mode directly.
- When DR bit in HCR is set, processor enters debug mode after finishing current instruction.
- When FDB bit in CSR is set, processor enters debug mode only when execute bkpt.
- When TME bit in HCR is set, processor enters debug mode after trace-counter decrease to 0.
- When CPU is in memory break-point debug mode, if BKPTA or BKPTB is triggered (MBCA or MBCB equals 0), or any of BKPTC-BKPTI is triggered, and the current instruction meets the requirement of break-point, the processor enters debug mode.

When processor is in low power mode, it can exit low power mode and enter debug mode by setting ADR in HCR or DR.

7.1.3.2. Exiting Debug mode

If GO,EX bit in HACR of CPU is set, and at the meantime R/W is 0(write), RS selects WBBR,PSR,PC,IR,CSR or Bypass register, the CPU exits debug mode when executes any instruction.

Note: In debug mode PC, CSR, PSR might get changed, so when exist debug mode, the value of these registers must be the ones stored when enter debug mode.

Appendix A: Example of Setting MPU

```
/*******************************************************************************
* Function: An example of set MPU.
*                 Enable/disable memory space region.
* Memory space: 0x28000000 ~ 0x29000000(16MBytes).
*
* Id  |          Memory Space          | Write   | Read   | Executable  | Security  |
* 0   | 0x00000000 ~ 0xFFFFFFFF        | Yes     | Yes    | NO          | NO        |
* 1   | 0x28000000 ~ 0x29000000        | Yes     | Yes    | Yes         | Yes       |
* 2   | 0x28000000 ~ 0x28100000        | No      | Yes    | Yes         | NO        |
* 3   | 0x28F00000 ~ 0x29000000        | Yes     | Yes    | No          | Yes       |
*
* Copyright (c) 2007 C-SKY Microsystems Co.,Ltd. All rights reserved.
********************************************************************************/

    /* Enable/disable every memory area. */
    /* Set the access authorization.       */
    /* Set the executable or not.              */
    /* Set the security or not.            */
    movih     r10,0xa00
    ori       r10,r10,0xef06
    mtcr      r10,cr<19,0>

    /* The first area (0x00000000 ~ 0xFFFFFFFF) */
    movi      r10,0
    mtcr      r10,cr<21,0>
    movi      r10,0x3f                /* 4G space, Base address: 0x00000000 */
    mtcr      r10,cr<20,0>

    /* The second area (0x28000000 ~ 0x29000000) */
    movi      r10,1
    mtcr      r10,cr<21,0>

    movih     r10,0x2800
ori       r10,r10,0x2f      /* 16M Space, Base address: 0x28000000 */
    mtcr      r10,cr<20,0>

    /* The third area (0x28000000 ~ 0x28100000) */
    movi      r10,2
    mtcr      r10,cr21
```

```
        movih     r10,0x2800
    ori       r10,r10,0x2f      /* 1M Space, Base address: 0x28000000 */
        mtcr      r10,cr<20,0>

        /* The fourth area (0x28F00000 ~ 0x29000000) */
        movi      r10,3
        mtcr      r10,cr21

        movih     r10,0x2800
    ori       r10,r10,0x27      /* 1M Space, Base address: 0x28F00000 */
        mtcr      r10,cr<20,0>

        /* Enable MPU */
        mfcr      r7, cr<18,0>
        bseti     r7, 0
        bclri     r7, 1
        mtcr      r7, cr<18,0>
```

Appendix B: Term list of basic instructions

Following is the detailed description of every CSKY V2 instruction, listed alphabetically.

Every instruction is ended with number ‘32’ or ‘16’, represent its instruction width. For example, ’addc32’ means it is a 32 bit unsigned add with carry, ‘addc16’ means it is a 16 bit unsigned add with carry.

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

If the width is omitted, the system would pick the optimized one during compiling. Moreover any instruction marked with # is a pseudo instruction.

ADDC – Add with carry unsigned

Unified instruction

Grammar	Operation	Compiling result

addc rz, rx	RZ ← RZ + RX + C,
	C ← 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, rz, rx;

addc rz, rx, ry	RZ ← RX + RY + C,
	C ← 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 ← carry

Exception: None

16-bit instruction

Operation: RZ ← RZ + RX + C, C ← 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 ← 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 ← RX + RY + C, C ← 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 bit: C ← carry

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 ← 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 ← 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 ← 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 ← RZ + 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 flag bit: No influence

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 ← RX + 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 flag bit: No influence

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

Operation: RZ ← RX + 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 bit: No influence

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

Exception: None

Instruction format:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	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.

000000000000 – +0x1

000000000001 – +0x2

……

111111111111 – +0x1000

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

Unified instruction

Grammar	Operation	Compiling result
addi rz, sp, imm	RZ ← SP + zero_extend(IMM)	Only 16-bit instructions exist. addi rz, sp, imm
addi sp, sp, imm	SP ← 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 ← 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 ← SP + zero_extend(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 {IMM2, IMM5} << 2.

Influence on flag bit: No influence

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

ADDU – Add unsigned

Unified instruction

Grammar	Operation	Compiling result

addu rz, rx	RZ ← 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, rz, rx;

addu rz, rx, ry	RZ ← 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;
		elsif (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 ← RZ + RX

Grammar: addu16 rz, rx

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

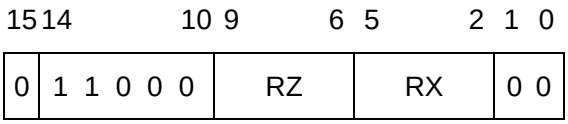
Influence on No influence

flag bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



16-bit instruction ----2

Operation: RZ ← RX + RY

Grammar: addu16 rz, rx, ry

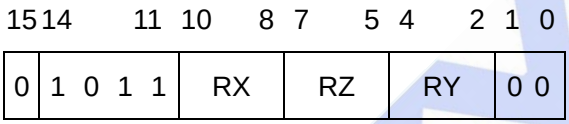
Description: Add the values of RX and RY, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r7.

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← RX + RY

Grammar: addu32 rz, rx, ry

Description: Add the values of RX and RY, 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 0 0		0 0 0 0 1		RZ		

AND – Bitwise AND

Unified instruction

Grammar	Operation	Compiling result

and rz, rx	RZ ← RZ 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 ← RX 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 ← RZ 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 ← 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 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 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 ← RX 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 0x0-0xFFF.

Exception: None

32-bit instruction

Operation: RZ ← RX and zero_extend(IMM12)

Grammar: andi32 rz, rx, imm12

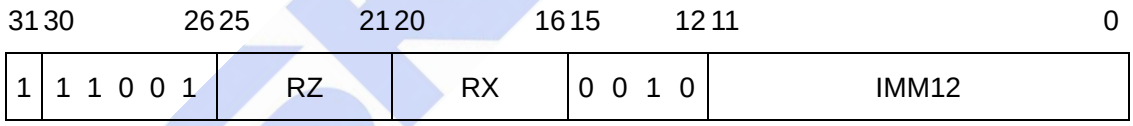
Description: Zero-extend the 12-bit immediate operand to 32 bits, perform a bitwise-AND with RX value, and save the result in RZ.

Influence on flag bit: No influence

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

Exception: None

Instruction format:



ANDN – Bitwise AND-NOT

Unified instruction

Grammar	Operation	Compiling result

andn rz, rx	RZ ← RZ 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 ← RX 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 flag bit: No influence

Exception: None

16-bit instruction

Operation: RZ ← RZ 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

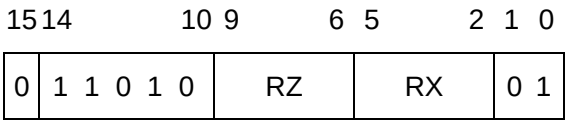
Influence on No influence

flag bit:

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← RX and (!RY)

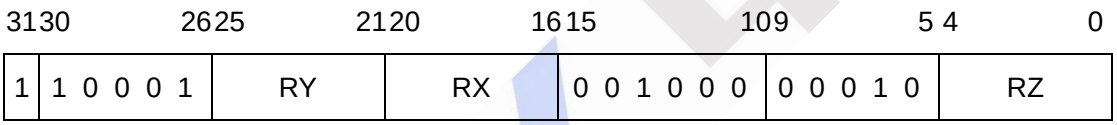
Grammar: andn32 rz, rx, ry

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

Influence on flag bit: No influence

Exception: None

Instruction format:



ANDNI – Bitwise AND-NOT immediate

Unified instruction

Grammar	Operation	Compiling result
andni rz, rx, imm16	RZ ← RX and !(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 0x0-0xFFF.

Exception: None

32-bit instruction

Operation: RZ ← RX and !(zero_extend(IMM12))

Grammar: andni32 rz, rx, imm12

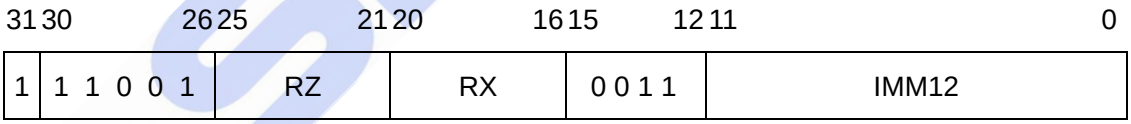
Description: Zero-extend the 12-bit immediate operand to 32 bits, perform a bitwise NOT, perform a bitwise-AND with RX value, and save the result in RZ.

Influence on flag bit: No influence

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

Exception: None

Instruction format:



ASR – Arithmetic shift right

Unified instruction

Grammar	Operation	Compiling result

asr rz, rx	RZ ← 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, rz, rx;

asr rz, rx, ry	RZ ← 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 bit: No influence

Exception: None

16-bit instruction

Operation: RZ ← RZ >>> RX[5:0]

Grammar: asr16 rz, rx

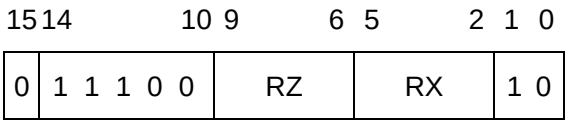
Description: Perform an arithmetic right shift on RZ value (the original value shifts right and the copy of original sign bit will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than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

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← 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 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 1 0 0 0 0		0 0 1 0 0		RZ		

ASRC – Arithmetic shift right immediate to C

Unified instruction

Grammar	Operation	Compiling result
asrc rz, rx, oimm5	RZ ← RX >>> OIMM5, C ← 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 bit: C ← RX[OIMM5 - 1]

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

Exception: None

32-bit instruction

Operation: RZ ← RX >>> OIMM5, C ← 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 bit: C ← 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 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 ← RX >>> 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 ← RX >>> 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 1 0	RX	RZ	IMM5

32-bit instruction

Operation: RZ ← RX >>> 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 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 0 1 0 0	RZ

BCLRI – Bit clear immediate

Unified instruction

Grammar	Operation	Compiling result

bclri rz, imm5	RZ ← 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 ← 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 ← 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 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 0	IMM5

32-bit instruction

Operation: RZ ← 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 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 0 1 0 1 0	0 0 0 0 1	RZ

BF – C=0 branch instruction

Unified instruction

Grammar	Operation	Compiling result

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

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: bf16 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 ← 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 ±1KB.

Influence on flag bit: No influence

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

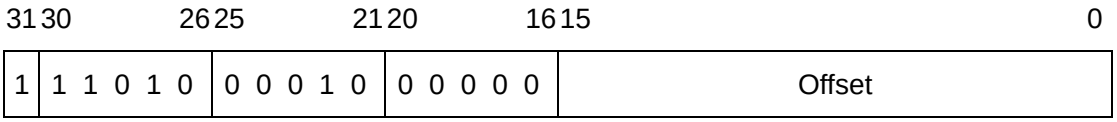
Description: If the condition flag bit C is equal to zero, the program will shift to label position before execution; otherwise the program will execute the next instruction, i.e. PC ← PC + 4.

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 ±64KB.

Influence on flag bit: No influence

Exception: None

Instruction format:



BGENI – Bit generation immediate#

Unified instruction

Grammar	Operation	Compiling result
bgeni rz, imm5	RZ ← (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 ← (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:

31	30	26	25	21	20	16	15	0
1	1 1 0 1 0		1 0 0 0 0		RZ		$(2)^{IMM5}$	

If IMM5 is greater than 16:

31	30	26	25	21	20	16	15	0
1	1 1 0 1 0		1 0 0 0 1		RZ		$(2)^{IMM5}$	

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

Exception: Breakpoint exception

16-bit instruction

Operation: Trigger a breakpoint exception or enter debugging mode

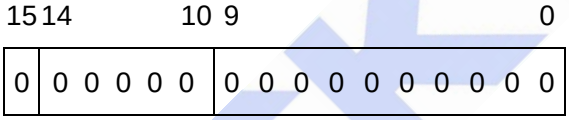
Grammar: bkpt16

Description: Breakpoint instruction

Influence on flag bit: No influence

Exception: Breakpoint exception

Instruction format:



BMASKI – Bit mask generation immediate

Unified instruction

Grammar	Operation	Compiling result
bmaski rz, oimm5	RZ ← $(2)^{OIMM5}$ - 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 ← $(2)^{OIMM5}$ - 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

BR – Unconditional jump

Unified instruction

Grammar	Operation	Compiling result

br label	PC ← PC + sign_extend(offset << 1)	Compiled into corresponding 16-bit or 32-bit instructions according to the range of jump.
		if(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 ← PC + sign_extend(offset << 1)

Grammar: br16 label

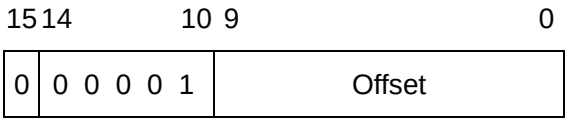
Description: The program unconditionally jumps to label for execution.
Label is gained by adding the current PC to the value of sign-extending the 10-bit relative offset shifting left by 1 bit to 32 bits. The jump range of BR16 instruction is the address space of ±1KB.

Influence on flag bit: No influence

Exception: None

Instruction

format:



32-bit instruction

Operation: PC ← PC + sign_extend(offset << 1)

Grammar: br32 label

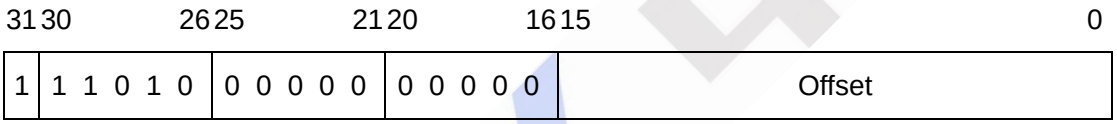
Description: The program unconditionally jumps to label for execution.

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 ±64KB.

Influence on flag bit: No influence

Exception: None

Instruction format:



BSETI – Bit set immediate

Unified instruction

Grammar	Operation	Compiling result

bseti rz, imm5	RZ ← RZ[IMM5] 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 ← RX[IMM5] 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 bit: No influence

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

Exception: None

16-bit instruction

Operation: RZ ← RZ[IMM5] set

Grammar: bseti16 rz, imm5

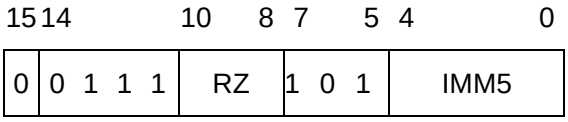
Description: Set the bit indicated by the value of IMM5 field as 1 in RZ value, keep other bits unchanged, and save the result in RZ.

Influence on flag bit: No influence

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

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← RX[IMM5] set

Grammar: bseti32 rz, rx, imm5

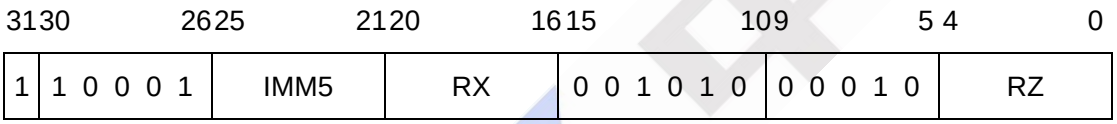
Description: Set the bit indicated by the value of IMM5 field as 1 in RX value, keep other bits unchanged, and save the result in RZ.

Influence on flag bit: No influence

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

Exception: None

Instruction format:



BSR – Jump to subprogram

Unified instruction

Grammar	Operation	Compiling result
bsr label	Link and jump to the subprogram: R15 ← next PC PC ← PC + sign_extend(offset << 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 bit: No influence

Exception: None

32-bit instruction

Operation: Link and jump to the subprogram:

R15 ← PC+4

PC ← PC + sign_extend(offset << 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 ±64KB.

Influence on flag bit: No influence

Exception: None

Instruction format:

31	30 26	25 0
1	1 1 0 0 0	Offset

BT – C=1 branch instruction

Unified instruction

Grammar	Operation	Compiling result

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

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 ← PC + 2.

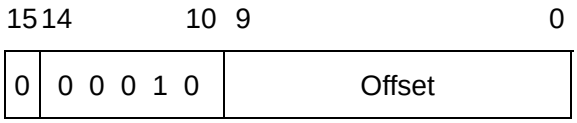
Label is gained by adding the current PC to the value of

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 ±1KB.

Influence on flag bit: No influence

Exception: None

Instruction format:



32-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 + 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. PC ← 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 ±64KB.

Influence on flag bit: No influence

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 ← 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 bit: C ← RX[IMM5]

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

Exception: None

16-bit instruction

Operation: C ← 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 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	RX	1 1 0	IMM5

32-bit instruction

Operation: C ← 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 bit: C ← RX[IMM5]

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

CMPHS – Compare unsigned when greater or equal

Unified instruction

Grammar	Operation	Compiling result

cmphs rx, ry	Make an unsigned comparison between RX and RY.
	If RX >= RY, then
	C ← 1;
	else
	C ← 0;
		cmphs16 rx, ry;

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 >= 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 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	0 0

CMPHSI – Compare immediate unsigned when greater or equal

Unified instruction

Grammar	Operation	Compiling result

cmphsi rx, oimm16	Make an unsigned comparison between RX and immediate operand.
	If rx>= zero_extend(OIMM16),
	C ← 1;
	else
	C ← 0;
		Compiled into corresponding 16-bit or 32-bit instructions according to the range of immediate operand and register.
		if (oimm16<33) and (x<8),then
		cmphsi16 rx, oimm5;
		else
		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 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 ← 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 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 >= zero_extend(OIMM16), then

C ← 1;

else

C ← 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	Make a signed comparison between RX and RY.
	If RX < RY, then
	C ← 1;
	else
	C ← 0;
		cmplt16 rx, ry;

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 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	0 1

CMPLTI – Compare immediate signed when smaller

Unified instruction

Grammar	Operation	Compiling result

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

Grammar: cmplti16 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 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 1	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	Make a comparison between RX and RY.
	If RX != RY, then
	C ← 1;
	else
	C ← 0;
		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

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

CMPNEI – Compare unequal immediate

Unified instruction

Grammar	Operation	Compiling result

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

16-bit instruction

Operation: Make a comparison between RX and immediate operand.
If RX != zero_extend(IMM5), then
C ← 1;
else
C ← 0;

Grammar: cmpnei16 rx, imm5

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		

DECF – C=0 SUBTRACT IMMEDIATE

Unified instruction

Grammar	Operation	Compiling result

decf rz, rx, imm5	if C==0, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← 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 ← RX - zero_extend(IMM5);
else
RZ ← 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	

DECT – C=1 subtract immediate

Unified instruction

Grammar	Operation	Compiling result

dect rz, rx, imm5	if C==1, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← 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 ← RX - zero_extend(IMM5);
else
RZ ← 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	

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

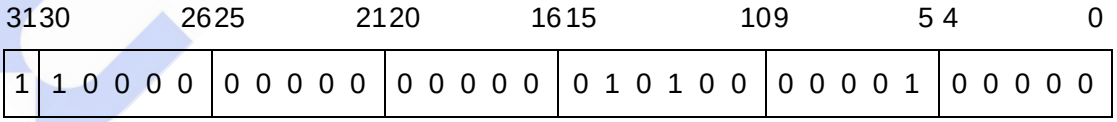
Attribute: Privileged instruction

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:



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 bit: No influence

Exception: None

32-bit instruction

Operation: RZ ← find_first_0(RX);

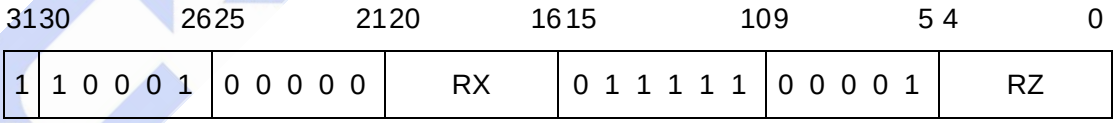
Grammar: ff0.32 rz, rx

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 bit: No influence

Exception: None

Instruction format:



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 bit: No influence

Exception: None

32-bit instruction

Operation: RZ ← find_first_1(RX);

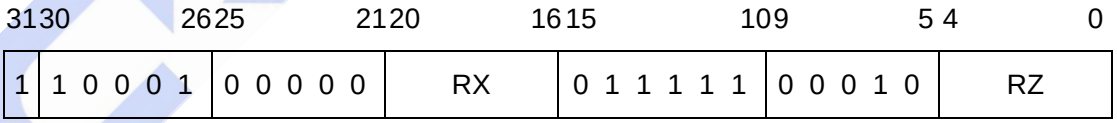
Grammar: ff1.32 rz, rx

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 bit: No influence

Exception: None

Instruction format:



INCF – C=0 add immediate

Unified instruction

Grammar	Operation	Compiling result

incf rz, rx, imm5	if C==0, then
	RZ ← RX + zero_extend(IMM5);
	else
	RZ ← 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 ← RX + zero_extend(IMM5);
else
RZ ← 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 ← RX + zero_extend(IMM5);
	else
	RZ ← 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 bit: No influence

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

Exception: None

32-bit instruction

Operation:
if C==1, then
RZ ← RX + zero_extend(IMM5);
else
RZ ← 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 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 1 0		IMM5	

IPUSH – Interrupt pop

Unified instruction

Grammar	Operation	Compiling result
ipush	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] ←{R13,R12,R3~R0}; SP←SP-24;	Only 16-bit instructions exist. ipush;

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] ←{R13,R12,R3~R0};
SP←SP-24;

Grammar: ipush16

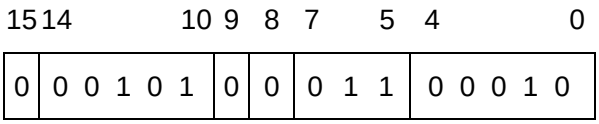
Description: Store the interrupted general-purpose register site {R0~R3, R12, R13} to the stack storage, and then update the stack pointer register to the top of stack storage. Adopt direct addressing mode of stack pointer register.

Influence on flag No influence

bit:

Exception: Access error exception and unaligned exception

Instruction format:



IPOP – Interrupt pop

Unified instruction

Grammar	Operation	Compiling result
ipop	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~R3,R12,R13} ← MEM[SP] ~ MEM[SP+20]; SP←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~R3,R12,R13} ← MEM[SP]~MEM[SP+20];
SP←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 No influence

bit:

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

IXH – Index half-word

Unified instruction

Grammar	Operation	Compiling result
ixh rz, rx, ry	RZ ← RX + (RY << 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 ← RX + (RY << 1)

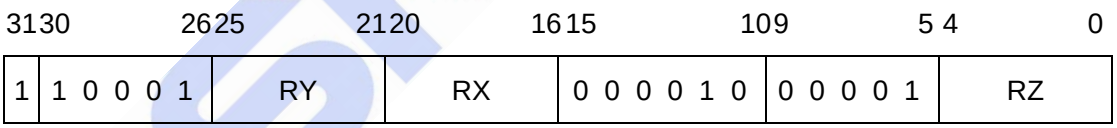
Grammar: ixh32 rz, rx, ry

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:



IXW – Index word

Unified instruction

Grammar	Operation	Compiling result
ixw rz, rx, ry	RZ ← RX + (RY << 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 ← RX + (RY << 2)

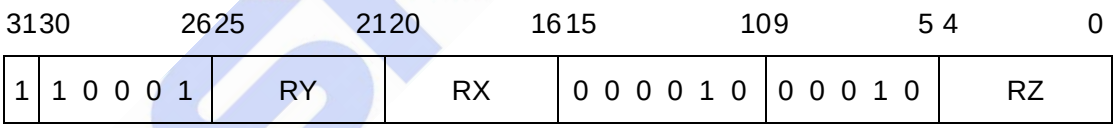
Grammar: ixw32 rz, rx, ry

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:



JMP – Register jump

Unified instruction

Grammar	Operation	Compiling result
jmp rx	Jump to the position appointed by register PC ← RX & 0xfffffffe	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

16-bit instruction

Operation: Jump to the position appointed by register
PC ← RX & 0xfffffffe

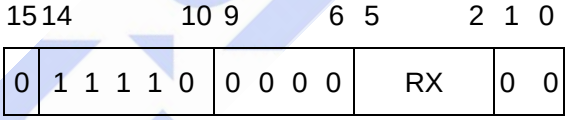
Grammar: jmp16 rx

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:



JSR – Register jump to subprogram

Unified instruction

Grammar	Operation	Compiling result
jsr rx	Link and jump to the subprogram position appointed by register R15 ← PC + 4, PC ← RX & 0xfffffffe	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 bit: No influence

Exception: None

16-bit instruction

Operation: Link and jump to the subprogram position appointed by register R15 ← PC + 4, PC ← RX & 0xfffffffe

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 bit: No influence

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

LD.B – Load unsigned and extended byte

Unified instruction

Grammar	Operation	Compiling result

ld.b rz,(rx, disp)	RZ ← zero_extend(MEM[RX + zero_extend(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 ← zero_extend(MEM[RX + zero_extend(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 bit: No influence

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 ← zero_extend(MEM[RX + zero_extend(offset)])

Grammar: ld32.b rz, (rx, disp)

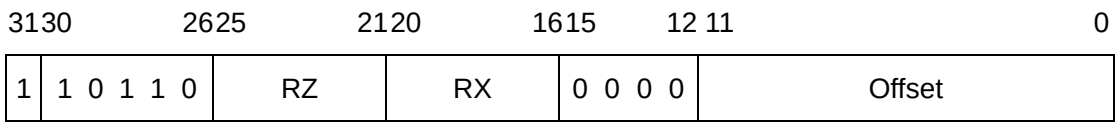
Description: Save the byte loaded from storage in register RZ after zero-extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset to 32 bits. The address space of LD32.B instruction is +4KB.

Attention: The offset DISP is the offset of binary operand.

Influence on flag bit: No influence

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

Instruction format:



LD.BS – Load signed and extended byte

Unified instruction

Grammar	Operation	Compiling result
ld.bs rz, (rx, disp)	RZ ← sign_extend(MEM[RX + zero_extend(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 ← sign_extend(MEM[RX + zero_extend(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 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 1 0 0		Offset	

LD.H – Load unsigned and extended half-word

Unified instruction

Grammar	Operation	Compiling result

ld.h rz, (rx, disp)	RZ ← zero_extend(MEM[RX + zero_extend(offset << 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 bit: No influence

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 ← zero_extend(MEM[RX + zero_extend(offset << 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.

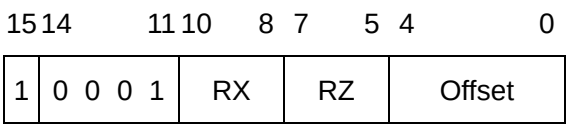
Attention: The offset DISP is gained after the offset of binary operand shifts left by 1 bit.

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:



32-bit instruction

Operation: Load half-word from storage to register, extend unsigned

RZ ← zero_extend(MEM[RX + zero_extend(offset << 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 ← sign_extend(MEM[RX + zero_extend(offset << 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 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 ← sign_extend(MEM[RX + zero_extend(offset << 1)])

Grammar: ld32.hs rz, (rx, disp)

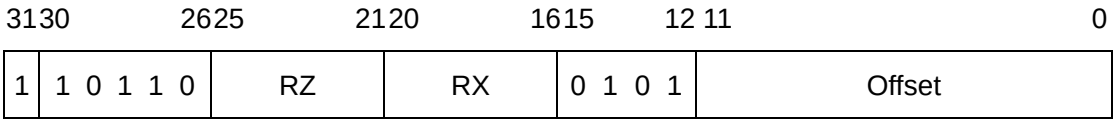
Description: Save half-word loaded from storage in register RZ after signed extension to 32 bits. Adopt the addressing mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset 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 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:



LD.W – Load word

Unified instruction

Grammar	Operation	Compiling result

ld.w rz, (rx, disp)	RZ ← MEM[RX + zero_extend(offset << 2)]	Compiled into 16-bit or 32-bit instructions according to the range of offset and register.
		if (x=sp) and (z<7) and (disp < 1024),
		ld16.w rz, (sp, disp);
		else if (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: No influence

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 ← MEM[RX + sign_extend(offset << 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 ← MEM[RX + zero_extend(offset << 2)]

Grammar: ld32.w rz, (rx, disp)

Description: Load word from storage to register RZ. Adopt the addressing

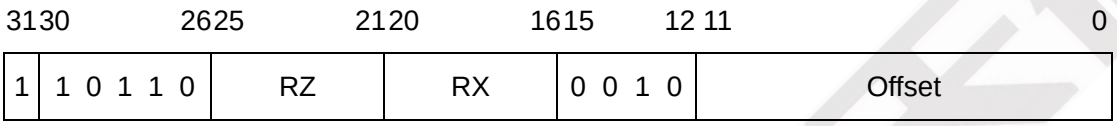
mode of register and immediate operand offset. The effective address of storage is gained by adding the base register RX to the value of unsigned extending the 12-bit relative offset shifting left by 2 bits to 32 bits. The address space of LD32.W instruction is +16KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag bit: No influence

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

Instruction format:



LDM – Load consecutive multiword

Unified instruction

Grammar	Operation	Compiling result

ldm ry-rz, (rx)	Load multiple consecutive words from storage to a group of consecutive register files
	dst ← Y; addr ← RX;
	for (n = 0; n <= (Z-Y); n++){
	Rdst ← MEM[addr];
	dst ← dst + 1;
	addr ← addr + 4;
	}
		Only 32-bit instructions exist.
		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 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)

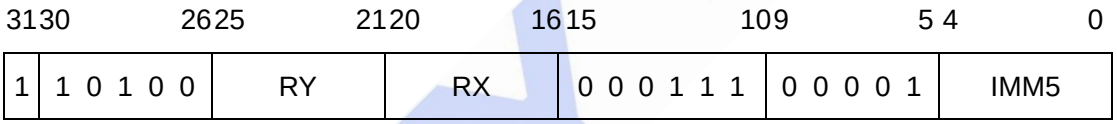
Description: Load multiple consecutive words from storage to a group of consecutive register files starting from register RY. In another word, load the first word in the address appointed by storage to register RY; load the second word to register RY+1, and the like; load the last word to register RZ. The effective address of storage is decided by the contents of base register RX.

Influence on 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

Instruction format:



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)	Load four consecutive words from storage to registers R4-R7
	dst ← 4; addr ← RX;
	for (n = 0; n <= 3; n++){
	Rdst ← MEM[addr];
	dst ← dst + 1;
	addr ← addr + 4;
	}
		Only 32-bit instructions exist.
		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 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 ← 4; addr ← RX;

for (n = 0; n <= 3; n++){

```
    Rdst ← MEM[addr];
    dst ← 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.

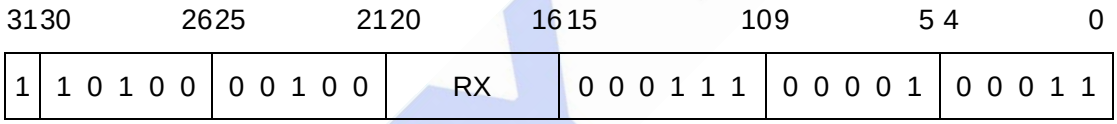
Attention: This instruction is the pseudo instruction of ldm32 r4-r7, (rx).

Influence on flag 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

Instruction format:



LRW – Memory read-in

Unified instruction

Grammar	Operation	Compiling result

lrw rz, label
lrw rz, imm32
	Load word from storage to register
	RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])
		Compiled into corresponding 16-bit or 32-bit instructions according to the range of load
		if(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 ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])

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.

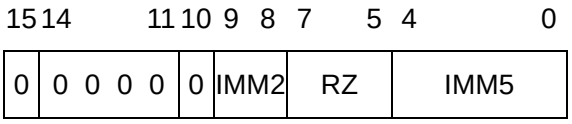
Attention: The relative offset is equal to the binary code {1, ~{IMM2, IMM5}}.

Influence on flag bit: No influence

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:



16-bit instruction----2

Operation: Load word from storage to register

RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])

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, {IMM2, IMM5}}.

Influence on No influence

flag 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 ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])

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 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 0 0	RZ	Offset

LSL – Logical shift left

Unified instruction

Grammar	Operation	Compiling result

lsl rz, rx	RZ ← 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, rz, rx;

lsl rz, rx, ry	RZ ← 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 bit: No influence

Exception: None

16-bit instruction

Operation: RZ ← RZ << RX[5:0]

Grammar: lsl16 rz, rx

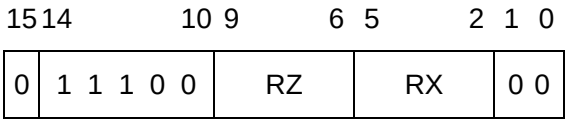
Description: Perform a logical left shift on RZ value (the original value shifts left and 0 will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 31, RZ will be cleared.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← RX << RY[5:0]

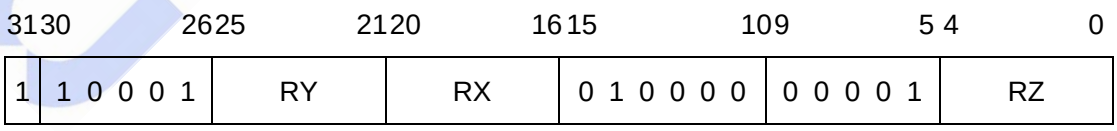
Grammar: lsl32 rz, rx, ry

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 bit: No influence

Exception: None

Instruction format:



LSLC – Logical shift left immediate to C

Unified instruction

Grammar	Operation	Compiling result
lslc rz, rx, oimm5	RZ ← RX << OIMM5, C ← 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 ← RX[32 – OIMM5]

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

Exception: None

32-bit instruction

Operation: RZ ← RX << OIMM5, C ← 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 ← 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 ← RX << 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 ← RX << 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 ← RX << 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 bit: No influence

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 ← 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, rz, rx;

lsr rz, rx, ry	RZ ← 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 bit: No influence

Exception: None

16-bit instruction

Operation: RZ ← RZ >> RX[5:0]

Grammar: lsr16 rz, rx

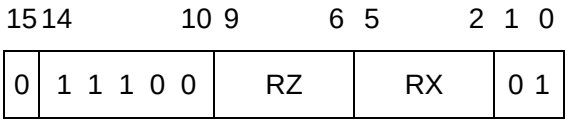
Description: Perform a logical right shift on RZ value (the original value shifts right and 0 will shift to the left side), and save the result in RZ; the range of right shift is decided by the value of six low bits of RX (RX[5:0]). If the value of RX[5:0] is greater than 31, RZ will be cleared.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← RX >> RY[5:0]

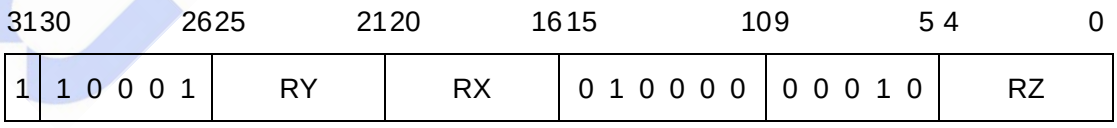
Grammar: lsr32 rz, rx, ry

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 bit: No influence

Exception: None

Instruction format:



LSRC – Logical shift right immediate to C

Unified instruction

Grammar	Operation	Compiling result
lsrc rz, rx, oimm5	RZ ← RX >> OIMM5, C ← 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 ← RX[OIMM5 - 1]

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

Exception: None

32-bit instruction

Operation: RZ ← RX >> OIMM5, C ← 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 ← 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 ← RX >> 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

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

Exception: None

16-bit instruction

Operation: RZ ← RX >> 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

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 1	RX	RZ	IMM5

32-bit instruction

Operation: RZ ← RX >> 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 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 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 ← 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 bit: No influence

Exception: Privilege violation exception

32-bit instruction

Operation: Transfer contents in control register to general-purpose register
RZ ← CR<X, sel>

Grammar: mfcr32 rz, cr<x, sel>

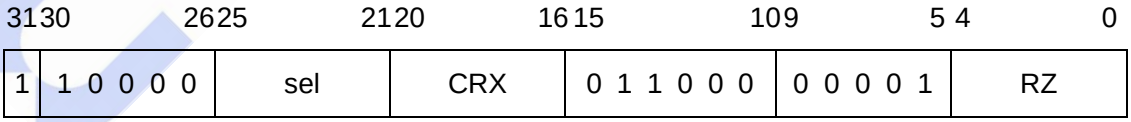
Attribute: Privileged instruction

Description: Transfer contents in control register CR<x, sel> to general-purpose register RZ.

Influence on flag bit: No influence

Exception: Privilege violation exception

Instruction format:



MOV – Move#

Unified instruction

Grammar	Operation	Compiling result
mov rz, rx	RZ ← RX	Always compiled into 16-bit instruction. mov16 rz, rx

Description: Copy the value in RX to destination register RZ.

Influence on flag bit: No influence

Exception: None

16-bit instruction

Operation: RZ ← RX

Grammar: mov16 rz, rx

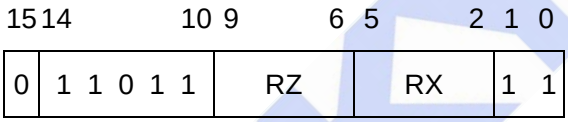
Description: Copy the value in RX to destination register RZ.

Attention: The register index range of this instruction is r0-r31.

Influence on flag bit: No influence

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← 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 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 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 ← zero_extend(IMM16);	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 ← zero_extend(IMM8);

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

15 14 11 10 8 7 0

0	0 1 1 0	RZ	IMM8

32-bit

instruction

Operation: RZ ← zero_extend(IMM16);

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 ← zero_extend(IMM16) << 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 ← zero_extend(IMM16) << 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
mtcr 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>

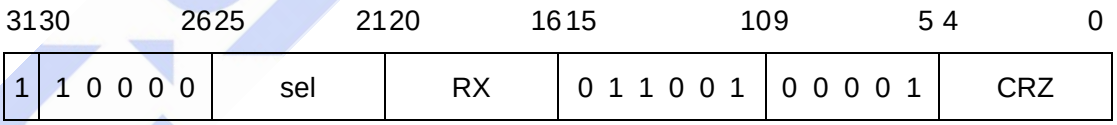
Attribute: Privileged instruction

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:



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 ← RX × 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, rz, rx;

mult rz, rx, ry	Multiply two numbers, and put the 32 low bits of the result in general-purpose register
	RZ ← 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
		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 ← RX × 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 bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:

32-bit instruction

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

Operation: Multiply two numbers, and put the 32 low bits of the result in general-purpose register

RZ ← RX × 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 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 0 0 1	0 0 0 0 1	RZ

MVC – C bit move

Unified instruction

Grammar	Operation	Compiling result
mvc rz	RZ ← 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 ← C

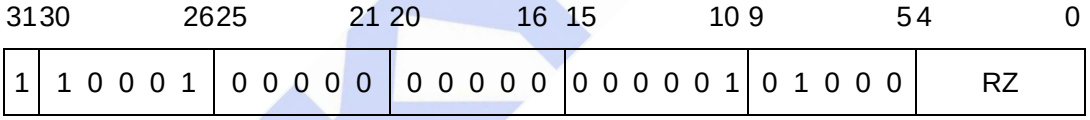
Grammar: mvc32 rz

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:



MVCV – C bit reverse move

Unified instruction

Grammar	Operation	Compiling result
mvcv rz	RZ ← (!C)	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

Exception: None

16-bit instruction

Operation: RZ ← (!C)

Grammar: mvcv16 rz

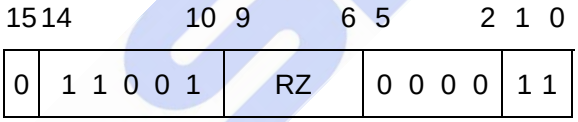
Description: Transfer the condition bit C to the lowest bit of RZ after negation, and clear other bits of RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



NIE – Interrupt nesting enable

Unified instruction

Grammar	Operation	Compiling result
nie	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	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

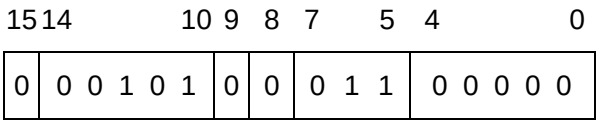
Attribute: Privileged instruction

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 bit: No influence

Exception: Unaligned access exception, unaligned access exception, and access error exception

Instruction format:



NIR – Interrupt nesting return

Unified instruction

Grammar	Operation	Compiling result
nir	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 EPSR←MEM[SP] EPC←MEM[SP+4]; SP←SP+8; PSR←EPSR; PC←EPC	Only 16-bit instructions exist. nir;

Attribute: Privileged instruction

Description: 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. 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: 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

EPSR←MEM[SP]

EPC←MEM[SP+4];

SP←SP+8;

PSR←EPSR;

PC←EPC

Grammar: nir16

Attribute: Privileged instruction

Description: 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. Adopt direct addressing mode of stack pointer register.

Influence on flag bit: No influence

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 1 1	0 0 0 0 1

NOR – Bitwise NOT-OR

Unified instruction

Grammar	Operation	Compiling result

nor rz, rx	RZ ← !(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 ← !(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 ← !(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 ← !(RX | 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 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 1 0 0 1	0 0 1 0 0	RZ

NOT – Bitwise NOT#

Unified instruction

Grammar	Operation	Compiling result

not rz	RZ ← !(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 ← !(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 ← !(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 ← !(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 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	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 ← 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 ← 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 ← 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 ← RX | 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 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 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 ← RX | 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 ← RX | zero_extend(IMM16)

Grammar: ori32 rz, rx, imm16

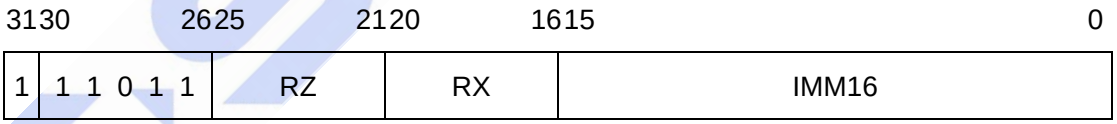
Description: Zero-extend the 16-bit immediate operand to 32 bits, perform a bitwise OR with RX value, and save the result in RZ.

Influence on flag bit: No influence

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

Exception: None

Instruction format:



POP – Pop

Unified instruction

Grammar	Operation	Compiling result

pop reglist	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 & 0xfffffffe;
		Compiled into corresponding 16-bit or 32-bit instructions according to the range of register
		if ({reglist}<16), then
		pop16 reglist;
		else
		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 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 & 0xfffffffe;

Grammar: pop16 reglist

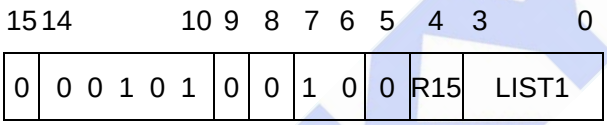
Description: Load multiple consecutive words from stack storage to a group of consecutive register files, update the stack pointer register, and realize the function of returning from the subprogram. In another word, the program jumps to the position appointed by link register R15 and the lowest bit of link register is ignored. Adopt the direct addressing mode of stack pointer register.

Influence on flag bit: No influence

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:



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

PUSH – Push

Unified instruction

Grammar	Operation	Compiling result

push reglist	Store words in register list to stack storage, and update stack register to the top of stack storage;
	src ← {reglist}; addr ← SP;
	foreach (reglist){
	addr ← addr - 4;
	MEM[addr] ← Rsrc;
	src ← next {reglist};
	}
	sp ← addr;
		Compiled into corresponding 16-bit or 32-bit instructions according to the range of register
		if ({reglist}<16), then
		push16 reglist;
		else
		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 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;
}
sp ← addr
```

Grammar: push16 reglist

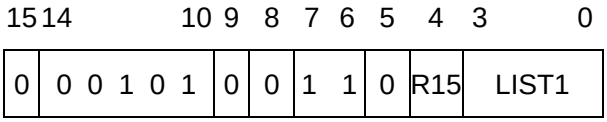
Description: Store words in register list to stack storage, and update stack register to the top of stack storage. Adopt the direct addressing mode of stack register.

Influence on flag bit: No influence

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:



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

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];	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

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

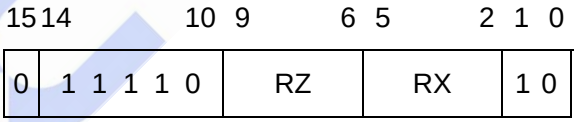
Description: Get the reverse order of RX value according to the byte, keep bit order inside the byte unchanged, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



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];	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 bit: No influence

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 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 1

ROTL – Rotate left

Unified instruction

Grammar	Operation	Compiling result

rotl rz, rx	RZ ← 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 ← 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 bit: No influence

Exception: None

16-bit instruction

Operation: RZ ← 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 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 0	RZ	RX	1 1

32-bit instruction

Operation: RZ ← 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 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 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 ← RX <<<< 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 ← RX <<<< IMM5

Grammar: rotli32 rz, rx, imm5

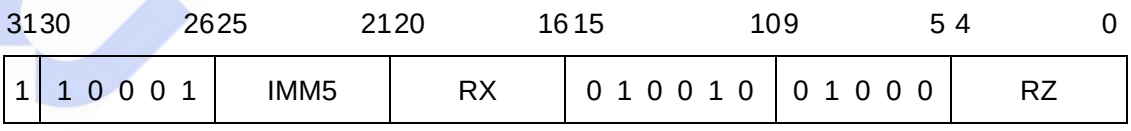
Description: Perform a ring left shift on RX value (the original value shifts left and the bit shifting out from left side will shift to the right side), and save the result in RZ; the range of left shift is decided by the value of 5-bit immediate operand (IMM5). If the value of IMM5 is equal to zero,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



RSUB – Reverse subtract#

Unified instruction

Grammar	Operation	Compiling result
rsub rz, rx, ry	RZ ← 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 ← 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 0 0	0 0 1 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 ← R15 & 0xfffffffe	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 bit: No influence

Exception: None

16-bit instruction

Operation: The program jumps to the position appointed by link register

PC ← R15 & 0xfffffffe

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 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	1 1 1 1	0 0

RTE – Return from exception/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

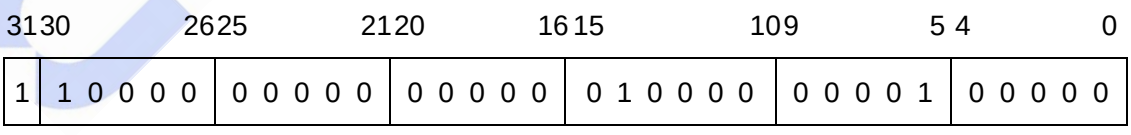
Attribute: Privileged instruction

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:



SEXTB – Extract byte and extend signed#

Unified instruction

Grammar	Operation	Compiling result
sextb rz, rx	RZ ← sign_extend(RX[7:0]);	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

Exception: None

16-bit instruction

Operation: RZ ← 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

SEXTH – Extract half-word and extend signed#

Unified

instruction

Grammar	Operation	Compiling result
sexth rz, rx	RZ ← sign_extend(RX[15:0]);	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

Exception: None

16-bit instruction

Operation: RZ ← sign_extend(RX[15:0]);

Grammar: sexth16 rz, rx

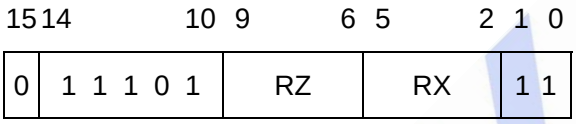
Description: Sign-extend low half-word of RX (RX[15:0]) to 32 bits, and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format



ST.B – Store byte

Unified instruction

Grammar	Operation	Compiling result

st.b rz, (rx, disp)	Store the lowest byte in register to storage
	MEM[RX + zero_extend(offset)] ← RZ[7:0]
		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
		st16.b rz, (rx, disp);
		else
		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 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

MEM[RX + zero_extend(offset)] ← 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 bit: No influence

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

MEM[RX + zero_extend(offset)] ← 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 bit: No influence

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.H – Store half-word

Unified instruction

Grammar	Operation	Compiling result

st.h rz, (rx, disp)	Store the lowest byte in register to storage
	MEM[RX + zero_extend(offset<< 1)] ← RZ[15:0]
		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
		st16.h rz, (rx, disp);
		else
		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 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 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 1 0 1	RX	RZ	IMM5

32-bit instruction

Operation: Store low half-word in register to storage

MEM[RX + zero_extend(offset << 1)] ← 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 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 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 + zero_extend(offset<< 2)] ← RZ[31:0]
		Compiled into 16-bit or 32-bit instructions according to the range of offset and register.
		if (x=sp) and (z<7) and (disp < 1024),
		st16.w rz, (sp, disp);
		else if (disp<128) and (x<7) and (z<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 + zero_extend(offset << 2)] ← 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 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:

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

MEM[RX + zero_extend(offset << 2)] ← 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

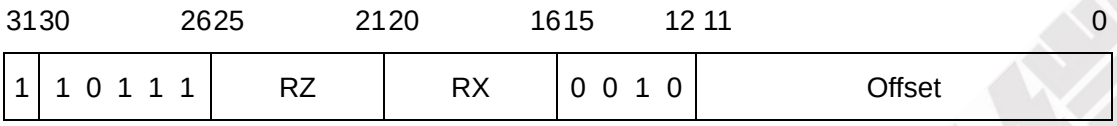
by two bits to 32 bits. The address space of ST32.W instruction is +16KB.

Attention: The offset DISP is gained after the offset of binary operand shifts left by 2 bits.

Influence on flag 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)	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 <=(Z-Y); n++){
	MEM[addr] ← Rsrc;
	src ← src + 1;
	addr ← addr + 4;
	}
		Only 32-bit instructions exist.
		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 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 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

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 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

STQ – Store consecutive quad word#

Unified instruction

Grammar	Operation	Compiling result
stq r4-r7, (rx)	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; }	Only 32-bit instructions exist. stq32 r4-r7, (rx);

Description: Store words in register file [R4,R7] (including boundary) to a group of consecutive storage addresses successively. In another 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: 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 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 another 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: 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 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

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	

SUBC – Subtract with borrow unsigned

Unified instruction

Grammar	Operation	Compiling result

subc rz, rx	RZ ← RZ - RX – (!C), C ← 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, rz, rx;

subc rz, rx, ry	RZ ← RX - RY - (!C), C ← 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 bit: C ← borrow

Exception: None

16-bit instruction

Operation: RZ ← RZ - RX – (!C), C ← 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 bit: C ← borrow

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 ← RX - RY – (!C), C ← 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 bit: C ← borrow

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 ← RZ - 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 ← RX - 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 ← RZ - 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 bit: No influence

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 ← RX - 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 bit: No influence

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 ← RX - 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 bit: No influence

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 ← SP- 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 ← SP - 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.

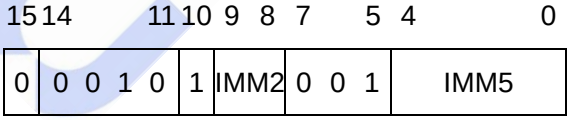
Attention: The immediate operand (IMM) is equal to the binary operand {IMM2, IMM5} << 2.

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) << 2.

Exception: None

Instruction format:



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 ← 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, rz, rx;

subu rz, rx, ry	RZ ← 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 bit: No influence

Exception: None

16-bit instruction----1

Operation: RZ ← RZ - RX

Grammar: subu16 rz, rx
sub16 rz, rx

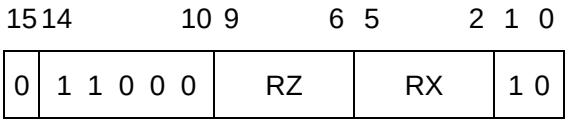
Description: Subtract RX value from RZ value and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



16-bit instruction----2

Operation: RZ ← RX- RY

Grammar: subu16 rz, rx, ry

sub16 rz, rx, ry

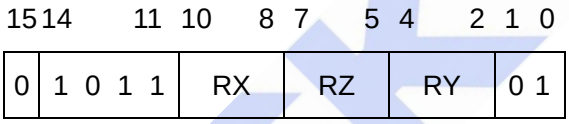
Description: Subtract RY value from RX value and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r7.

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← RX - RY

Grammar: subu32 rz, rx, ry

Description: Subtract RY value from 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 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 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 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;
		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

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

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;
		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

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

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

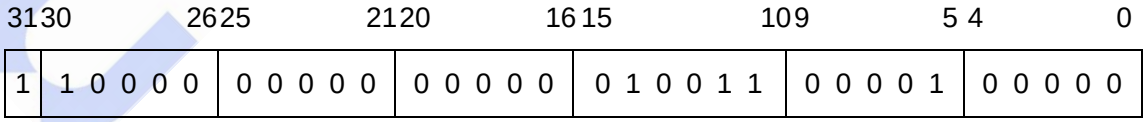
Attribute: Privileged instruction

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:



XOR – Bitwise XOR

Unified instruction

Grammar	Operation	Compiling result

xor rz, rx	RZ ← RZ ^ 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 ← RX ^ 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 bit: No influence

Exception: None

16-bit instruction

Operation: RZ ← RZ ^ RX

Grammar: xor16 rz, rx

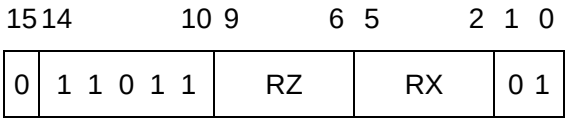
Description: Perform a bitwise XOR of RZ and RX values and save the result in RZ.

Influence on flag bit: No influence

Restriction: The range of register is r0-r15.

Exception: None

Instruction format:



32-bit instruction

Operation: RZ ← RX ^ RY

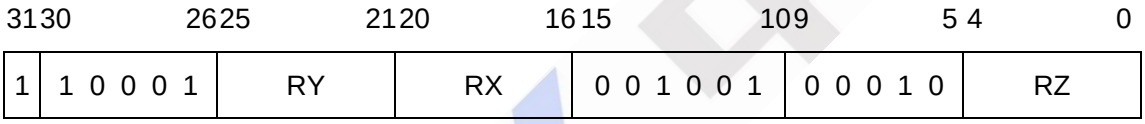
Grammar: xor32 rz, rx, ry

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

Influence on flag bit: No influence

Exception: None

Instruction format:



XORI – Bitwise XOR immediate

Unified instruction

Grammar	Operation	Compiling result
xori rz, rx, imm16	RZ ← RX ^ 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 0x0-0xFFF.

Exception: None

32-bit instruction

Operation: RZ ← RX ^ 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 0x0-0xFFF.

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} ←{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 ← RX[OIMM5 - 1]

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

Exception: None

32-bit instruction

Operation: {RZ,C} ←{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 ← 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 ← zero_extend(RX[31:24]);
	if (RX[31:24] == 0), then
	C ← 0;
	else
	C ← 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 ← zero_extend(RX[31:24]);
if (RX[31:24] == 0), then
C ← 0;
else
C ← 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 ← zero_extend(RX[23:16]);
	if (RX[23:16] == 0), then
	C ← 0;
	else
	C ← 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 ← zero_extend(RX[23:16]);
if (RX[23:16] == 0), then
C ← 0;
else
C ← 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 ← zero_extend(RX[15:8]);
	if (RX[15:8] == 0), then
	C ← 0;
	else
	C ← 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 ← zero_extend(RX[15:8]);
if (RX[15:8] == 0), then
C ← 0;
else
C ← 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 ← zero_extend(RX[7:0]);
	if (RX[7:0] == 0), then
	C ← 0;
	else
	C ← 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 ← zero_extend(RX[7:0]);
if (RX[7:0] == 0), then
C ← 0;
else
C ← 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

ZEXTB – Extract byte and extend unsigned#

Unified instruction

Grammar	Operation	Compiling result
zextb rz, rx	RZ ← zero_extend(RX[7:0]);	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

Exception: None

16-bit instruction

Operation: RZ ← 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

ZEXTH – Extract half-word and extend unsigned#

Unified instruction

Grammar	Operation	Compiling result
zexth rz, rx	RZ ← zero_extend(RX[15:0]);	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

Exception: None

16-bit instruction

Operation: RZ ← 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

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