凌陽科技股份有限公司
SUNPLUS TECHNOLOGY CO., LTD.

unSP ISA-1.2 Manual

19, Innovation Road 1, Science-Based Industrial Park, Hsin-Chu, Taiwan, R.O.C.
TEL : +886-3-5786005 FAX : +886-3-5786006 WWW : http://www.sunplus.com.tw
IP Design Department

FEATURES

- 16-bits micro controller with some DSP functions.
- Memory bus interface
  - Address width: 22 bits
  - Data width: 16 bits
  - 4M words (8M bytes) memory space
  - 64 banks/ 64k words per bank
- 13*16-bits registers
  - 4 general registers (R1~R4)
  - 4 secondary registers (SR1~SR4)
  - 4 system registers (SP, BP, SR, PC)
  - 1 flag register (FR)
- 10 interrupts
  - 1 fast interrupt (FIQ)
  - 8 normal interrupt (IRQ0-IRQ7)
    - Support IRQ nested mode with user customized priority
  - Software interrupt (BRK)
- 6 addressing modes
  - Register
  - Immediate
  - Direct
  - Indirect
  - Multi-indirect
  - Displacement
- 16-bits multiplication
  - 3 operation modes: signed*signed, unsigned*signed, unsigned*unsigned
  - Integer/Fraction mode
- 16-levels inner product operation
  - 3 operation modes: signed*signed, unsigned*signed, unsigned*unsigned
  - Integer/Fraction mode
  - 4 guard bits to avoid overflow
- Non-restoring Division

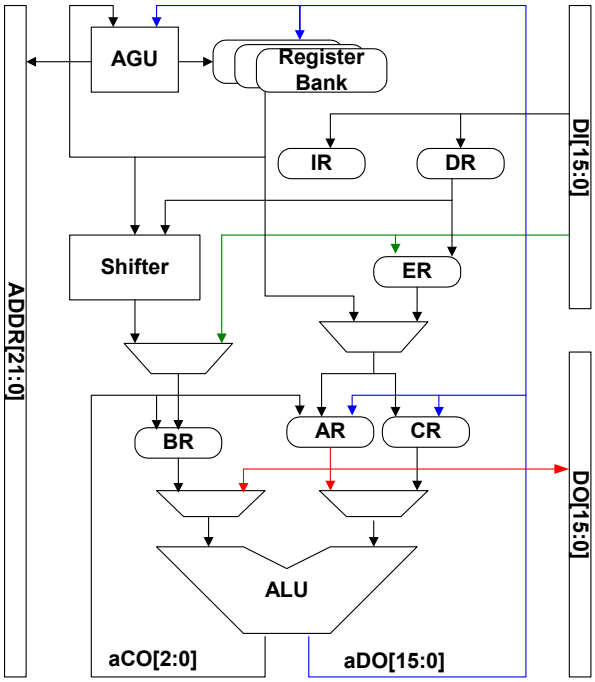
- 32-bits dividend and 16-bits divisor
- Need 16 continuous operations (DIVS, DIVQ) to generate correct quotient

- Bit-operation
  - Bit test/ set/ clear/ inverse operation
  - Destination can be register or memory
- Effective-exponent detect operation
- 16-bits shift operation
  - Support 32-bits shift operation by combining 2 shift instructions
- Support DMA function
- Support power down/sleep mode

ARCHITECTURE

The design goal of unSP is to achieve high performance with low cost, so it uses the traditional multi-cycle architecture, the organization of unSP is illustrated as below. The principal components are:

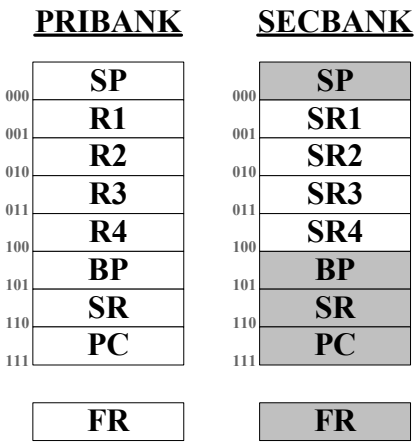
- The register bank, which stores the processor state. There are 4 general registers (R1-R4), 4 secondary bank registers (SR1-SR4), 4 system registers (SP, BP, SR, PC) and 1 flag register (FR) in the register bank.
- The instruction register (IR), which store the current instruction fetched from data bus. The decoder will generate all control signals for data path according to the instruction register.
- The data register (DR), which store the second word of current instruction if instruction length is 2 words.
- The address generator unit (AGU), which select and hold all memory address and generate sequential address when required.
- The shifter, which can shift or rotate one operand by any number of 4-bits.
- The ALU, which performs the arithmetic and logic functions required by current instruction.



REGISTER DESCRIPTIONS

■ Registers Bank

unSP ISA 1.2 add 4 registers (SR1~SR4) for interrupt service routine to reduce the push/pop effort, user can use SECBANK On/Off instruction to switch register bank. If SECBANK mode is on, all access to R1~R4 will be redirected to SR1~SR4.



- Primary Bank
  - Stack Pointer (SP)
  - General Register (R1~R4)
  - Base Pointer (BP)
  - Status Register (SR)
  - Program Counter (PC)
  - Flag Register (FR)
- Secondary Bank
  - Secondary Register (SR1~SR4)

■ Status Register (SR)

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
DS						N	Z	S	C	CS					

- Conditional Flag

- N: Negative flag, denotes the 16th bit of ALU result.
- Z: Zero flag, denotes whether the ALU result is zero.
- S: Sign flag, denotes the MSB(18th) bit of ALU result.
- C: Carry flag, denotes the 17th bit of ALU result

- Data Segment (DS)
  - Data segment can be used to access memory large than 64K words memory space
- Code Segment (CS)
  - Code segment can be used to fetch instruction location large than 64K words memory space
- Code segment and Data segment will be updated automatically when the target address crossing segment boundary.

■ Flag Register (FR)

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

- AQ: DIVS/DIVQ AQ flag, default is 0.
- BNK: Register bank, default is 0 (PRIBANK).
- FRA: Fraction mode, default is 0 (OFF).
- FIR: FIR MOVE mode, default is 0 (FIR Move On).
- SB: Shift buffer/Guard bits, default is 4'b0000.
- FIQ: FIQ Enable, default is 0 (Disable)
- IRQ: IRQ Enable, default is 0 (Disable)
- INE: IRQ nest mode, default is 0 (OFF)
- PRI: IRQ priority register, default is 4'b1000 after reset, if IRQ Nest mode is On and any IRQ occurred, PRI register will be set as the IRQ priority before cpu executing IRQ service routine and only the IRQ with higher priority can interrupt it, user can customize the IRQ nest behavior with setting priority register.

  Priority: IRQ0>IRQ1>IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7

  Note: FIQ still has highest priority than any IRQ if FIQ is enable.

  For example:

  1. If PRI is 1000, all IRQ 0-7 are enable
  2. If PRI is 0000, all IRQ 0-7 are disable
  3. IRQ3 occurred, PRI will be set to 0011, only IRQ0-2 are enable

ADDRESSING MODES

6 addressing modes

- Register
  - Users can shift the source register (Rs) value first and then executing alu operation with destination register (Rd), place the result at destination register.
- Immediate
  - Users can do alu operation between source register and a 6-bits or a 16-bits immediate value, then place the result at destination register.
- Direct
  - Users can do alu operation between source register and the value at memory location indexed by 6-bits or 16-bits operand, then place the result at destination register
  - Users can store register value to a memory location indexed by 6-bits or 16-bits operand.
- Indirect
  - Users can do alu operation between destination register and the value at memory location indexed by source register, then place the result at destination register.
  - Users can store destination register value to a memory location indexed by source register.
  - The source register can be increased by 1 before alu operation or increased/decreased by 1 after alu operation.
  - Users can use the "D:" indicator to access memory location larger than 64k words, if the "D:" indicator is used, the high 6-bits of accessing address will use data segment (DS) value or be zeroed.
- Multi-indirect
  - Users can push or pop multiple registers' value to memory location indexed by stack pointer (SP)
- Displacement
  - Users can do alu operation between destination register and the value at memory location indexed by base pointer (BP) with a 6-bits displacement.

INSTRUCTION SET

unSP Operation Mode

Switch Register Bank

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	0	0	1	0	1	BNK

Syntax:

SECBANK ON/OFF;

Flag Register:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

Switch register bank. The BNK=1 denotes secondary register bank is used, else the primary register bank is used. The default value of BNK is 0 (OFF).

Example:

SECBANK ON;

Fraction Mode

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	0	0	0	1	1	FRA

Syntax:

FRACTION ON/OFF;

Flag Register:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

Switch to fraction mode. If fraction mode is on, the result of multiplication will be shift 1 bit left to present the correct result of fraction number multiplication. The FRA=1'b1 denotes the fraction mode ON, else it denotes OFF. The default value of FRA is 1'b0 (OFF).

Example:

FRACTION ON;

FIR Move Mode

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	0	0	0	1	0	FIR

Syntax:

FIR_MOV ON/OFF;

Flag Register:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

Cycles:

2

Word length:

1

Version:

ALL

Description:

Switch FIR MOVE mode on/off. If the FIR Move mode is on, the value stored in multiplication parameter array indexed by Rd will be moved forward while executing MULS instruction. The FIR=0 denotes the FIR MOVE mode ON, else it denotes OFF. The default value of FIR is 0 (ON).

Example:

FIR_MOV ON;

FIQ ON/OFF

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	0	0	1	1	FIQ	0

Syntax:

FIQ ON/OFF;

Flag Register:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

Cycles:

2

Word length:

1

Version:

ALL

Description:

Enable/Disable FIQ interrupt. The FIQ=1 denotes the FIQ enable, else it denotes disable. The default value of FIQ is 0 (disable).

Example:

FIQ ON;

IRQ ON/OFF

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	0	0	1	0	0	IRQ

Syntax:

IRQ ON/OFF;

Flag Register:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

Cycles:

2

Word length:

1

Version:

ALL

Description:

Enable/Disable IRQ interrupt. The IRQ=1 denotes IRQ enable, else it denotes IRQ disable. The default value of IRQ is 0 (disable).

Example:

IRQ ON;

Interrupt Setting (INT)

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	0	0	0	0	FIQ	IRQ

Syntax:

INT FIQ/IRQ/OFF;

Flag Register:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

Cycles:

2

Word length:

1

Version:

ALL

Description:

Enable/Disable FIQ/IRQ interrupt at the same time.

Instruction	FIQ	IRQ	OP[1:0]
INT OFF	OFF	OFF	00
INT IRQ	OFF	ON	01
INT FIQ	ON	OFF	10
INT FIQ,IRQ	ON	ON	11

Example:

```
INT   FIQ,IRQ;      // Enable FIQ, IRQ (OP[1:0]= 2'b11)
INT   FIQ;          // Enable FIQ, disable IRQ (OP[1:0]= 2'b10)
INT   OFF;          // disable FIQ, IRQ (OP[1:0]= 2'b00)
```

IRQ Nest Mode

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	0	0	1	1	INE	1

Syntax:

IRQNEST ON/OFF;

Flag Register:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	AQ	BNK	FRA	FIR	SB				FIQ	IRQ	INE	PRI			

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

Switch IRQ NEST mode On/Off. If IRQ NEST mode is on, IRQ interrupt which priority greater than the PRI register can be accepted while cpu executing IRQ service routine, in such case cpu will push FR/SR/PC into stack and change the PRI register with IRQS value before entering IRQ service routine, and restore PC/SR/FR from stack while leaving IRQ service routine. The INE=1 denotes the IRQ NEST mode ON, else it denotes OFF. The default value of INE is 0 (OFF).

Example:

IRQNEST ON;

Data Processing Instruction

ALU operations with 6-bits immediate value

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ALU_OP				Rd			0	0	1	IM6					

Syntax:

Rd {ALU_OP}= IM6, {Carry♦}; Rd: R0~R6

Conditional Flag:

N	Z	S	C
●	●	●	●

Cycles:

2

Word length:

1

Version:

ALL

Description:

Executing ALU operations between register and 6-bits immediate value, the result will be placed at destination register.

ALU_OP	Type	Operation	N	Z	S	C	Syntax
0000	ADD	a + b	●	●	●	●	Rd += IM6;
0001	ADC	a + b + carry	●	●	●	●	Rd += IM6, Carry;
0010	SUB	a + ~b + 1	●	●	●	●	Rd -= IM6;
0011	SBC	a + ~b + carry	●	●	●	●	Rd -= IM6, Carry;
0100	CMP	a + ~b + 1	●	●	●	●	CMP Rd, IM6;
0110	NEG	~b + 1	●	●	-	-	Rd =- IM6;
1000	XOR	a ^ b	●	●	-	-	Rd ^= IM6;
1001	LOAD	a = b	●	●	-	-	Rd = IM6;
1010	OR	a | b	●	●	-	-	Rd | = IM6;
1011	AND	a & b	●	●	-	-	Rd &= IM6;
1100	TEST	a & b	●	●	-	-	TEST Rd, IM6;

Example:

R1 += 0x20, Carry; // R1= R1 + 0x20 + SR[6]

♦ Carry is a flag from ALU operation, and it is stored in the SR[6].

ALU operations with 16-bits immediate value

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ALU_OP				Rd			1	0	0	0	0	1	Rs		

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
IM16															

Syntax:

Rd = Rs {ALU_OP} IM16, {Carry};

Conditional Flag:

N	Z	S	C
●	●	●	●

Cycles:

4 / 5*

Word length:

2

Version:

ALL

Description:

Executing ALU operations between source register and 16-bits immediate value, the result will be placed at destination register.

ALU_OP	Type	Operation	N	Z	S	C	Syntax
0000	ADD	a + b	●	●	●	●	Rd = Rs + IM16;
0001	ADC	a + b + carry	●	●	●	●	Rd = Rs + IM16, Carry;
0010	SUB	a + ~b + 1	●	●	●	●	Rd = Rs − IM16;
0011	SBC	a + ~b + carry	●	●	●	●	Rd = Rs − IM16, Carry;
0100	CMP	a + ~b + 1	●	●	●	●	CMP Rs, IM16;
0110	NEG	~b + 1	●	●	-	-	Rd = − IM16;
1000	XOR	a ^ b	●	●	-	-	Rd = Rs ^ IM16;
1001	LOAD	a = b	●	●	-	-	Rd = IM16;
1010	OR	a | b	●	●	-	-	Rd = Rs | IM16;
1011	AND	a & b	●	●	-	-	Rd = Rs & IM16;
1100	TEST	a & b	●	●	-	-	TEST Rs, IM16;

Example:

```
PC = BP + 0x2400;        // PC = BP + 0x2400 (Fast table-lookup and JUMP)
```

* Write to program counter (PC)

ALU operations with shifted register operand

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ALU_OP				Rd			1	SFT_OP			n		Rs		

Syntax:

Rd {ALU_OP}= Rs {SFT_OP n}, {Carry};

Conditional Flag:

N	Z	S	C
●	●	●	●

Cycles:

3 / 5*

Word length:

1

Version:

ALL

Description:

Executing ALU operation between source register and destination register, the result will be placed at destination register. Optional shift operation can be applied to source register (Rs) before ALU operation. 5 kinds of shift operations are supported, and all of them can shift 4-bits long (n=1~4).

SFT_OP	Type	Meaning
000		Disable shifted operand, and OP[4:3]=2'b00
001	ASR	Arithmetic shift right
010	LSL	Logic shift left
011	LSR	Logic shift right
100	ROL	Rotate left
101	ROR	Rotate right

n	OP[4:3]
1	00
2	01
3	10
4	11

An additional 4-bits shift buffer (SB♦) will store the shifted-out data automatically.

* write to PC
♦ SB= FR[10:7]

Suppose Rs={B_F, B_E, …, B_0}, SB={S_3, S_2, S_1, S_0}, and n=3

Input:

Rs | B_F | B_E | B_D | B_C | B_B | B_A | B_9 | B_8 | B_7 | B_6 | B_5 | B_4 | B_3 | B_2 | B_1 | B_0 SB | S_3 | S_2 | S_1 | S_0

Suppose n=3, after shift op of

ASR: (Shift Right with MSB of Rs)

Rd | B_F | B_F | B_F | B_F | B_E | B_D | B_C | B_B | B_A | B_9 | B_8 | B_7 | B_6 | B_5 | B_4 | B_3 SB | B_2 | B_1 | B_0 | S_3

LSL: (Shift Left)

SB | S_0 | B_F | B_E | B_D Rd | B_C | B_B | B_A | B_9 | B_8 | B_7 | B_6 | B_5 | B_4 | B_3 | B_2 | B_1 | B_0 | 0 | 0 | 0

LSR: (Shift Right)

Rd | 0 | 0 | 0 | B_F | B_E | B_D | B_C | B_B | B_A | B_9 | B_8 | B_7 | B_6 | B_5 | B_4 | B_3 SB | B_2 | B_1 | B_0 | S_3

ROL: (Rotate Left with SB)

SB | S_0 | B_F | B_E | B_D Rd | B_C | B_B | B_A | B_9 | B_8 | B_7 | B_6 | B_5 | B_4 | B_3 | B_2 | B_1 | B_0 | S_3 | S_2 | S_1

ROR: (Rotate Right with SB)

Rd | S_2 | S_1 | S_0 | B_F | B_E | B_D | B_C | B_B | B_A | B_9 | B_8 | B_7 | B_6 | B_5 | B_4 | B_3 SB | B_2 | B_1 | B_0 | S_3

ALU_OP	Type	Operation	N	Z	S	C	Syntax
0000	ADD	a + b	●	●	●	●	Rd += Rs SFT_OP n;
0001	ADC	a + b + carry	●	●	●	●	Rd += Rs SFT_OP n, Carry;
0010	SUB	a + ~b + 1	●	●	●	●	Rd -= Rs SFT_OP n;
0011	SBC	a + ~b + carry	●	●	●	●	Rd -= Rs SFT_OP n, Carry;
0100	CMP	a + ~b + 1	●	●	●	●	CMP Rd, Rs SFT_OP n;
0110	NEG	~b + 1	●	●	-	-	Rd =- Rs SFT_OP n;
1000	XOR	a ^ b	●	●	-	-	Rd ^= Rs SFT_OP n;
1001	LOAD	a = b	●	●	-	-	Rd = Rs SFT_OP n;
1010	OR	a | b	●	●	-	-	Rd | = Rs SFT_OP n;
1011	AND	a & b	●	●	-	-	Rd &= Rs SFT_OP n;
1100	TEST	a & b	●	●	-	-	TEST Rd, Rs SFT_OP n;

Example:

```
BP += R1 ASR 4, Carry;      // BP = BP + ( R1/(2**8) ) + SR[6]
BP = R1 LSL 2;              // BP = R1 << 2
```

ALU operations with 6-bits direct memory addressing

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ALU_OP				Rd			1	1	1	A6					

Syntax:

Rd {ALU_OP}= [A6], {Carry};

Conditional Flag:

N	Z	S	C
●	●	●	●

Cycles:

5 / 6*

Word length:

1

Version:

ALL

Description:

Executing ALU operation between register and the value at memory location indexed by 6-bits operand.

ALU_OP	Type	Operation	N	Z	S	C	Syntax
0000	ADD	a + b	●	●	●	●	Rd += [A6];
0001	ADC	a + b + carry	●	●	●	●	Rd += [A6], Carry;
0010	SUB	a + ~b + 1	●	●	●	●	Rd -= [A6];
0011	SBC	a + ~b + carry	●	●	●	●	Rd -= [A6], Carry;
0100	CMP	a + ~b + 1	●	●	●	●	CMP Rd, [A6];
0110	NEG	~b + 1	●	●	-	-	Rd =- [A6];
1000	XOR	a ^ b	●	●	-	-	Rd ^= [A6];
1001	LOAD	a = [b]	●	●	-	-	Rd = [A6];
1010	OR	a | b	●	●	-	-	Rd | = [A6];
1011	AND	a & b	●	●	-	-	Rd &= [A6];
1100	TEST	a & b	●	●	-	-	TEST Rd, [A6];
1101	STORE	[a] = b	-	-	-	-	[A6] = Rd;

Example:

```
                    //Init_Q is located in memory 0x00 ~ 0x3f
BP = [Init_Q];      //LOAD content of Init_Q to BP
[Init_Q] = R2;      //STORE R2 to MEM[Init_Q]
```

* write to PC

ALU operations with 16-bits direct memory addressing

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ALU_OP				Rd			1	0	0	0	1	W	Rs		

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
A16															

Syntax:

Rd = Rs {ALU_OP} [A16], {Carry}; (W=0)

[A16] = Rd {ALU_OP} Rs, {Carry}; (W=1)

Conditional Flag:

N	Z	S	C
●	●	●	●

Cycles:

7 / 8*

Word length:

2

Version:

ALL

Description:

Executing ALU operation between register and the value at memory location indexed by 16-bits operand.

ALU_OP	W	Type	Operation	N	Z	S	C	Syntax
0000	0	ADD	a + b	●	●	●	●	Rd = Rs + [A16];
0001		ADC	a + b + carry	●	●	●	●	Rd = Rs + [A16], Carry;
0010		SUB	a + ~b + 1	●	●	●	●	Rd = Rs − [A16];
0011		SBC	a + ~b + carry	●	●	●	●	Rd = Rs − [A16], Carry;
0100		CMP	a + ~b + 1	●	●	●	●	CMP Rs, [A16];
0110		NEG	~b + 1	●	●	-	-	Rd = − [A16];
1000		XOR	a ^ b	●	●	-	-	Rd = Rs ^ [A16];
1001		LOAD	a = b	●	●	-	-	Rd = [A16];
1010		OR	a | b	●	●	-	-	Rd = Rs | [A16];
1011		AND	a & b	●	●	-	-	Rd = Rs & [A16];
1100		TEST	a & b	●	●	-	-	TEST Rs, [A16];
0000	1	ADD	a + b	●	●	●	●	[A16] = Rd + Rs;
0001		ADC	a + b + carry	●	●	●	●	[A16] = Rd + Rs, Carry;

* write to PC

0010		SUB	a + ~b + 1	●	●	●	●	[A16] = Rd — Rs;
0011		SBC	a + ~b + carry	●	●	●	●	[A16] = Rd — Rs, Carry;
0100		CMP	a + ~b + 1	●	●	●	●	CMP Rd, Rs;
0110		NEG	~b + 1	●	●	-	-	[A16] = — Rd;
1000		XOR	a ^ b	●	●	-	-	[A16] = Rd ^ Rs;
1010		OR	a | b	●	●	-	-	[A16] = Rd | Rs;
1011		AND	a & b	●	●	-	-	[A16] = Rd & Rs;
1100		TEST	a & b	●	●			TEST Rd, Rs;
1101		STORE	[a] = b	-	-	-	-	[A16] = Rd;

Example:

```
BP = R1 + [Q_table];          // BP = R1 + MEM[Q_table], where Q_table is in 0x0000~0xffff
[Q_table+10] = BP ^ R2;       // BP ^ R2 is assigned to MEM[Q_table+10]
```

ALU operations with indirect memory addressing

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ALU_OP				Rd			0	1	1	D	@		Rs		

Syntax:

Rd {ALU_OP}= {D:♣}[Rs@], {Carry};

Conditional Flag:

N	Z	S	C
●	●	●	●

Cycles:

6 / 7*

Word length:

1

Version:

ALL

Description:

Executing ALU operation between destination register and the value at memory location indexed by source register. Users can use the "D:" indicator to access memory space large than 64K words. If "D:" indicator is used, the MSB 6-bits of accessing address will use data segment (DS) value and automatic update while crossing segment boundary else the MSB 6-bits will be zeroed. Source register can be increased by 1 before ALU operation or increased/decreased by 1 after ALU operation.

ALU_OP	Type	Operation	N	Z	S		Syntax
0000	ADD	a + b	●	●	●	●	Rd += {D:}[Rs@];
0001	ADC	a + b + carry	●	●	●	●	Rd += {D:}[Rs@], Carry;
0010	SUB	a + ~b + 1	●	●	●	●	Rd -= {D:}[Rs@];
0011	SBC	a + ~b + carry	●	●	●	●	Rd -= {D:}[Rs@], Carry;
0100	CMP	a + ~b + 1	●	●	●	●	CMP Rd, {D:}[Rs@];
0110	NEG	~b + 1	●	●	-	-	Rd =- {D:}[Rs@];
1000	XOR	a ^ b	●	●	-	-	Rd ^= {D:}[Rs@];
1001	LOAD	a = [b]	●	●	-	-	Rd = {D:}[Rs@];
1010	OR	a | b	●	●	-	-	Rd | = {D:}[Rs@];
1011	AND	a & b	●	●	-	-	Rd &= {D:}[Rs@];
1100	TEST	a & b	●	●	-	-	TEST Rd, {D:}[Rs@];
1101	STORE	[a] = b	-	-	-	-	{D:}[Rs@] = Rd;

♣ if use the {D:} indicator, the MSB 6-bits of access address will use data segment (DS) value else will be zeroed .
* write to PC

The prefix of source register

Rs@	OP[4:3]	Meaning
Rs	2'b00	No increment/decrement
Rs--	2'b01	After ALU_OP, Rs= Rs-1
Rs++	2'b10	After ALU_OP, Rs= Rs+1
++Rs	2'b11	Before ALU_OP, Rs= Rs+1

Example:

```
R1 &= D:[R2++];      // R1 = R1& MEM[{SR[15:10],R2}], then R2= R2+1
                     // if R2= 0xffff, after R2= R2+1, SR[15:10]= SR[15:10]+1 and R2=0
D:[R2--] = R1;       // MEM[{SR[15:10],R2}]= R1, then R2= R2-1
                     // if R2=0, after R2= R2-1, SR[15:10]= SR[15:10]- 1 and R2= 0xffff
CMP R1, [++R2];      // Compare R1, MEM[R2+1]
```

ALU operations with 6-bits displacement memory addressing

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ALU_OP				Rd			0	0	0	IM6					

Syntax:

Rd {ALU_OP}= [BP+IM6], {Carry}; Rd : R0~R6

Conditional Flag:

N	Z	S	C
●	●	●	●

Cycles:

6

Word length:

1

Version:

ALL

Description:

Executing ALU operation between destination register and the value at memory location indexed by base pointer (BP) with a 6-bits displacement.

ALU_OP	Type	Operation	N	Z	S	C	Syntax
0000	ADD	a + b	●	●	●	●	Rd += [BP+IM6];
0001	ADC	a + b + carry	●	●	●	●	Rd += [BP+IM6], Carry;
0010	SUB	a + ~b + 1	●	●	●	●	Rd -= [BP+IM6];
0011	SBC	a + ~b + carry	●	●	●	●	Rd -= [BP+IM6], Carry;
0100	CMP	a + ~b + 1	●	●	●	●	CMP Rd, [BP+IM6];
0110	NEG	~b + 1	●	●	-	-	Rd =- [BP+IM6];
1000	XOR	a ^ b	●	●	-	-	Rd ^= [BP+IM6];
1001	LOAD	a = [b]	●	●	-	-	Rd = [BP+IM6];
1010	OR	a | b	●	●	-	-	Rd | = [BP+IM6];
1011	AND	a & b	●	●	-	-	Rd &= [BP+IM6];
1100	TEST	a & b	●	●	-	-	TEST Rd, [BP+IM6];
1101	STORE	[a] = b	-	-	-	-	[BP+IM6] = Rd;

Example:

```
[BP+0x00] = R1;          // MEM[BP+0x00] = R1 (STORE)
```

Bit operations with register

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Rd			0	0	0	BIT_OP		0	Rs		

Or

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Rd			0	0	1	BIT_OP		offset			

Syntax:

BIT_OP Rd, Rs; Rd: R0~R6

BIT_OP Rd, offset;

Conditional Flag:

N	Z	S	C
-	●	-	-

Cycles:

4

Word length:

1

Version:

ISA 1.2 and above

Description:

Executing bit operation at register value, the original value of accessing bit will affect the zero flag, that is, if the original bit is zero, the Zero flag will be 1 else will be 0.

Notes that only the least significant 4 bits of source register (Rs[3:0]) are used and only R0~R6 is available for the destination register (Rd).

BIT_OP	Type	Syntax	Meaning
00	TSTB	TSTB Rd, Rs;	Z= (Rd[Rs[3:0]]== 1)? 1'b0: 1'b1
		TSTB Rd, offset;	Z= (Rd[offset]== 1)? 1'b0: 1'b1
01	SETB	SETB Rd, Rs;	Rd[Rs[3:0]]= 1
		SETB Rd, offset;	Rd[offset]= 1
10	CLRB	CLRB Rd, Rs;	Rd[Rs[3:0]]= 0
		CLRB Rd, offset;	Rd[offset]= 0
11	INVB	INVB Rd, Rs;	Rd[Rs[3:0]]= ~Rd[Rs[3:0]]
		INVB Rd, offset;	Rd[offset]= ~Rd[offset]

Example:

```
INVB R4, R5;      // if R5[3:0]= 0x3, R4[3]= ~R4[3]
CLRB R3, 10;      // R3[10]= 0
```

Bit operations with indirect memory addressing

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Rd			1	0	D	BIT_OP		0	Rs		

Or

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Rd			1	1	D	BIT_OP		offset			

Syntax:

BIT_OP {D:}[Rd], Rs; Rd: R0~R6

BIT_OP {D:}[Rd], offset;

Conditional Flag:

N	Z	S	C
-	●	-	-

Cycles:

7

Word length:

1

Version:

ISA 1.2 and above

Description:

Executing bit operation with the value at memory location indexed by register. Users can use the "D:" indicator to access memory space large than 64K words, If "D:" indicator is used, the MSB 6-bits of accessing address will use data segment (DS) value else will be zeroed. The original value of accessing bit will affect the zero flag, that is, if the original bit is zero, the Zero flag will be 1 else will be 0.

Notes that only the least significant 4 bits of source register (Rs[3:0]) are used and only R0~R6 is available for the destination register (Rd).

BIT_OP	Type	Syntax	Meaning
00	TSTB	TSTB {D:}[Rd], Rs;	Z= (MEM[{DS,Rd}][Rs[3:0]]== 1)? 1'b0: 1'b1
		TSTB {D:}[Rd], offset;	Z= (MEM[{DS,Rd}][offset]== 1)? 1'b0: 1'b1
01	SETB	SETB {D:}[Rd], Rs;	MEM[{DS,Rd}][Rs[3:0]]= 1
		SETB {D:}[Rd], offset;	MEM[{DS,Rd}][offset]= 1
10	CLRB	CLRB {D:}[Rd], Rs;	MEM[{DS,Rd}][Rs[3:0]]= 0
		CLRB {D:}[Rd], offset;	MEM[{DS,Rd}][offset]= 0
11	INVB	INVB {D:}[Rd], Rs;	MEM[{DS,Rd}][Rs[3:0]]= ~ MEM[{DS,Rd}][Rs[3:0]]
		INVB {D:}[Rd], offset;	MEM[{DS,Rd}][offset]= ~ MEM[{DS,Rd}][offset]

Example:

```
SETB [R1], R3;      // if R3[3:0]= 0x3, MEM[R1][3]= 1
SETB D:[R1], 13;    // MEM[{DS,R1}][13]= 1
```

Shift operations

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Rd			1	0	LSFT_OP			1	Rs		

Syntax:

Rd= Rd LSFT_OP Rs*;

Conditional Flag:

N	Z	S	C
-	-	-	●

Cycles:

8

Word length:

1

Version:

ISA 1.2 and above

Description:

This is a 16-bits multi-cycle shift operation, but it can support 32-bits shift operation by combining 2 shift operations. The result of 32-bits shift operation is placed at MR, the shifted bits and R4/R3 will be applied an OR operation automatically. Shift operations can support ASR/ASROR/LSL/LSLOR/LSR/LSROR/ROL/ROR commands. The ROR/ROL operation will shift with carry flag, and the drop bit will place at carry flag after operation. Only R0~ R6 is available for the destination register (Rd).

LSFT_OP	Type	Syntax
000	ASR	Rd= Rd ASR Rs;
001	ASROR	MR|= Rd ASR Rs;
010	LSL	Rd= Rd LSL Rs;
011	LSLOR	MR|= Rd LSL Rs;
100	LSR	Rd= Rd LSR Rs;
101	LSROR	MR|= Rd LSR Rs;
110	ROL	Rd= Rd ROL Rs;
111	ROR	Rd= Rd ROR Rs;

Example:

```
R2= R2 ASR R1;    // 16-bits arithmetic right shift

R3= R3 LSR R1;    // 32-bits arithmetic right shift
MR|= R4 ASR R1;
```

* Rs[4:0] valid: ASR/ASROR/LSL/LSLOR/LSR/LSROR; Rs[3:0] valid: ROL/ROR

Multiplication (MUL)

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	Ss	Rd			Sd	0	0	0	0	1	Rs		

Syntax:

MR= Rd*Rs, {ss/us/uu};

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

12 (signed*signed, unsigned*signed)

13 (unsigned*unsigned)

Word length:

1

Version:

signed*signed, unsigned*signed : ALL version

unsigned*unsigned : ISA 1.2 and above

Description:

This operation is to multiply two registers and place the result at MR. It supports 3 kinds of multiplication, signed*signed, signed*unsigned, and unsigned*unsigned. The signed-to-signed multiplication is used as default. If the fraction mode is ON, the result of multiplication will be shifted 1-bit left. Only R0~ R6 is available for the destination register (Rd).

The encoding of {Ss, Sd}

{Ss, Sd}	Rd*Rs
00	unsigned*unsigned
10	unsigned*signed
11	signed*signed

Example:

```
MR= R3*R1;        // signed*signed
MR= R1*R2, us;    // unsigned*signed
```

Note:

MUL only support signed*signed, unsigned*signed, unsigned*unsigned types to increase the encoding space of machine code. If user uses the signed*unsigned type in the program, the assembler will exchange the Rd, Rs position in the output machine code. For example, the instruction “MR=R1*R2,su” will be compiled the same as “MR=R2*R1,us”.

Inner product operation (MULS)

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	Ss	Rd			Sd	1	N				Rs		

Syntax:

MR= [Rd]*[Rs], {ss/us/uu}, N♣;

Conditional Flag:

N	Z	S	C
-	-	●	-

Cycles:

6+10*N (signed*signed, signed*unsigned)

6+11*N (unsigned*unsigned)

Word length:

1

Version:

signed*signed, unsigned*signed : ALL version

unsigned*unsigned : ISA 1.2 and above

Description:

This operation is using a 36-bits arithmetic unit to sum up a consecutive register multiplication and propagate coefficients for next FIR. It supports 3 kinds of multiplication, signed*signed, unsigned*signed, and unsigned*unsigned. The signed-to-signed multiplication is used as default. If the fraction mode is ON, the result of every multiplication will be shifted 1-bit left and then sum up. The pointer register Rd and Rs will be adjusted automatically. If FIR MOVE mode is ON and N>1, the contents of memory pointed by Rd are also moved forward. After the operation, the 4-bits MSB of ALU (guard bits) will be placed at shift buffer (SB). The sign flag will be set if overflow occurred with the final result.

Example:

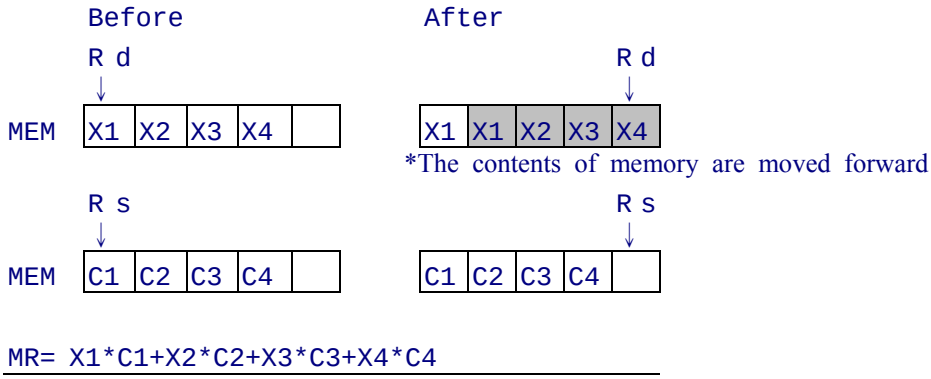
MR= Rd*Rs, 4;

```
FIR MOVE MODE is OFF

          Before                    After

          R d                                 R d
          ↓                                   ↓
MEM       X1 | X2 | X3 | X4 |       X1 | X2 | X3 | X4 |

          R s                                 R s
          ↓                                   ↓
MEM       C1 | C2 | C3 | C4 |       C1 | C2 | C3 | C4 |
```

♣ The inner product levels, N=0-15 and N=0 denotes the 16-level inner product operation.

```
MR= X1*C1+X2*C2+X3*C3+X4*C4
```

```
FIR MOVE MODE is ON
```




```
MR= X1*C1+X2*C2+X3*C3+X4*C4
```

Note:

The result of multiplication will be incorrect if the following conditions are both met:
(N>1) and (either Rd or Rs are set to R3 or R4) and (Rd and Rs are set to the same register)

This operation of previous version unSP ISA 1.0/ISA 1.1 doesn't change the sign flag, but the unSP ISA 1.2 will change this flag to indicate overflow condition.

MULS only support signed*signed, unsigned*signed, unsigned*unsigned types to increase the encoding space of machine code. If user uses the signed*unsigned type in the program, the assembler will exchange the Rd, Rs position in the output machine code. For example, the instruction "MR=[R1]*[R2], su, 4" will be compiled the same as "MR=[R2]*[R1], us, 4".

Divide-Quotient operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	1	-	-	0	1	1

Syntax:

DIVQ MR, R2;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

3

Word length:

1

Version:

ISA 1.2 and above

Description:

DIVQ uses the non-restoring division algorithm to yield a 1-bit quotient at each instruction. To implements a division with a 32-bits unsigned dividend and 16-bits unsigned divisor, the 32-bits dividend must be placed at MR, 16-bits divisor must be placed at R2, and the AQ* flag must be cleared before executing. Finally, the quotient will be placed at R3.

Example:

```
//    32-bits unsigned dividend / 16-bits unsigned divisor
//    0x0003_1713 / 0x0625

R4= 0x0003;        //
R3= 0x1713;        //    Load data
R2= 0x0625;        //

R1= FR;            //
CLRB R1, 0xe;      //    Clear AQ flag
FR= R1;            //

R1= 1;             //
R4= R4 LSL R1;     //    Shift 1-bit left
MR|= R3 LSL R1;    //
```

* AQ= FR[14], AQ flag determines the ADD or SUB operation in the non-restoring division algorithm

```
        R1= 0;
div_unsigned:           //      Implement an unsigned division with 16 iterations
        DIVQ MR, R2;
        R1+= 1;
        CMP R1, 0x10;
        JNE div_unsigned;
```

Divide-Sign operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	1	-	-	0	1	0

Syntax:

DIVS MR, R2;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

DIVS uses the non-restoring division algorithm to compute the sign of the quotient. To implements a division with a 32-bits signed dividend and 16-bits signed divisor, the 32-bits dividend must be placed at MR, 16-bits divisor must be placed at R2, and the AQ flag must be cleared. The DIVS instruction is executed at the beginning of the division. Then DIVQ instruction is executed repeatedly. Finally, the quotient will be placed at R3.

Output Formats

The format of a division result is based on the format of the input operands. The division logic has been designed to work most efficiently with fully fractional numbers. If the dividend is in M.N format (M bits before the binary point, N bits after), and the divisor is O.P format, the quotient's format will be (M-O+1).(N-P-1).

Integer Division

To generate an integer quotient, you must shift the dividend to the left one bit, placing it in 31.1 format. The output format for this division will be (31-16+1).(1-0-1), or 16.0. You must ensure that no significant bits are lost during the left shift, or an invalid result will be generated.

ERROR Conditions

There are two cases where an invalid or inaccurate result can be generated

1. Negative Divisor Error

   If you attempt to use a negative number as the divisor in signed division, the quotient generated may be one LSB less than the correct result unless the result should equal 0x8000. there are two ways to correct for this error

A. Avoid division by negative numbers. If your divisor is negative, take its absolute value and invert the sign of the quotient after division.

B. Check the result by multiplying the quotient by the divisor. Compare this value with the dividend, and if they are off by more than the value of the divisor, increase the quotient by one.

2. Unsigned Division Error

Unsigned divisions can produce erroneous results if the divisor is greater than 0x7FFF. If it is necessary to perform a such division, both operands should be shifted right one bit. This will maintain the correct orientation of operands.

Shifting both operands may result in a one LSB error in the quotient. This can be solved by multiplying the quotient by the original (not shifted) divisor. Subtract this value from the original dividend to calculate the error. If the error is greater than the divisor, add one to the quotient, if it is negative, subtract one from the quotient.

Example:

```
//    32-bits signed dividend / 16-bits signed divisor
//    0xffff_1713 / 0x0625

    R4= 0xffff;          //
    R3= 0x1713;          //    Load data
    R2= 0x0625;          //

    R1= FR;              //
    CLRB R1, 0xe;        //    Clear AQ flag
    FR= R1;              //

    R1= 1;               //
    R4= R4 LSL R1;       //    Shift 1-bit left
    MR|= R3 LSL R1;      //

    R1= 0;
    DIVS MR, R2;         //    Get the sign of the quotient

div_signed:              //    Implement an unsigned division with 15 iterations
    DIVQ MR, R2;
```

```
R1+= 1;
CMP R1, 0xf;
JNE div_signed;
```

EXP operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	1	-	-	1	0	0

Syntax:

EXP R2, R4;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

The EXP instruction derives the effective exponent of the R4 register to prepare for the normalization operation, and places the result in the R2. The result is equal to the number of the redundant sign bit in the R4.

R4																R2
F	**E**	**D**	**C**	**B**	**A**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
S	N	D	D	D	D	D	D	D	D	D	D	D	D	D	D	0
S	**S**	N	D	D	D	D	D	D	D	D	D	D	D	D	D	1
S	**S**	**S**	N	D	D	D	D	D	D	D	D	D	D	D	D	2
S	**S**	**S**	**S**	N	D	D	D	D	D	D	D	D	D	D	D	3
S	**S**	**S**	**S**	**S**	N	D	D	D	D	D	D	D	D	D	D	4
S	**S**	**S**	**S**	**S**	**S**	N	D	D	D	D	D	D	D	D	D	5
S	**S**	**S**	**S**	**S**	**S**	**S**	N	D	D	D	D	D	D	D	D	6
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	N	D	D	D	D	D	D	D	7
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	N	D	D	D	D	D	D	8
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	N	D	D	D	D	D	9
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	N	D	D	D	D	10
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	N	D	D	D	11
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	N	D	D	12
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	N	D	13
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	N	14
S	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	15

*(S)ign bit, (N)on-sign bit, (D)on't care bit

Example:

```
EXP R4, R2;        // If R4= 16'b1111_0111_0111_0000, then R2= 3
                   // If R4= 16'b0000_0000_0100_1111, then R2= 8
```

NOP operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	1	-	-	1	0	1

Syntax:

NOP;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

No operation, only increase PC to the next address.

Control Flow Instruction

Branch operations

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
BRANCH_OP				1	1	1	0	0	D*	IM6					

Syntax:

BRANCH_OP IM6;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

2 (not-taken) / 4 (taken)

Word length:

1

Version:

ALL

Description:

A conditional short jump instruction to local label (address within ±63 words).

BRANCH_OP	Type	Condition	Description
0000	JCC	C==0	carry clear
0000	JB	C==0	below (unsigned)
0000	JNAE	C==0	not above and equal (unsigned)
0001	JCS	C==1	carry set
0001	JNB	C==1	not below (unsigned)
0001	JAE	C==1	above and equal (unsigned)
0010	JSC	S==0	sign clear
0010	JGE	S==0	great and equal (signed)
0010	JNL	S==0	not less (signed)
0011	JSS	S==1	sign set
0011	JNGE	S==1	not great than (signed)
0011	JL	S==1	Less (signed)
0100	JNE	Z==0	not equal
0100	JNZ	Z==0	not zero
0101	JZ	Z==1	zero
0101	JE	Z==1	equal
0110	JPL	N==0	plus
0111	JMI	N==1	minus

* D=0 denotes the forward jump, else D=1 denotes the backward jump.

1000	JBE	Not (Z==0 and C==1)	below and equal (unsigned)
1000	JNA	Not (Z==0 and C==1)	not above (unsigned)
1001	JNBE	Z==0 and C==1	not below and equal (unsigned)
1001	JA	Z==0 and C==1	above (unsigned)
1010	JLE	Not (Z==0 and S==0)	less and equal (signed)
1010	JNG	Not (Z==0 and S==0)	not great (signed)
1011	JNLE	Z==0 and S==0	not less and equal (signed)
1011	JG	Z==0 and S==0	great (signed)
1100	JVC	N == S	not overflow (signed)
1101	JVS	N != S	overflow (signed)
1110	JMP	Always	unconditional branch

Example:

```
Loop:
    JCC Loop;      // Jump to Loop, if Carry flag = 0
```

Far jump to 22-bits indirect address in MR

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	0	1	1	-	-	-	-	-	-

Syntax:

GOTO MR;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

4

Word length:

1

Version:

ISA 1.2 and above

Description:

This is a far jump instruction with MR register. The 22-bits content of MR {R4[5:0],R3} will be used as destination address.

Example:

```
GOTO MR;          // Jump to {R4[5:0],R3}
```

Far jump to 22-bits immediate address

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	0	1	0	A22[21:16]					

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
A22[15:0]															

Syntax:

GOTO A22;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

5

Word length:

2

Version:

ISA 1.1 and above

Description:

Go to user's specified address. The 22-bits target address is range from 0x000000 to 0x3fffff. After this operation, the Code Segment (CS) will be updated automatically.

Example:

```
0x008010    GOTO far_func;
            ..
0x035678    far_func:
            ..
```

Data Segment Access Instruction

Assign Data Segment (DS) value with 6-bits immediate

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	0	0	0	IM6					

Syntax:

DS= IM6;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

Assign Data Segment (DS) value with a 6-bits immediate value.

Example:

DS=0x12;

Access Data Segment (DS) value with register

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	0	0	0	1	0	W	Rs		

Syntax:

Rs=Ds (W=0)

Ds=Rs (W=1)

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

Access Data Segment (DS) with register. Only 6-bits value of the source register (Rs[5:0]) will be set on DS. The zero-extended is used when getting DS segment.

Example:

```
DS=R1;
R2=DS;
```

Flag Register Access

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	0	0	0	1	1	W	Rs		

Syntax:

Rs=FR (W=0)

FR=Rs (W=1)

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

2

Word length:

1

Version:

ISA 1.2 and above

Description:

Access the Flag Register (FR) value.

Example:

```
FR=R1;
R2=FR;
```

Push operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Rd			0	1	0	N			Rs		

Syntax:

Push R_H, R_L to [Rs]; Rd: R_H N: register counts from R_H to R_L

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

4+2*N

Word length:

1

Version:

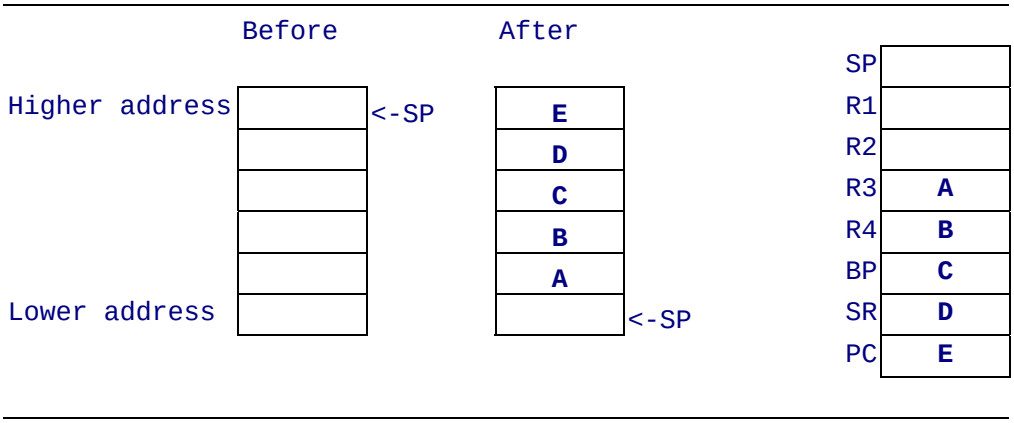
ALL

Description:

Push a set of registers (R_H ~ R_L) to memory location indicated by Rs consecutively.

Example:

Push PC, R3 to [SP]; // Push PC(R7) through R3, and N=5



Note:

Push R1, BP to [SP] is equivalent to Push BP, R1 to [SP].

Pop operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Rd			0	1	0	N			Rs		

Syntax:

Pop R_L, R_H from [Rs]; Rd : R_{L-1} N: register counts from R_L to R_H.

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

4+2*N

Word length:

1

Version:

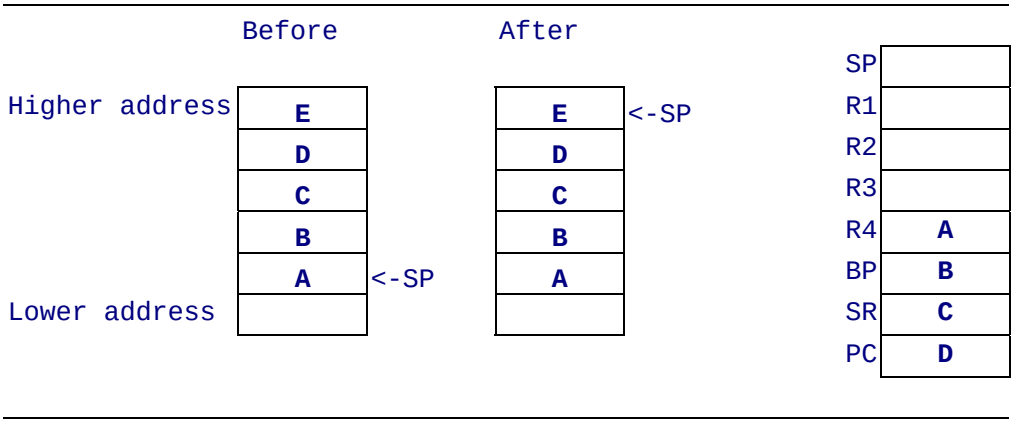
ALL

Description:

Pop a set of registers (R_L ~ R_H) from memory location indicated by Rs consecutively.

Example:

Pop P4, PC from [SP]; // Pop R4 through PC, and N=4



Function Call and Interrupt

CALL function with 22-bits indirect address in MR

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	1	-	-	0	0	1

Syntax:

CALL MR;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

8

Word length:

1

Version:

ISA 1.2 and above

Description:

This is a far function call instruction with MR register. The 22-bits content of MR {R4[5:0], R3} will be used as destination address. PC and SR will be pushed to memory location indexed by SP and SP, CS will be updated automatically after this operation.

Example:

```
CALL MR;          // Push PC and SR, then jump to {R4[5:0],R3}
```

CALL function with 22-bit immediate address

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	0	0	1	A22[21:16]					

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
A22[15:0]															

Syntax:

CALL A22;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

9

Word length:

2

Version:

ALL

Description:

This is a far function call instruction with 22-bits immediate address. Both PC and SR will be pushed to memory indexed by SP and SP, CS will be updated automatically after this operation.

Example:

CALL 0x12345;

RETF operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	0	1	0	0	1	0	0	0	0

Syntax:

RETF;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

8

Word length:

1

Version:

ALL

Description:

Return from subroutine instruction. The SR and PC will be popped from memory location indexed by SP, and return to the calling function.

Example:

retf;

RETI operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	0	1	0	0	1	1	0	0	0

Syntax:

RETI;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

8 (IRQ NEST OFF) / 10 (IRQ NEST ON)

Word length:

1

Version:

ALL

Description:

Return from interrupt service routine, if the IRQ Nest Mode (INE) is ON and cpu is executing IRQ service routine, the FR, SR, PC will be popped from memory location indexed by SP and return to the interrupted program. Else only the SR, PC will be popped and return to the interrupted program. After this instruction the BRK, FIQ, IRQ servicing flag inside cpu will be cleared according to priorities.

Example:

reti;

Software interrupt operation

Machine Code Format:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1	1	1	1	-	-	-	1	0	1	1	-	-	0	0	0

Syntax:

BREAK;

Conditional Flag:

N	Z	S	C
-	-	-	-

Cycles:

10

Word length:

1

Version:

ALL

Description:

This is a software interrupt instruction (SWI). CPU will interrupt current program executing sequence, save the PC, SR to memory location indexed by SP and jump to the BREAK service routine which address stored in memory location [0x00fff5].

Example:

break;

Instruction Set Summary

Syntax	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SECBANK ON/OFF	1	1	1	1	-	-	-	1	0	1	0	0	1	0	1	BNK
FRACTION ON/OFF	1	1	1	1	-	-	-	1	0	1	0	0	0	1	1	FRA
FIR_MOV ON/OFF	1	1	1	1	-	-	-	1	0	1	0	0	0	1	0	FIR
FIQ ON/OFF	1	1	1	1	-	-	-	1	0	1	0	0	1	1	FIQ	0
IRQ ON/OFF	1	1	1	1	-	-	-	1	0	1	0	0	1	0	0	IRQ
INT FIQ/IRQ/OFF	1	1	1	1	-	-	-	1	0	1	0	0	0	0	FIQ	IRQ
IRQNEST ON/OFF	1	1	1	1	-	-	-	1	0	1	0	0	1	1	INE	1
Rd {ALU_OP}= IM6;	ALU_OP				Rd			0	0	1	IM6					
Rd= Rs {ALU_OP} IM16;	ALU_OP				Rd			1	0	0	0	0	1	Rs		
	IM16															
Rd {ALU_OP}= Rs {SFT_OP}{n};	ALU_OP				Rd			1	SFT_OP		n			Rs		
Rd {ALU_OP}= [A6] ;	ALU_OP				Rd			1	1	1	A6					
Rd= Rs {ALU_OP} [A16];	ALU_OP				Rd			1	0	0	0	1	W	Rs		
	A16															
Rd {ALU_OP}= {D:}[Rs@];	ALU_OP				Rd			0	1	1	D	@		Rs		
Rd {ALU_OP}= [BP+IM6];	ALU_OP				Rd			0	0	0	IM6					
BIT_OP Rd, Rs;	1	1	1	0	Rd			0	0	0	BIT_OP		0	Rs		
BIT_OP Rd, offset;	1	1	1	0	Rd			0	0	1	BIT_OP		offset			
BIT_OP {D:}[Rd], Rs;	1	1	1	0	Rd			1	0	D	BIT_OP		0	Rs		
BIT_OP {D:}[Rd], offset;	1	1	1	0	Rd			1	1	D	BIT_OP		offset			
Rd= Rd LSFT_OP Rs;	1	1	1	0	Rd			1	0	LSFT_OP			1	Rs		
MR= Rd*Rs, {ss/us/uu};	1	1	1	S	Rd			S	0	0	0	0	1	Rs		
MR= [Rd]*[Rs], {ss/us/uu}, N;	1	1	1	S	Rd			S	1	N				Rs		
DIVQ MR, R2;	1	1	1	1	-	-	-	1	0	1	1	-	-	0	1	1
DIVS MR, R2;	1	1	1	1	-	-	-	1	0	1	1	-	-	0	1	0
EXP R2, R4;	1	1	1	1	-	-	-	1	0	1	1	-	-	1	0	0
NOP	1	1	1	1	-	-	-	1	0	1	1	-	-	1	0	1
BRANCH_OP IM6	BRANCH_OP				1	1	1	0	0	D	IM6					
GOTO MR;	1	1	1	1	1	1	1	0	1	1	-	-	-	-	-	-
GOTO A22;	1	1	1	1	1	1	1	0	1	0	A22[21:16]					
	A22[15:0]															
DS= IM6	1	1	1	1	1	1	1	0	0	0	IM6					
DS= Rs / Rs= DS	1	1	1	1	-	-	-	0	0	0	1	0	W	Rs		
FR= Rs / Rs= FR	1	1	1	1	-	-	-	0	0	0	1	1	W	Rs		
Push R_H, R_L to [Rs];	1	1	0	1	Rd			0	1	0	N			Rs		
Pop R_L, R_H from [Rs]	1	0	0	1	Rd			0	1	0	N			Rs		
CALL MR;	1	1	1	1	-	-	-	1	0	1	1	-	-	0	0	1
CALL A22;	1	1	1	1	-	-	-	0	0	1	A22[21:16]					
	A22[15:0]															
RETF	1	0	0	1	1	0	1	0	1	0	0	1	0	0	0	0
RETI;	1	0	0	1	1	0	1	0	1	0	0	1	1	0	0	0
BREAK;	1	1	1	1	-	-	-	1	0	1	1	-	-	0	0	0

Appendix A : CPU Cycle Formula

Class	Instruction	Code Width (unit: word)	CPU Cycle Formula (if Rd is NOT pc)	Example (SPCE060)		Example Condition
				Program Memory	1	
				Data Memory	1	
				Counter(N)	5	
Control Flow						
	CALL	2	9+2*PM+2*DM♦	13		call func_1
	RETF	1	8+PM+2*DM	11		
	RETI (IRQ, FIQ, BRK)	1	8+PM+2*DM	11		
	RETI (Nested IRQ ON)	1	10+PM+3*DM	14		
	BREAK	1	10+2*PM+2*DM	14		
	GOTO	2	5+2*PM	7		
	JUMP (non-taken)	1	2+PM	3		
	JUMP (taken)	1	4+2*PM	6		
	GOTO MR	1	4+2*PM	6		
	CALL MR	1	8+2*PM+2*DM	12		
	NOP	1	2+PM	3		
Operation Mode						
	INT FIQ, IRQ	1	2+PM	3		
	INT OFF	1	2+PM	3		
	INT FIQ	1	2+PM	3		
	INT IRQ	1	2+PM	3		
	IRQ ON/OFF	1	2+PM	3		
	FIQ ON/OFF	1	2+PM	3		
	FIR_MOV ON/OFF	1	2+PM	3		
	FRACTION ON/OFF	1	2+PM	3		
	SECBANK ON/OFF	1	2+PM	3		
	IRQNEST ON/OFF	1	2+PM	3		
Push/ Pop						
	PUSH R_H, R_L to [Rs]	1	4+2*N+N*DM+PM	20		PUSH R5, R1 to [SP]; SP point to DM
	POP R_H, R_L to [Rs]	1	4+2*N+N*DM+PM	20		POP R1, R5 from [SP]; SP point to DM
Multiplication						

♦ PM denotes the waiting cycle from program memory, and DM denotes the waiting cycle from data memory.

	MR= Rd* Rs,{ss,us,uu}	1	13+PM	14	MR=R1*R2
	MR= [Rd]*[Rs],{ss,us,uu},N	1	6+10*N+N*(PM+DM)+PM	67	MR=[R1]*[R2], 5;
Division					
	DIVS MR,R2	1	2+PM	3	DIVS MR,R2
	DIVQ MR,R2	1	3+PM	4	DIVQ MR,R2
Exponential Detect					
	R2=EXP R4	1	2+PM	3	R2=EXP R4
Shift Operation					
	Rd=Rd SFT Rs	1	8+PM	9	R1=R1 ASR R2
Bit Operation					
	bitop Rd, Rs	1	4+PM	5	TSTB R1,R2
	bitop Rd,offset	1	4+PM	5	SETB R2,0x3
	bitop D:[Rd], Rs	1	7+PM+2*DM	10	CLRB [R1],R3
	bitop D:[Rd],offset	1	7+PM+2*DM	10	INVB [R2],0x4
Flag Setting					
	DS=Rs / Rs=DS	1	2+PM	3	
	FR=Rs / Rs=FR	1	2+PM	3	
ALU Operation					
	Rd= Rd op [BP+IM6]	1	6+PM+DM	8	R1=R1+[BP+3]
	Rd=Rd op D:[Rs@]	1	6+PM+DM	8	R3=R3+D:[R1++]
	Rd = Rd op Rs SFT N	1	3+PM	4	R2=R2-R3 ASR 2
	Rd=Rd op IM6	1	2+PM	3	R1=R1+0x8
	Rd=Rd op IM16	2	4+2*PM	6	R2=R1+0x5678
	Rd=Rd op [A6]	1	5+PM+DM	7	R3=[0x20]
	Rd=Rs op [A16]	2	7+2*PM+DM	10	R4=[0x7600]